International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-7 Issue-6, August 2018

# Design of Software Testing Model Based on UML Class and Activity Diagram

**Suman Rani, Mukesh Rana**

***Abstract:** Quality software can be developed when it is properly tested. Due to increase in the size and complexity of object-oriented software, manual testing has become time, resource and cost consuming. Properly designed test cases discover more errors and bugs present in the software. The test cases can be generated much early in the software development process, during the design phase. The unified modeling language (UML) is the most widely used language to describe the analysis and designs of object-oriented software. Test cases can be derived from UML models more efficiently. In our work, we propose a novel approach for automatic test case generation from the combination of UML class diagrams. In our approach, we first draw the UML class diagrams using any online drawing tool like smart draw or yuml.com. Then, we generate XML information of these models. The XML file is processed to extract variables from the class and predicates from class diagram using Java code. The predicates are then used to generate the test cases. Our approach achieves 100% branch coverage and suitable for mutation testing and unit testing.*

***Keywords:** Quality Software, (UML), XML file, Manual Testing Has Become Time,*

## I. INTRODUCTION

Software testing usually involves executing a program on a set of tests and comparing the expected output with the actual output. Testing is done to find the errors, which may latter cause system failure. Now-a-days most of the project is developed in object-oriented language. These object-oriented languages are quite large and complex in nature because of its features, like encapsulation, inheritance, polymorphism. Automatically generating test cases from code of object-oriented programs are very much difficult because of its feature like encapsulation, inheritance, polymorphism, etc. The unified modeling language (UML) is most widely used language to model object-oriented designs. UML diagrams an important source of test case generation. In the recent days, researchers have considered different UML diagrams for generating test cases. Identifying error early during the design phase is much more efficient than identifying error after developing code.

## II. LITERATURE REVIEW

Wang et al. [1] proposed an approach to generate the test case from an interaction and class diagrams. They used test adequacy criteria for the coverage of the design model elements. They have adopted the category partition approach to get the function units, then for each function unit, generate test cases from class diagram criteria. An intermediate graph from the UML diagrams is often required to derive the test case. The testing technique described in this paper is based on model elements, not works on intermediate graph. A method is introduced by Asthana et al. [2] for generating test cases using class and sequence diagrams. First, they get the lower and upper bound of variable from the given class diagram. Then, they have traversed the sequence diagram to obtain the all variable passed. Out of these variables, they found out the variables on which the output will derive and have applied robustness testing on these variables to compare the results. In this approach, user has to specify model information in XMI format. They have automated the process by parsing xml metadata interchange (XMI). So this approach can be used if our diagram information is in XMI format only. A method is proposed by Swain et al. [3] to generate test case based on use case and sequence diagram. They constructed Concurrent Control Flow Graph (CCFG) from sequence diagrams and Use case Dependency Graph (UDG) from use case diagram to generate test sequence. They have used UML 2.0 sequence diagram for generating test cases. They have developed a semi automated tool (Com Test) which takes XMI representation of sequence diagram as input and generate the test cases. Their testing strategies to derive test cases use full predicate coverage criteria. The generated test cases are suitable for detecting dependency of use cases and synchronization and messages, object interaction and operational faults. This work includes the test oracles and handling synchronizations in the CCFP. Also, this work does not generate test data. A methodology is proposed by Swain et al. [4] to prioritize test scenario from UML communication and activity diagrams. They presented an integrated approach and a prioritization technique to generate cluster-level test scenarios from UML communication and activity diagrams. First, they convert the communication and activity diagrams into a tree representation respectively. Then combines the tree representation of diagrams into intermediate tree named as COMMACT tree. The COMMACT tree is then traversed to generate the test scenarios. They have proposed a prioritization metric considering the coupling or impact or influence of activity and methods. They considered the criticality of guard conditions to perform those activities and methods. Their approach generates prioritized test scenarios and test scenarios are not redundant.

**Manuscript published on 30 August 2018.**
***** Correspondence Author (s)

**Suman Rani,** Research Scholar, HCTM Technical Campus, Kaithal (Haryana), India.
**Dr. Mukesh Rana,** Assistant Professor, HCTM Technical Campus, Kaithal (Haryana), India.

In this approach, Communication Diagram or Activity diagrams are to be converted in to Tree Structure which is not feasible in all situations. Pilskalns et al. [5] presented a graph based approach to combine the information form sequence diagrams and class diagrams. In this approach, first sequence diagram is transformed into an object-method directed acyclic graph (OMDAG). The values of variable in class diagram are then associated with objects in OMDAG during path traversal. The execution sequence and attribute value of generated test cases is stored into an object method execution table (OMET). This approach requires both class diagrams & sequence diagrams, and then it is to be transformed in to Graph structure. If we have only one type of diagrams, it does not work.

In [6] Jeevarathinam and Antony Selvadoss Thanama-ni have proposed automation generation of test cases from software specification. Test inputs can be derived by two major approaches as static approach, in which inputs can be generated from models of the system, where as dynamic approach generates tests by repeatedly executing the program. They presented a framework for testing based on automated test case generation using input specification. The frameworks combine symbolic execution and model checking techniques for the verification of java program. Test coverage criteria used is branch coverage criteria.

## III. PROPOSED APPROACH

In this section, we discuss our proposed approach to generate test scenarios from class and activity diagrams. The four basic steps in our approach are

1. Construct activity and class diagram.
2. Generate the XML code.
3. Parse the XML code.
4. Apply our proposed algorithm.
5. Calculate the decision coverage.

<u>Construct activity and class diagram:</u>

Initially the activity diagram and class diagram are constructed using IBM Rational Software Architect (RSA). RSA is most widely used tool to construct the UML diagrams. Other tools like smartdraw or yuml.com may also be used

<u>Generate the XML code:</u>

Next, we define the XML code of the relevant diagrams draw in step 1. XML format is the standard format used for UML diagrams portability across different object-oriented software. The XML file contains all the necessary information needed to inter-operate between different tools and transfer UML diagrams.

<u>Parse the XML code:</u>

We develop a parser in java language which extract the necessary information such as name of attribute present in the class diagram, predicates present in activity diagram from the XML code. This parser traverse the XML file in sequential order. For each class, obtain the attributes present in the class.

All these class attribute and predicate is stored in attribute array and decision array respectively. These two arrays are given as input to our proposed algorithm. The total number of decision node in stored in variable to talnumber of branch and the output is stored in output array.

Proposed algorithm:

To implement our proposed work, we developed algorithm Integrated test case generation algorithm (ITCGA). Detail of ITCGA is discussed in section 4.4.

Calculate the decision coverage: The formula for calculating the decision coverage is:

decision coverage=(number of decision outcome exercised/(number of decision outcome))*100%

Each decision node in activity diagram takes two possible outcome. So, total number of decision outcome=number of decision node*2. Total number of decision outcome exercised is obtained by Algorithm.

## IV. PROPOSED ALGORITHM

In this section, we present our ITCGA algorithm 1 to generate test cases, in pseudo code form.

### A. Description of Algorithm

Input to our Algorithm 1 is attribute array, decision array. Output of the Algorithm is set of test cases and total number of branch covered. At the beginning of algorithm 1, the testcaseid and branchcovered variable are initialized to null. The Algorithm traverse the decisionarray in sequencial order. We have considered the binary relational operator such as ==, ! =, >, <, >=, <= i.e., each operator has two operands. The left operand is extracted and it is checked whether it is present in decisionarray or not. If it is not present the algorithm display the message that operand is not present in any class, otherwise test case is generated. The left operand of an operator is variable and the right operand is an integer value. Right operand is assigned to testinput variable and the value of testcaseid is incremented by 1. Left operand is assigned to testcondition variable. Next, algorithm check for the type of operator. If the operator is >= the algorithm print the two test case as (testcaseid, testcondtion, testinput, expected output).Other test case is (testcaseid, testcondtion, testinput+1, expected output). If the operator is > the algorithm print the one test case as (testcaseid, testcondtion, testinput+1, expected output). If the operator is <= the algorithm print the two test case as (testcaseid, testcondtion, testinput, expected output). Other test case is (testcaseid, testcondtion, testinput-1, expected output). If the operator is < the algorithm print the one test case as (testcaseid, testcondtion, testinput-1, expected output). If the operator is != the algorithm print the one test case as (testcaseid, testcondtion, testinput-1, expected output). Other test case is (testcaseid, testcondtion, testinput+1, expected output). If the operator is == the algorithm print the two test case as (testcaseid, testcondtion, testinput, expected output). The process is repeated until decisionarray is empty.

### B. Algorithm 1 Integrated Test Case Generation Algorithm (ITCGA)

```
Input: attributearray, decisionarray, outputarray
Output: set of test cases, number of branch covered
    testcaseid=
    branchcovered=
    testinput=
    testcondition=
operatorsymbol=
for each elementj in decisionarray do
testcondition= leftoperand
if testcondition present in attributearray then
operatorsymbol= operator
testinput= rightoperand
if operatorsymbol==’>=’ then
branchcovered= branchcovered + 1
print (testcaseidi, testcondition, testinput, outputarrayj )
testcaseid= testcaseid + 1
print (testcaseidi, testcondition, testinput+1, outputarrayj )
else
if operatorsymbol==’>’ then
branchcovered =branchcovered + 1
testcaseid =testcaseid + 1
print (testcaseidi, testcondition, testinput+1, outputarrayj )
end if
else
if operatorsymbol==’<’ then
branchcovered =branchcovered + 1
testcaseid =testcaseid + 1
print (testcaseidi, testcondition, testinput-1, outputarrayj )
end if
else
if operatorsymbol==’<=’ then
branchcovered =branchcovered + 1
testcaseid =testcaseid + 1
print (testcaseidi, testcondition, testinput, outputarrayj )
testcaseid =testcaseid + 1
print (testcaseidi, testcondition, testinput-1, outputarrayj )
end if
else
if operatorsymbol== ’=’ then
branchcovered =branchcovered + 1
testcaseid =testcaseid + 1
print (testcaseidi, testcondition, testinput, outputarrayj )
testcaseid =testcaseid + 1
end if
else
if operatorsymbol== ‘!=’ then
branchcovered =branchcovered + 1
testcaseid =testcaseid + 1
print (testcaseidi, testcondition, testinput-1, outputarrayj )
print (testcaseidi, testcondition, testinput+1, outputarrayj )
end if
end if
else
Print testcondition not present in any class
end if
end for
```

## C. Case Study

We consider the example of railway reservation system to discuss our proposed approach. We model the class diagram and activity diagram of railway reservation system which is shown in Figure 1 and Figure 2 respectively. Figure 1 represents the class diagram of railway reservation system. There are five classes named as; train, bank account, passenger, ticket, railwaysystem. Each class has three parts. There are five attributes in train class; trainno, trainname, numberofseatleft, source and destination. The train class does not perform any operation so, it does not contain any method. There are four attribute in the passenger class; name, age, gender, contact number. The passenger can book and.

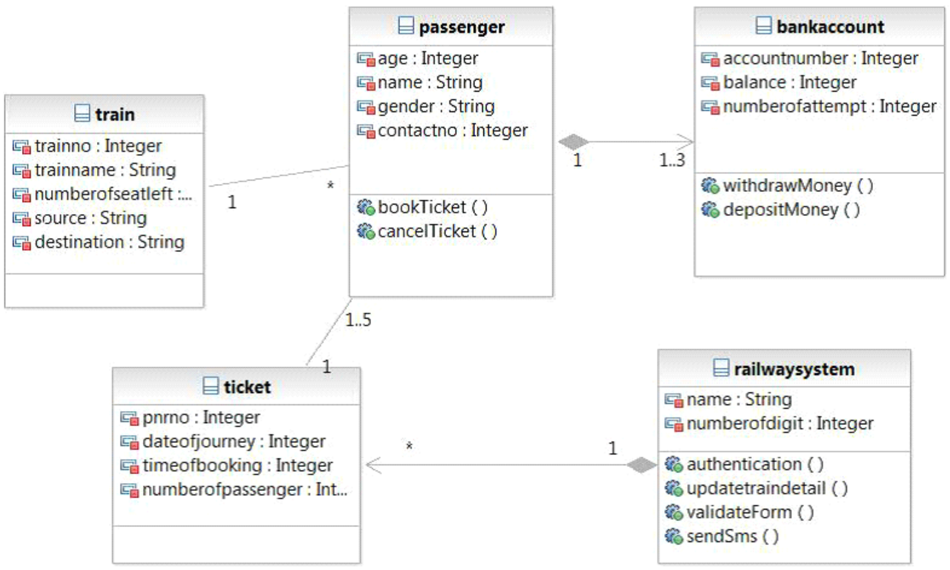


**Figure 1: Class Diagram of Railway Reservation System**

Cancel ticket via book Ticket and the cancel Ticket method in passenger class. The ticket class has four attributes; pnrno, date of journey, time of booking, number of passenger. Similary the bank account and railway system have their corresponding attribute and method as given in _gure 1. The link between train and passenger class is 1 to * association. A train can have any number of passengers. The link between ticket and passenger is 1 to 1..5 association, i.e. With a single ticket maximum _ve people can travel. The link between passenger and bank account is 1 to 1..3 composition.

A passenger can have three bank account and composition relationship indicates that when a passenger does not exist his bank account will also not exist. Similarly, link between ticket and the railway system is * to 1 composition.

Figure 4.2 shows the activity diagram for railway reservation system. First user visits the website. Then the user enters form and to station code and enter the book ticket button. Now the system check in time of booking, in case it is greater than or equal to 8, the system will ask for entering the date of journey; otherwise the system displays an error message to the user that Inter incoming day booking not allowed before 8 AM. The user then enters the date of journey. Now the system check for date of journey, in case it is below or equal to 60 days the system shows the list of trains; otherwise it shows an error message that date of journey is beyond the advance reservation period.

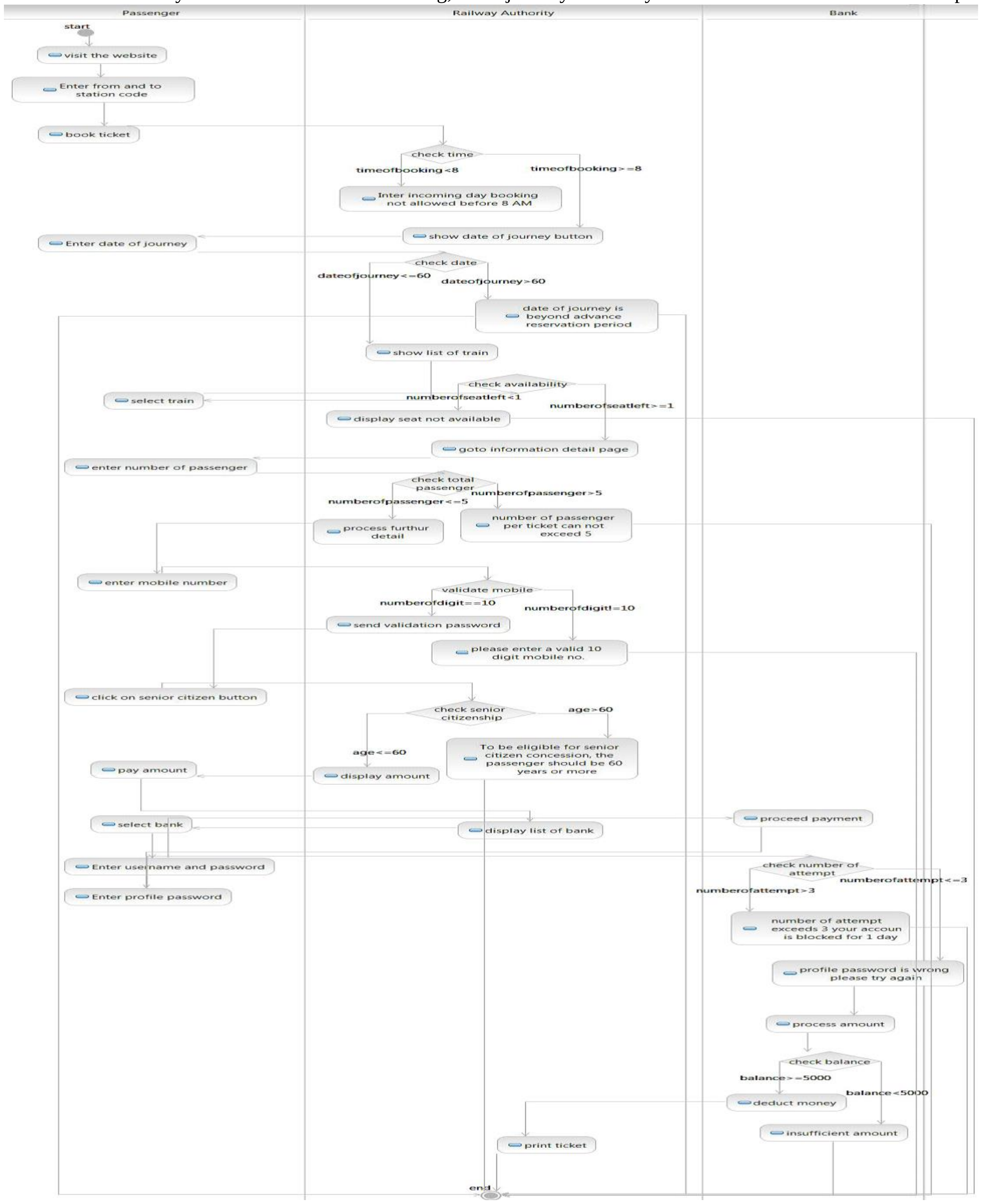


**Figure 2: Activity Diagram of Railway Reservation System**

The user then selects one of the trains. Next, the system check for whether the seat is available or not; in case seat is available, the system display an information detail form; otherwise the system display a message to the user that seat not available. The user then enters the number of passengers. The system checks for number of people travelling if it is greater than or equal to 5; the system displays the error message that number of passengers per ticket cannot exceed 5; otherwise the system will ask for entering the mobile number. The user then enters the mobile number. The system check for number of digits in mobile number; in case number of digits is equal to 10 it send the validation code; otherwise the system displays an error message that please enter the 10 digit number. The system checks for senior citizens; if the age is greater than 60 then it will show to be eligible for senior citizen concession, the passenger should be 60 years or more error message; otherwise the system will display the amount.
The user chooses to pay amount and the system will display the list of banks. The user chooses one of the bank and enter username name and password. User proceed further and enter the profile password. If the user enters the wrong password an error message will be displayed that profile password is wrong, please try again. The number of attempts should not be greater than 3, if it exceeds 3 the system will display number of attempt exceeds 3 your account is blocked for 1 day. In order to generate the ticket total balance should then be greater than the amount; otherwise insufficient amount message is displayed. The transfer of control is represented by swimlanes. Three swimlane are required to represent the control flow, worst for passenger, second for railway authority, third for the bank.

## V. WORKING OF ALGORITHM

The Algorithm 1 begin by reading the element of decisionarray in sequential order.First element is the predicate timeofbooking<8. leftoperand timeofbooking is assigned to variable testcondition, rightoperand 8 is assigned to variable testinput and the operator < is assigned to operatorsymbol. Now algorithm check whether timeofbooking is present in attributearray or not. since it is present the if condition is satisfied and it check for type of operator. since the operator symbol is < it matches the appropriate if...elseif....else statement and output is printed as (1, timeofbooking, 7, Inter incoming day booking not allowed befor 8 AM). The value of variable branchcovered and testcaseid is increment by 1.
The algorithm then proceed to next element which is found to be timeofbooking>=8. leftoperand timeofbooking is assigned to variable testcondition, rightoperand 8 is assigned to variable testinput and the operator >= is assigned to operatorsymbol.Now algorithm check whether timeofbooking is present in attributearray or not. since it is present the if condition is satisfied and it check for type of operator. since the operator symbol is >= it matches the appropriate if...elseif....else statement. Output printed is (2, timeofbooking, 8, show date of journey button) and (2, timeofbooking, 9, show date of journey button). The value of variable branchcovered is increment by 1 and testcaseid is incremented by 2.

Above process is repeated until the decisionarray becomes empty. Overall test cases generated is shown in table 1 and the value of branchcovered becomes 16.Next, decision coverage is calculated in following way: since there is only one condition in decision node, there is only two possible outcome one is true and other is false.
number of decision node are: 8
number of decision outcome: 8*2
number of decision outcome excersized: 16
branch coverage=(number of decision outcome exercized/(number of decision outcomes))*100
branch coverage=100%

**Table 1: Table Showing Test Input with Expected Output**

| testcaseid | testcondition | input | expected output |
|---|---|---|---|
| 1 | timeofbooking | 7 | Inter incoming day booking not allowed before 8 AM |
| 2 | timeofbooking | 8 | show date of journey button |
| 3 | timeofbooking | 9 | show date of journey button |
| 4 | dateofjourney | 61 | date of journey is beyond advance reservation period |
| 5 | dateofjourney | 60 | show list of train |
| 6 | dateofjourney | 59 | show list of train |
| 7 | numberofseatleft | 0 | display seat not available |
| 8 | numberofseatleft | 1 | goto information detail page |
| 9 | numberofseatleft | 2 | goto information detail page |
| 10 | age | 61 | To be eligible for senior citizen concession, the passenger should be 60 years or more |
| 11 | age | 60 | display amount |
| 12 | age | 59 | display amount |
| 13 | numberofattempt | 4 | number of attempt exceeds 3 your account is blocked for 1 day |
| 14 | numberofattempt | 3 | profile password is wrong please try again |
| 15 | numberofattempt | 2 | profile password is wrong please try again |
| 16 | balance | 5000 | deduct money |
| 17 | balance | 5001 | deduct money |
| 18 | balance | 4999 | insufficient amount |
| 19 | numberofdigit | 9 | please enter a valid 10 digit mobile no. |
| 20 | numberofdigit | 11 | please enter a valid 10 digit mobile no. |
| 21 | numberofdigit | 10 | send validation password |
| 22 | numberofpassenger | 6 | number of passenger per ticket can not exceed 5 |
| 23 | numberofpassenger | 5 | process furthur detail |
| 24 | numberofpassenger | 4 | process furthur detail |

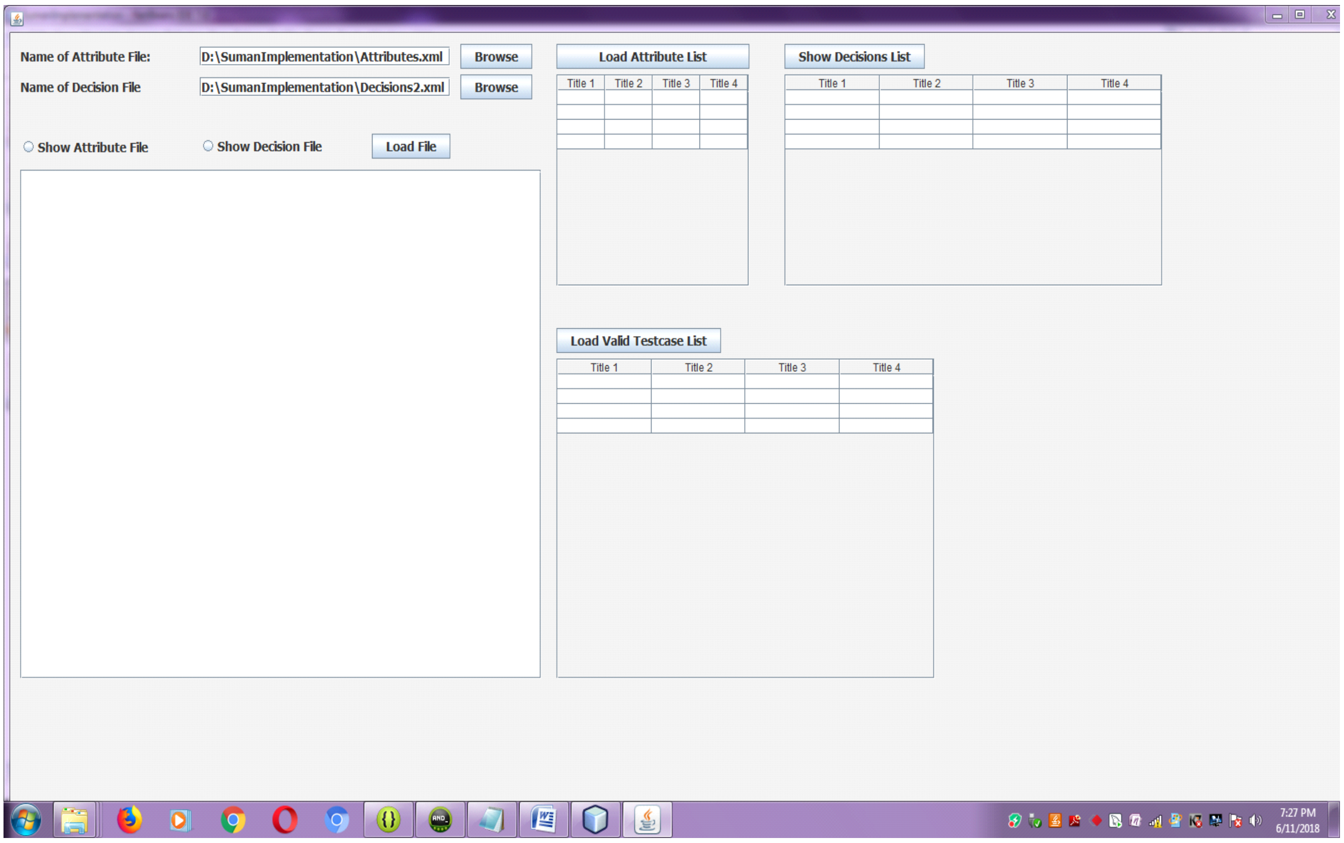
Table 2 show the result of actual operator and its mutants. The actual operator is timeofbooking<8 and its mutants are timeofbooking>8, timeofbooking<=8, timeofbooking>=8, timeofbooking==8, timeofbooking! =8.

**Table 2: Outcome of Different Possible Relational Operator Mutant**

| testcaseid | actual operator | mutant | mutant | mutant | mutant | mutant |
|---|---|---|---|---|---|---|
| | *timeofbooking* $<$ *8* | *timeofbooking* $>$ *8* | *timeofbooking* $\leq$ *8* | timeofbooking$\geq$8 | *timeofbooking* $=$ *8* | *timeofbooking* $\neq$ *8* |
| 1 | T | F | T | F | F | T |
| 2 | F | F | T | T | T | F |
| 3 | F | T | F | T | F | T |

## A. Implementation

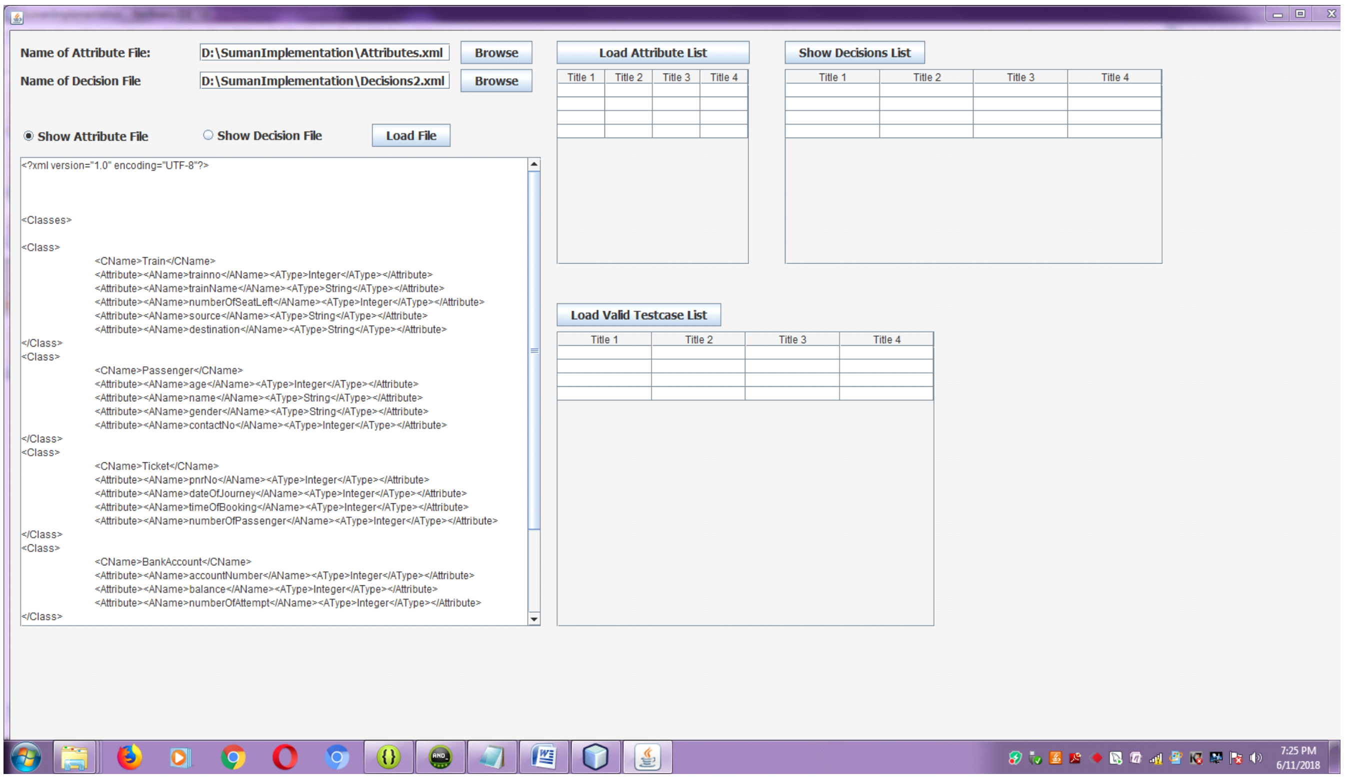
In this section we describe the implementation details of our system. Whole system is implemented using java language and Net Beans IDE. To run this system, information of class diagram and activity diagrams must be specified in to XML format.


Name of Attribute File: D:\SumanImplementation\Attributes.xml

Name of Decision File D:\SumanImplementation\Decisions2.xml

Show Attribute File [ ] Show Decision File [ ]

**Snapshot 3: Specify Attribute and Decision Files**


Name of Attribute File: D:\SumanImplementation\Attributes.xml

Name of Decision File D:\SumanImplementation\Decisions2.xml

[✓ Show Attribute File] Show Decision File

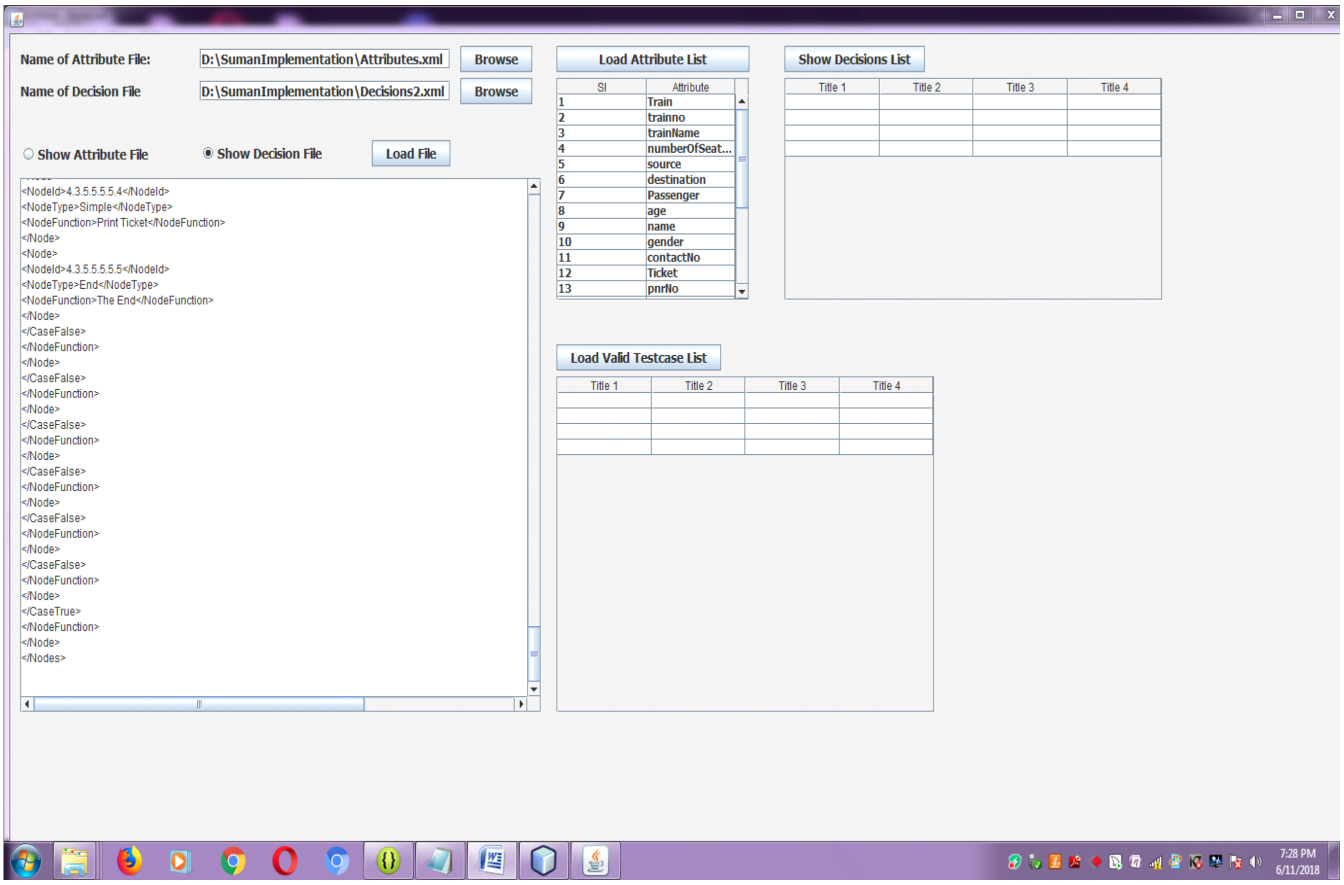
```
<?xml version="1.0" encoding="UTF-8"?>



<Classes>

<Class>
          <CName>Train</CName>
          <Attribute><AName>trainno</AName><AType>Integer</AType></Attribute>
          <Attribute><AName>trainName</AName><AType>String</AType></Attribute>
          <Attribute><AName>numberOfSeatLeft</AName><AType>Integer</AType></Attribute>
          <Attribute><AName>source</AName><AType>String</AType></Attribute>
          <Attribute><AName>destination</AName><AType>String</AType></Attribute>
</Class>
<Class>
          <CName>Passenger</CName>
          <Attribute><AName>age</AName><AType>Integer</AType></Attribute>
          <Attribute><AName>name</AName><AType>String</AType></Attribute>
          <Attribute><AName>gender</AName><AType>String</AType></Attribute>
          <Attribute><AName>contactNo</AName><AType>Integer</AType></Attribute>
</Class>
<Class>
          <CName>Ticket</CName>
          <Attribute><AName>pnrNo</AName><AType>Integer</AType></Attribute>
          <Attribute><AName>dateOfJourney</AName><AType>Integer</AType></Attribute>
          <Attribute><AName>timeOfBooking</AName><AType>Integer</AType></Attribute>
          <Attribute><AName>numberOfPassenger</AName><AType>Integer</AType></Attribute>
</Class>
<Class>
          <CName>BankAccount</CName>
          <Attribute><AName>accountNumber</AName><AType>Integer</AType></Attribute>
          <Attribute><AName>balance</AName><AType>Integer</AType></Attribute>
          <Attribute><AName>numberOfAttempt</AName><AType>Integer</AType></Attribute>
</Class>
```

**Snapshot 4: Show Attribute File**


Name of Attribute File: D:\SumanImplementation\Attributes.xml Browse
Name of Decision File D:\SumanImplementation\Decisions2.xml Browse

Show Attribute File [✓ Show Decision File] Load File

```
<NodeId>4.3.5.5.5.5.4</NodeId>
<NodeType>Simple</NodeType>
<NodeFunction>Print Ticket</NodeFunction>
</Node>
<Node>
<NodeId>4.3.5.5.5.5.5</NodeId>
<NodeType>End</NodeType>
<NodeFunction>The End</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseTrue>
</NodeFunction>
</Node>
</Nodes>
```

Load Attribute List

| SI | Attribute |
|---|---|
| 1 | Train |
| 2 | trainno |
| 3 | trainName |
| 4 | numberOfSeat... |
| 5 | source |
| 6 | destination |
| 7 | Passenger |
| 8 | age |
| 9 | name |
| 10 | gender |
| 11 | contactNo |
| 12 | Ticket |
| 13 | pnrNo |

Show Decisions List

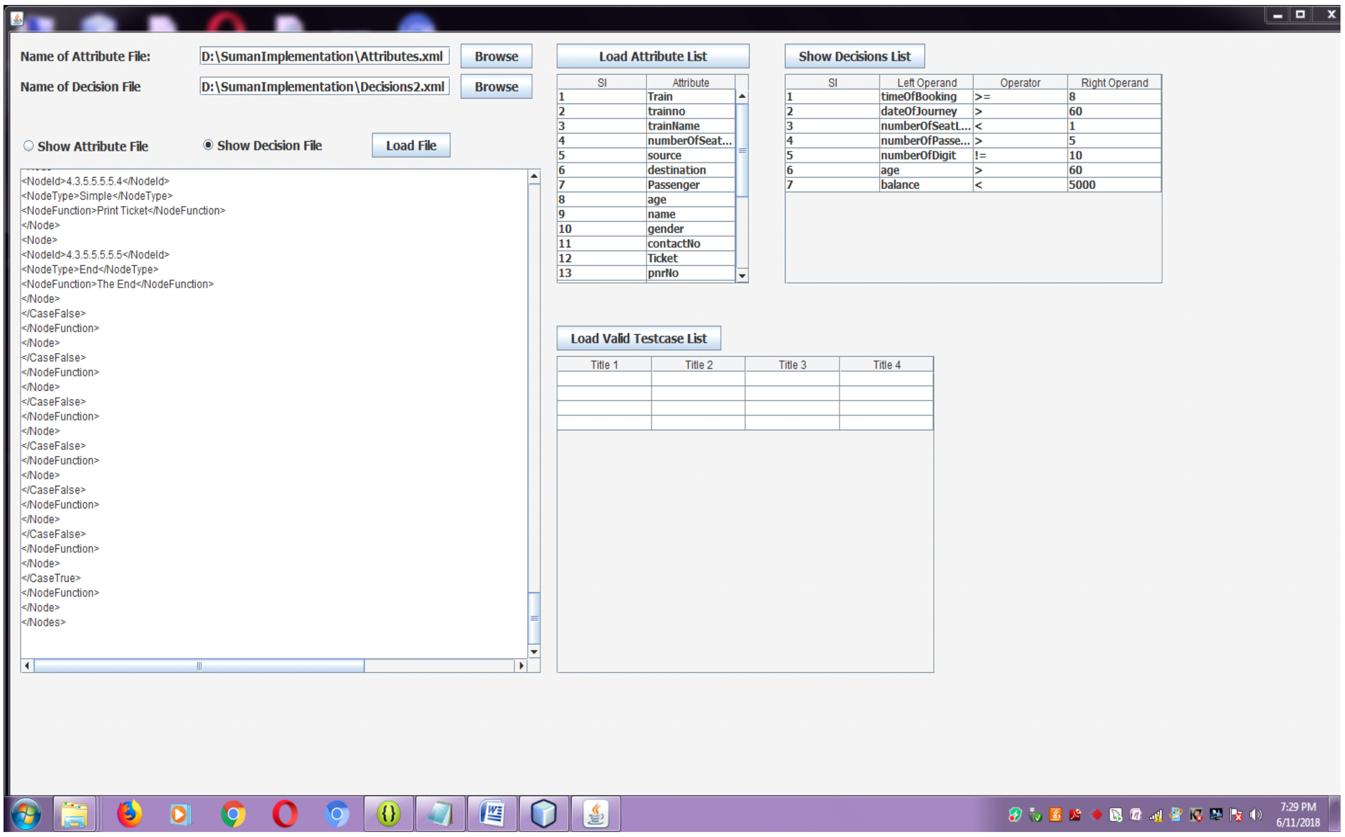
| Title 1 | Title 2 | Title 3 | Title 4 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

Load Valid Testcase List

| Title 1 | Title 2 | Title 3 | Title 4 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

7:28 PM 6/11/2018

Snapshot 6: Load Attribute List


Name of Attribute File: D:\SumanImplementation\Attributes.xml Browse
Name of Decision File D:\SumanImplementation\Decisions2.xml Browse

Show Attribute File [✓ Show Decision File] Load File

```
<NodeId>4.3.5.5.5.5.4</NodeId>
<NodeType>Simple</NodeType>
<NodeFunction>Print Ticket</NodeFunction>
</Node>
<Node>
<NodeId>4.3.5.5.5.5.5</NodeId>
<NodeType>End</NodeType>
<NodeFunction>The End</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseTrue>
</NodeFunction>
</Node>
</Nodes>
```

Load Attribute List

| SI | Attribute |
|---|---|
| 1 | Train |
| 2 | trainno |
| 3 | trainName |
| 4 | numberOfSeat... |
| 5 | source |
| 6 | destination |
| 7 | Passenger |
| 8 | age |
| 9 | name |
| 10 | gender |
| 11 | contactNo |
| 12 | Ticket |
| 13 | pnrNo |

Show Decisions List

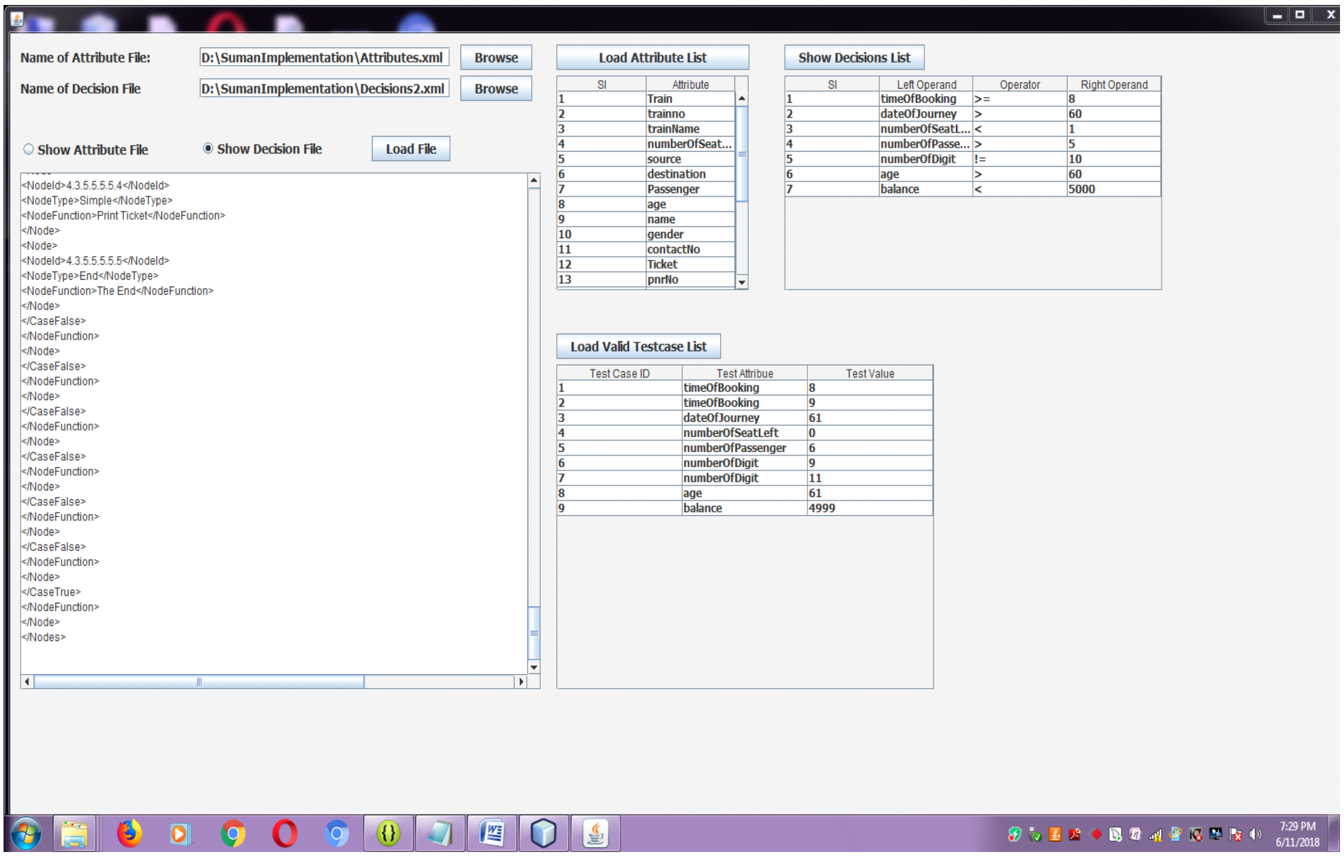
| SI | Left Operand | Operator | Right Operand |
|---|---|---|---|
| 1 | timeOfBooking | >= | 8 |
| 2 | dateOfJourney | > | 60 |
| 3 | numberOfSeatL... | < | 1 |
| 4 | numberOfPasse... | > | 5 |
| 5 | numberOfDigit | != | 10 |
| 6 | age | > | 60 |
| 7 | balance | < | 5000 |

Load Valid Testcase List

| Title 1 | Title 2 | Title 3 | Title 4 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

7:29 PM 6/11/2018

Snapshot 7: Load Decision List



```
<NodeId>4.3.5.5.5.5.4</NodeId>
<NodeType>Simple</NodeType>
<NodeFunction>Print Ticket</NodeFunction>
</Node>
<Node>
<NodeId>4.3.5.5.5.5.5</NodeId>
<NodeType>End</NodeType>
<NodeFunction>The End</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseFalse>
</NodeFunction>
</Node>
</CaseTrue>
</NodeFunction>
</Node>
</Nodes>
```

| Sl | Attribute |
|---|---|
| 1 | Train |
| 2 | trainno |
| 3 | trainName |
| 4 | numberOfSeat... |
| 5 | source |
| 6 | destination |
| 7 | Passenger |
| 8 | age |
| 9 | name |
| 10 | gender |
| 11 | contactNo |
| 12 | Ticket |
| 13 | pnrNo |

| Sl | Left Operand | Operator | Right Operand |
|---|---|---|---|
| 1 | timeOfBooking | >= | 8 |
| 2 | dateOfJourney | > | 60 |
| 3 | numberOfSeatL... | < | 1 |
| 4 | numberOfPasse... | > | 5 |
| 5 | numberOfDigit | != | 10 |
| 6 | age | > | 60 |
| 7 | balance | < | 5000 |

| Test Case ID | Test Attribue | Test Value |
|---|---|---|
| 1 | timeOfBooking | 8 |
| 2 | timeOfBooking | 9 |
| 3 | dateOfJourney | 61 |
| 4 | numberOfSeatLeft | 0 |
| 5 | numberOfPassenger | 6 |
| 6 | numberOfDigit | 9 |
| 7 | numberOfDigit | 11 |
| 8 | age | 61 |
| 9 | balance | 4999 |

**Snapshot 8: Load Valid Test Cases**

## VI. CONCLUSION AND FUTURE WORK

The major aim of our research was to automate the test cases generation from UML diagrams. Below, we summarize the important contributions of our work. At the end, some suggestions for future work are given.

In this research, we discussed a methodology to automatically generate test scenarios from the combination of class diagram and activity diagram. The proposed methodology is completely model-based and suitable for mutation testing. We developed a parser to generate the test cases automatically. We implemented the proposed algorithm Integrated test case generation algorithm (ITCGA). We have achieved the 100% branch coverage and the generated test cases are suitable for unit testing and mutation testing. The test cases are able to detect all the relational operators mutant. The generated test cases are not redundant. In the future, we further will generalize the approach by considering float, string data type. In most of the work generated test cases cannot be directly fed into the system under test (SUT). So, our next work is to propose a methodology to feed the test cases directly into the system under test (SUT).

## REFERENCE

1. S. K. Swain, S. K. Pani, and D. P. Mohapatra, "Model based object-oriented software testing.," Journal of Theoretical & Applied Information Technology, vol. 14, 2010.
2. M. Aggarwal and S. Sabharwal, "Test case generation from uml state machine diagram: A survey," in Computer and Communication Technology (ICCCT), 2012 Third International Conference on, pp. 133{140, IEEE, 2012.
3. R. K. Swain, V. Panthi, D. P. Mohapatra, and P. K. Behera, "Prioritizing test scenarios from uml communication and activity diagrams," Innovations in Systems and Software Engineering, pp. 1{16, 2013.
4. Y. Le Traon, T. Jeron, J.-M. Jezequel, and P. Morel, "Efficient object-oriented integration and regression testing," Reliability, IEEE Transactions on, vol. 49, no. 1, pp. 12{25, 2000.
5. V. Le Hanh, K. Akif, Y. Le Traon, and J.-M. Jezeque, "Selecting an efficient oo integration testing strategy: an experimental comparison of actual strategies," in ECOOP Object-Oriented Programming, pp. 381{401, Springer, 2001.
6. Jeevarathinam, Antony Selvadoss Thanamani. 2010. Towards Test Case Generation from Software Specification. Internation-al Journal of Engineering Science and Technology, Vol. 2(11), pp. 6578-6584.
7. Navnath Shete; Avinash Jadhav, "An empirical study of test cases in software testing" International Conference on Information Communication and Embedded Systems (ICICES2014), Year: 2014, Pages: 1 – 5
8. Muhammad Nomani Kabir; Jahan Ali; AbdulRahman A. Alsewari; Kamal Z. Zamli, "An adaptive flower pollination algorithm for software test suite minimization", 2017 3rd International Conference on Electrical Information and Communication Technology (EICT)Year: 2017, Pages: 1 - 5
9. Jai Gaur; Akshita Goyal; Tanupriya Choudhury; Sai Sabitha, "A walk through of software testing techniques", International Conference System Modeling & Advancement in Research Trends (SMART), Year: 2016, Pages: 103 – 108