Automatic Generation of Test Cases from UML Models

Vinaya Sawant
Lecturer
D.J. Sanghvi COE
Mumbai

Ketan Shah
Associate Professor
MPSTME
NMIMS University, Mumba

ABSTRACT
The paper presents a novel technique to create the test cases from UML models. In this technique, the UML diagrams such as Use Case Diagram, Class Diagram & Sequence Diagram of any application are considered for creating the test cases. A graph is created to store the necessary information that can be extracted from these diagrams & data dictionary expressed in OCL for the same application. The graph is then scanned to generate the test cases that are suitable for system testing.

Keywords
Testcases, UML Diagrams, OCL expressions

1. INTRODUCTION
Traditional testing has often generated tests from program source code, usually by abstracting the program into control flow diagrams, data flow graphs, call graphs, or other high level representations. Techniques to derive tests from formal specifications have also been developed. A more general term is that of *model-based testing*, which generally creates tests from an abstract model of the software, including formal specifications and semi-formal design descriptions such as UML diagrams. Automatically generating test cases directly from design models has several benefits:

1) Test-case generation is a time-consuming task and automating it saves resources.
2) The test cases can be generated before any code is written, which will allow developers to use the test cases as they develop the code. This reduces the number of iterations between development and testing, further saving resources.
3) One major cause of software related accidents occurs when requirements are miscommunicated to the developers or are not delivered to them at all.

Test cases generated directly from system requirements can be used to detect such errors, whereas most white-box test-generation algorithms would have no means of doing so[9]. The Unified Modeling Language (UML) is a collection of languages for specifying, visualizing, constructing, and documenting the artifacts of software systems [2]. Sequence
diagrams capture time dependent (temporal) sequences of interactions between objects. Sequence diagrams describe interactions among software components, and thus are naturally considered to be a good source for integration testing.
The aim of system testing is to ensure that a fully developed and integrated system is error free. System testing is often considered to be the most complex and intricate among all types of testing. In the grey box approach, system test cases are designed from design documents. In recent years, Unified Modeling Language (UML) [2] has emerged as the de facto standard for modeling software systems and has received significant attention from researchers as well as practitioners. The importance of UML models in designing test cases has been well recognized.
Model Based Testing (MBT) is gaining its popularity in both academia and in industry. As systems are increasing in complexity, more systems perform mission-critical functions, and dependability requirements such as safety, reliability, availability, and security are vital to the users of these systems.

2. PROBLEM DEFINITON
The aim is to develop an automatic test case generation tool using UML models. The sequence diagram is considered as a source of test case generation. The generated test suite aims to cover operational and use case dependency faults, various interaction as well as scenario faults. For generating the different components of a test case, i.e. input, expected output and pre- and post-condition, use case diagram, class diagram, data dictionary in the form of OCL expressions along with sequence diagram is considered. The generated test cases can be stored separately in a different file for future use.

3. THE PROPOSED APPROACH
Given a use case diagram (UD), class diagram (CD), sequence diagram (SD), transform it into a representation called sequence diagram graph (SDG) [1]. Each node in the SDG stores necessary information for test case generation. This information are collected from the use case template (also called extended use case), class diagrams, and data dictionary expressed in the form of object constrained language (OCL) [3], which are associated with the use case for which the sequence diagram is considered. Then traverse SDG and generate test cases based on a coverage criteria and a fault model. A schematic diagram of the approach is as shown in the fig 3.1[1].

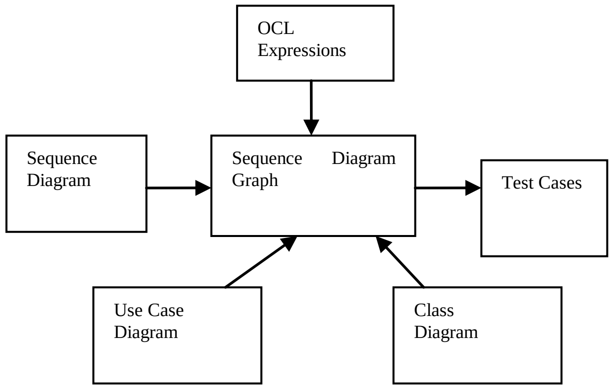

Fig 3.1: Schematic Block Diagram for the proposed approach

4. IMPLEMENTATION WORK

The different UML tools that such as MagicDraw and Rational Rose that has support for drawing UML diagrams as well as OCL expressions can be used to draw the UML diagrams according to the specification of any application that has to be considered. These UML diagrams are then exported to XML format. The process of exporting generates XML file from UML diagrams. This file contains all the XML tags describing all the UML diagrams. This XML file needs to be parsed to generate the graph. The parser is written that reads XML file and generate the different nodes for the graph by considering the sequence diagram of an application. These nodes are then mapped into the different scenarios according to the flow of messages in the sequence diagram. The nodes of the graph stores the information such as attributes of the corresponding objects at that state, arguments in the method, and predicate of the guard if any, involved in the interaction. The Use case template is also considered while generating the test cases. The Use Case template provides the information such as precondition & postcondition for a particular scenario that has to be considered. The OCL syntax will be followed to represent the data dictionary [3]. For the specification of a test case, the test specification language according to the IEEE Standard 829 is followed. Test cases generated will be recorded in a temporary file for future references. The case study for Bank ATM System is considered to generate the test cases. The Use Case Diagram, Class Diagram and Sequence Diagram is drawn which clearly gives the detailed description of the application. These diagrams are drawn using Magic Draw tool. Using the same tool, the diagrams are exported to XML format. The parser has been written in java that reads XML file and generates the different nodes of the graph. The sequence diagram is scanned to identify the set of the scenarios from the start node to end nodes. Now these set of scenarios along with Use Case Template and OCL data dictionary need to be traversed to generate the test cases. The generated test cases are recorded into a separate temporary file.

5. CASE STUDY: BANK ATM SYSTEM

The case study of Bank ATM system is considered to show the implementation work of the project. The Bank ATM system allows user to perform login with authenticated PIN number. If the user logs in with authorized PIN no, then the user is allowed to do deposit the money in his bank account, withdraw money from his bank account. He is able to check the balance and also can request for the mini statement. The following are the steps that need to be executed for generating test cases for the system from the UML Design Diagrams [6].

The implementation for this is done using JAVA programming language. The User Interface for the tool is provided so that on clicking on the desired buttons, the different steps needed to generate test cases can be easily done.

5.1 Step 1: Drawing UML Diagrams from the problem statement

The Use Case Diagram, Class Diagram and Sequence Diagram for the system are drawn to understand the system as a whole. The UML diagrams are drawn using the MagicDraw tool since this tool has the good support for OCL expressions as well as compared to any other tool. The fig 5.1 represents the Sequence Diagram for the Bank ATM System for withdraw money use case. The UseCase Diagram, Class Diagram with OCL expressions are also used for this tool.

5.2 Step 2: Generating XML File

The UML design tool has also the support for generating XML file from the UML diagrams which are easily transportable without installing the UML design tool. The MagicDraw Software has the option of exporting the UML diagram to XML file. The XML file thus generated contains the tags for all the properties of the diagrams including the color properties that are not required for generating the test cases. Thus, this XML file needs to be edited according to the requirements for further processing. This is one of the disadvantages of this method. Once the necessary changes in the XML file is made, this XML file can be used as input for the ATCUM (Automatic Test Case generation from UML Models) tool for generating test cases. Using the ATCUM tool, the option of opening the XML file is provided.

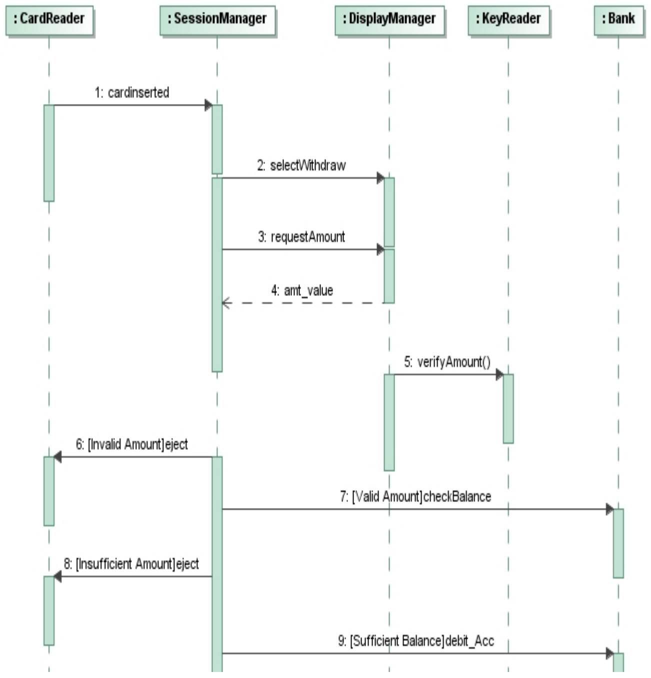

Fig 5.1: Sequence Diagram of Bank ATM System (Withdraw Money)

5.3 Step 3: Parsing XML File

The parser is written that reads XML file that is selected from the previous step and gives the description about all the tags as well as the attributes of every tags from the XML file. This information will be useful for generating the description for the Sequence Diagram Graph that contains the nodes which stores the necessary information to generate scenarios.

The project makes use of a tree-based API (such as Document Object Model, DOM) builds an in-memory tree representation of the XML document. It provides classes and methods for an application to navigate and process the tree [8].

5.4 Step 4: Generating Scenarios

The input to this step requires the parsed XML file of the previous step. The nodes for the graph are generated at this step. Each node stores the information such as message passed between the two objects, object that sends the message, the object that receives the message, guard condition if any and the OCL expression of that message [3]. Depending upon the message sent from one object to another object, the number of nodes is determined. The sequence diagram is then scanned again using the XML file to generate the scenario which is nothing but the combination of different nodes. Initially scenario begins with StateX. Then the message detail that is in the form of node is added to the scenario. If the receiver object is same as the sender object of the first node, then that scenario is completed and finally ends with StateY or StateZ. In this manner all the remaining scenarios from the sequence diagram is determined. These scenarios are used as test cases for the test case generator module. From the above sequence diagram, two different scenarios can be generated. The following figure Fig 5.2 represent two scenarios from Bank ATM System for Withdraw Money Use Case.

contents of scenario:1:
<stateX>
S1: null cardinserted :SessionManager :CardReader uml:Message
S2: null selectWithdraw :DisplayManager :SessionManager uml:Message
S3: null requestAmount :DisplayManager :SessionManager uml:Message
S4: null reply amt_vlaue :SessionManager :DisplayManager uml:Message
S5: null verifyAmount :KeyReader :SessionManager uml:Message
S6: Invalid Amount eject :CardReader :SessionManager uml:Message context CardReader::eject();pre: SessionManager.amount="Invalid Amt";post: result="eject card & Displays Welcome Message" OCL2.0 uml:OpaqueExpression
<StateY>

contents of scenario:2:
<stateX>
S1: null cardinserted :SessionManager :CardReader uml:Message
S2: null selectWithdraw :DisplayManager :SessionManager uml:Message
S3: null requestAmount :DisplayManager :SessionManager uml:Message
S4: null reply amt_vlaue :SessionManager :DisplayManager uml:Message
S5: null verifyAmount :KeyReader :SessionManager uml:Message
S7: Valid Amount checkBal :Bank :SessionManager uml:Message context Bank::checkBal();pre: SessionManager.amount="Valid Amt";post: result="Check the balance & " OCL2.0 uml:OpaqueExpression
S8: Insufficient Amount eject :CardReader :SessionManager uml:Message context CardReader::eject();pre: SessionManager.amount="Insufficient Amt";post: result="eject card & Displays Welcome Message" OCL2.0 uml:OpaqueExpression
<StateY>

Fig 5.2: Output of Scenario Generation

5.5 Step 5: Display Graph

The graphical representation of the nodes starting from StateX to StateY or StateZ is displayed in this frame. Initially all the nodes of the first scenario is displayed, then after clicking on the button one by one all the nodes of the remaining scenarios can be displayed. Thus, the visual representation of the graph as well as scenarios can be obtained on clicking the required button of the ATCUM tool. The fig 5.3 represents the required graph for the above sequence diagram.

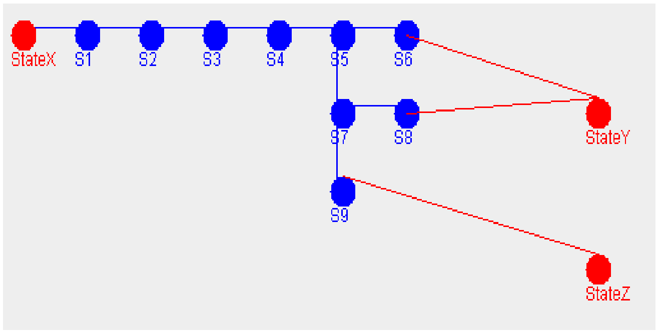

Fig 5.3: Displaying Sequence Diagram Graph

5.6 Step 6: Test Set Generation

This step reads the different scenarios that are generated from the previous step. Each scenario corresponds to test case. All the paths from the start node to final node need to be scanned to generate the test cases. The information that is stored in the nodes is used to determine the input and expected output of the scenario. The OCL expression plays a major role in determining the test cases. Form the OCL expression, the precondition and the post condition for a message can be determined. A message having the guard condition and OCL expression is selected for generating the test case. The following figure 5.4 represents the output of test case set generation.

5.7 Step 6: Creating & Saving Temporary File

The last step is to save the test cases generated from the previous step in a temporary file for future reference. Also this file can be used software testers to create the test plans for the desired software. The temporary file can be saved with .txt extension so that it can be easily available for other users. The following figure fig 5.4 represents the text file.

6. CONCLUSION

The aim was to automatically generate test cases from UML Models. To generate test cases various UML Diagrams are considered such as Use Case Diagram, Class Diagram, Use Case template, sequence diagram & also data dictionary using OCL. The paper presents the methodology to convert the UML sequence diagram into a graph called sequence diagram graph (SDG). The information those are required for the specification of input, output, pre- and post- conditions etc. of a test case are retrieved from the extended use cases, data dictionary expressed in OCL 2.0, class diagrams (composed of application domain classes and their contracts) etc. and are stored in the SDG. The approach provides an ATCUM tool that straightway can be used to automate testing process. The approach does not require any modification in the UML models or manual intervention to set input/output etc. to compute test cases. A graph based methodology is followed and run-time complexity is governed by the breadth-first search algorithm to enumerate all paths. In fact deciding test data, which are embedded in design artifacts, is

Test Case Generation
PreCondition : ATM card is valid and User selects Withdraw option
Test Scenario:Invalid Amount
Output:context CardReader::eject();pre: SessionManager.amount="Invalid Amt";post: result="eject card & Displays Welcome Message"
Input: amount="Invalid Amt"
Output: "eject card & Displays Welcome Message"
Test Scenario:Insufficient Amount
Output:context CardReader::eject();pre: SessionManager.amount="Insufficient Amt";post: result="eject card & Displays Welcome Message"
Input: amount="Insufficient Amt"
Output: "eject card & Displays Welcome Message"
Test Scenario:Sufficient Balance
Output:context Bank::debit_acct()pre: SessionManager.amount=" Valid" and Bank.bal="sufficient";post: result="Amount Withdraw Successfully & Menu for Transaction "
Input: amount=" bal="sufficient";post:
Output: "Amount Withdraw Successfully & Menu for Transaction "
PostCondition : Amount Withdraw Successfully

Fig 5.4: Test Cases Generation

7. REFERENCES

[1]. Monalisa Sarma, Debasish Kundu, Rajib Mall, "Automatic Test Case Generation from UML Sequence Diagrams", Department of Computer Science & Engineering, IIT Kharagpur, IEEE 2007

[2]. Peter Frohlick and Johannes Link, "Automated Test Case Generation from Dynamic Models"

[3]. Object Constraint Language 2.0 is available from Object Mangement Group's web site http://www.omg.org/

[4]. J. D. McGregor, and D. A. Sykes, "A Practical Guide to

[5]. Testing Object-Oriented Software ", Addison Wesley, NJ, 2001.

[6]. A. Abdurazik, J. Offutt., "Using UML Collaboration Diagrams for static Checking and Test Generation", Proceedings of the Third International Conference on the UML

[7]. A Rational Approach to Software Development using Rational Rose 4.0, IBM's Rational Rose, http://www.ibm.com/

[8]. Monalisa Sarma, & Rajib Mall, "System Testing using UML models", Department of Computer Science & Engineering, IIT Kharagpur, 16th IEEE Asian Test Symposium

[9]. W3 school available at: http://www.w3schools.com/dom/dom_node.asp

[10].C. Nebut, F. Fleurey, Y. L. Traon, and J. Jean-Marc, "Automatic Test Generation: A Use Case Driven Approach", *IEEE Transaction on Software Engineering*, Vol. 32, No. 3, 2006