Testing Graphical User Interface Using Decision Based Sequence Diagrams

Mahamaya Mohanty
Assistant Professor, School of Computer Engineering KIIT University, Bhubaneswar
Email: mahamayamohanty@yahoo.co.in

ABSTRACT
This paper presents a novel approach to generate and execute graphical user interfaces from decision based UML sequence diagrams. This has been designed especially for efficient user interfaces where with less generation and verification effort are required different types of decision tables and UML sequence diagrams. The decision based sequence diagrams test sets did better at revealing integration level faults than only sequence diagrams. The experiment showed that model-based testing can be used to systematically generate test data and indicates that decision based UML models can play better roles in testing. Decision based Sequence Diagrams helps in deriving a reasonably complete reference model usable in industrial contexts, which can then be used for automatically deriving the test cases from user interfaces without going through the lines of codes of a program. Here the paper concentrates on generation of test cases for integration-level testing based on UML interaction diagrams where sequence diagram is taken into consideration. In this approach, different User Interfaces are designed from different decision based interaction diagrams.

KEYWORDS
Decision table, UML diagrams, sequence or interaction diagram, test case, user interface, integration level testing.

1. INTRODUCTION
Using object-oriented languages decreases the amount of testing needed. To plan and execute tests, software testers must consider the software and the function it computes, the inputs and how they can be combined, and the environment in which the software will eventually operate. This difficult, time-consuming process models can be refined to get actual implementation of the system. Besides testing the codes it is efficient to test the user Interface because if code and input coverage were sufficient, released products would have very few bugs. Sequence diagrams can be used for integration level testing [1,6,9]. Integration testing concentrates entirely on module interactions, assuming that the details within each module are accurate. Object-orientation has rapidly become accepted as the preferred paradigm for large scale system design. The reasons for this are well known and understood. First, classes provide an excellent structuring mechanism. They allow a system to be divided into well defined units which may then be implemented separately. Second, classes support information-hiding. A class can export a purely procedural interface and the internal structure of data may be hidden. This allows the structure to be changed without affecting users of the class, thus simplifying maintenance. Third, object-orientation encourages and supports software reuse. This may be achieved either through the simple reuse of a class in a library, or via inheritance, whereby a new class may be created as an extension of an existing one. In both cases the result is a reduction in the amount of software which must be written and, as a result, an improvement in the reliability of the resultant system since previously tested classes may be utilised. The key advantage of the object-oriented paradigm is that it provides a uniform structure for all components in the form of a procedural interface. Although as indicated above, this appears to complicate the testing process, it may be exploited to support an object-oriented approach to testing. At the other extreme, module and integration testing can be combined, verifying the details of each module's implementation in an integration context. It can optimize the total number of graphical user interfaces as modules as per the requirements for a project after deducting the GUIs from different sequence diagrams. This can be considered as a descent approach for solving the complexity of GUIs. Further this work is aimed to find the GUIs from extended sequence diagrams.

The rest of the paper is organized as follows. Section 2 is the literature survey and review of related work. Section 3 is about the basic concepts and definitions. In Section 4 we have presented the proposed algorithm for obtaining GUI from sequence diagram. Section 5 gives the analysis of the proposed algorithm along with description and Section 6 concludes the paper.

2. RELATED WORKS
A number of different methods have been proposed for generating test cases [4]. A test case is a description of a test, independent of the way a given system is designed. Test cases can be mapped directly to and derived from use cases. They can also derived from system requirement. One of the advantages of producing test cases from specifications and design is that they can be created earlier in the development life cycle and be ready for use before the programs are constructed. Additionally, when the test cases are generated early, Software Engineers can often find inconsistencies and ambiguities in the requirements specification and design documents. This brings down the cost of building the software system as errors are eliminated early during the life cycle. Random techniques determine test cases based on assumptions concerning fault distribution. Path-oriented techniques generally use control flow information to identify a set of paths to be covered and
generate the appropriate test cases for these paths. These techniques can further be classified as static and dynamic. Static techniques are often based on symbolic execution, whereas dynamic techniques obtain the necessary data by executing the program under test. Goal-oriented techniques identify test cases covering a selected goal such as a statement or branch. Since UML (Unified Modeling Language) is the most widely used language, many researchers are using UML diagrams such as state-chart diagrams, sequence diagrams, use-case diagram etc to generate test cases and has led to Model based test case generation. The possibility of using UML for software testing was addressed by Clay E. Williams [12]. UML models are built extensively for Object Oriented software systems. Class diagram, state diagram and OMT (Object Modeling Technique) and unified process are used to test Object Oriented systems. Here a framework to show the generation of Graphical User Interface from UML sequence diagram is provided. Even though variety of approaches have been proposed, with the advent of modeling tools like Rational Rose, for a decade there has been constant research on generating test cases based on specifications and design models. For easy understanding we have classified test case generation approaches mainly into two categories-Specification based test case generation [14] and Model based test case generation. In this paper the work has been carried out on Model based test case generation using sequence diagram and deriving respective GUIs. Specification Based testing described by Marlon E Viera et al [7] also employs UML state chart diagrams as the basis and automatically generates test drivers and test script to validate the component under test. Constant research and challenge in Model Based approach led Jeff Offutt et al [2,3] to extend their work from system level analysis using state-chart diagrams to integration level analysis using collaboration diagram. The authors defined criteria for both static and dynamic testing of specification level and instance level collaboration diagrams. An algorithm has been designed to ensure that tests satisfy the formal testing criteria and help the tester to trace the design of appropriate GUIs. At unit level, state charts were better compared to sequence but in integration level it was vice-versa. Many new tools have been built to work with existing tools to generate test cases. Architecture exists for automated test generation using UML profile and Test directives that consists of Test purposes. A framework for model level testing of behavioral UML models was proposed by Andras Toth et al [8]. Planner algorithm has been used for automatic generation of test cases. A planner meta-model was constructed to provide a high level representation of mathematical definition, in-order to keep their methodology open for other planner tools. As a result of this project UML designs can be tested and design flaws can be detected in modeling phase of development process prior to any implementation activities saving a considerable amount of cost and effort. Matthias Riebish et al [5] presented a procedure for iterative software development process in generating test cases. Scenarios and Use cases fed as input for requirement engineering provided the basis for test case generation. An approach that focuses on developing effective techniques and tools for test case generation and coupling them with suitable execution tools for unit, integration and system testing was proposed by Jean Hartmann et al [13]. The authors have integrated their tool with UML to automatically generate black box conformance test early in the development life cycle. For unit and integration testing the authors have derived tests from State-chart diagrams and Sequence diagrams. For system level they have used use-case and Activity diagrams. User Interface testing tool Rational Rose is used. An approach that allows software testers as well as developers to generate effective test cases based on UML use case has been presented by Ahmed M. Salem et al [11]. The authors have demonstrated their work by performing case studies on Procurement/payment system. Use cases have been documented with pre-condition, post condition, basic and alternate flows. With the help of this test cases have generated in the form of matrix. This approach offers good traceability to original requirements, to test and verify requirements and to discover inconsistency in requirements. Further improvement has been suggested to develop a model for non-functional requirements.

3. BASIC CONCEPTS

Decision table:
A decision table defines logical procedure by means of a set of conditions and related actions. A decision table is divided into four parts by horizontal and vertical lines. There is a condition stub having a list of conditions in the upper half of the table. The bottom half of the decision table lists the actions to be performed. The bottom quadrant has crosses (X) and dashes (--) entries. As shown in fig. 1b a cross indicates that the action listed in that row has to be performed and a dash indicates that the action is not performed which are called as action entries. Vertical columns to the right of the condition stub is known as rule which is interpreted by using condition listed in the condition stub and the answers listed in condition entries and its corresponding actions are obtained from the action stub and entries.

<table>
<thead>
<tr>
<th>CONDITION STUB</th>
<th>ACTION STUB</th>
<th>RULE1</th>
<th>RULE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDITION 1</td>
<td>ACTION 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONDITION 2</td>
<td>ACTION 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONDITION k</td>
<td>ACTION m</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONDITION ROW</th>
<th>ACTION ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDITION 1</td>
<td>ACTION 1</td>
</tr>
<tr>
<td>CONDITION 2</td>
<td>ACTION 2</td>
</tr>
<tr>
<td>CONDITION k</td>
<td>ACTION m</td>
</tr>
</tbody>
</table>
Testing Graphical User Interface Using Decision Based Sequence Diagrams

**Test Case:** It is a composition of inputs and pre-test states of the software.

**Expected Results:** It is a composition of outputs and post-test states conditions of a software.

**UML diagrams:** These provide different perspective of the software system to be developed and facilitate to get comprehensive understanding of the system. These models can be refined to get actual implementation of the system. Here we have considered the sequence diagram as the testing resource. A diagram shows interactions among objects. The objects participating in the interaction are very clear from the diagram. The messages on the arrows show the sequence of operation.

**Sequence diagrams:** In the UML, a message is a request for a service from one UML actor to another, these is typically implemented as method calls. Each sequence diagram represents a complete trace of messages during the execution of a user-level operation. We form message sequence paths by using the messages and their sequence numbers. Message sequence paths can be traces of Graphical user interface interactions or component (object) level interactions. We defined the following coverage criteria for generating test cases from sequence diagrams.

**User interface testing:** It is performed to know how user-friendly the application is. It also determines whether
- Appropriate input help is displayed on the screen.
- Correct messages are displayed when an error is encountered.
- Navigation within the application is easy.

**Integration-level Testing:** This is a type of testing where different modules of a system are integrated and planned to ensure that there are no errors in parameter passing when one module invokes another module.

4. Proposed Algorithm

For this we had proposed an simple stepwise algorithm

Step 1: Start

Step 2: Consider all the possible instances for a given problem.

Step 3: Find out all the possible conditions for each instance and enter a yes or no in the condition entry row.

Step 4: For each conditions satisfied with its respective action sequence diagram can be drawn.

Step 5: Sequence diagrams satisfying different conditional rules and performing a certain action can be derived from the decision table.

Step 6: Now graphical user interface is drawn according to each sequence and conditional sequence diagrams.

Step 7: The number of GUIs are reduced according to the requirements which reduces the overall complexity for GUI testing.

Step 8: Finally each GUI is tested by integrating them and performing integration-level method of automation testing.

Step 9: Stop

5. Working of the proposed algorithm

Let us consider a cash withdrawal operation in an application of Internet Banking Customer Service shown in fig.2 which has three conditions and three actions.

- When the correct Account Number and PIN are entered then we can check the next web page, where the details about the customer are displayed. Respective GUI is shown in fig.7 and fig.8 derived from sequence diagram shown in fig.3.
- If a wrong account number or PIN is entered then a message box indicating them to be invalid should appear as in fig.9 and fig 10 derived from sequence diagram of fig.4.
- If the cash requested is more than the available balance in account then the user should get a respective message i.e. is shown in sequence diagram of fig 5.

Suppose someone tries to access with the input only to a single field then an alert message should appear indicating that another text field should be entered with some values as shown in fig 11 and its respective sequence diagram is given in fig 6. The sequence diagrams are as follows and the respective graphical user interfaces that can be deduced from the sequence diagram are shown below. The different sequence diagram can help to draw out all the required GUI components for framing a GUI for particular software.
6. DESCRIPTION

Fig.2 Decision Table showing different conditions that may arise during a cash withdrawal.

The above decision table is for cash withdrawal which portrays different conditions along with its respective actions. When the account number is correct, PIN entered is correct and the cash amount requested is less than the available balance then the transaction is possible which takes place in action stub. Secondly when account number is correct, PIN entered is correct but the cash amount requested is more than the available balance, then the transaction is not possible. When either of account number or PIN entered is incorrect but the cash amount requested is less than the available balance, then also the transaction is not possible shown in the action stub.

Another action that displays an alert message also occurs during any invalid entry during any transaction. From different sequence diagrams it is easier to obtain user interfaces. Finally, performing testing on user interfaces reduces number of errors, redundancies as well as time saving.

Testing of software involves executing the software and observing how it reacts. If the software behaves as expected, confidence in its reliability is increased. Software testing has progressed from being a post-development activity to an activity that is integrated throughout the development phases. In the traditional waterfall software development life cycle, software testing was done after a software system had already been built. More recently defined software development methodologies and/or processes recommend that testing be done at various stages throughout the development cycle. Software testing is based on assumed inputs that may give false test results if the assumptions are wrong. Testing may reveal failures but does not necessarily help with pointing out where the faults in the software are located. Moreover, testing requires that there is an oracle present to specify what the expected results from the test are. In worse cases, testing may not even reveal failures because the faults either do not get executed, or get executed but deceptively give correct results for the inputs that are chosen. The size of software systems also affects testing. As systems get larger testing becomes more expensive. Estimates of the cost of software testing range from 40% to 80% of the entire development costs. Hence testing software with respect to its graphical user interface also consumes less cost. When the user interface is a decision based derived GUI it requires less testing effort as well as less time which are the beneficial factors for a tester.

![Fig.3 Sequence diagram for cash withdrawal with valid Account Number and PIN](image-url)
Testing Graphical User Interface Using Decision Based Sequence Diagrams

Fig. 4 Sequence diagram for an invalid PIN entry

Fig. 5 Sequence diagram for unsuccessful withdrawal due to requested cash is greater than available cash

Fig. 6 Sequence diagram for an alert message due to less digit Account Number entry then the minimum requirement

Fig. 7 GUI showing the valid input values

Welcome

Dear Mr/Mrs Laxmi Nayak,

Account No: 11111
Your Cash Withdrawal as on 14-Mar-08 10:22:19 A.M is 12000.0

Thank you for giving an opportunity to serve you.

Fig. 8 Output for valid input values
7. CONCLUSION
We have derived different user interfaces based on UML specifications both in component and object based contexts. This method guarantees that all available sequences are covered, but it cannot provide any coverage measure over the implementation system. This approach meets some important requirements imposed by the Industry. Advantage of this approach is that incompleteness in the model need not be modified leading to low testing effort and generation of accurate test cases. By knowing the interactions and messages between the objects they are integrated and testing is performed with a goal to find defects that arise when these fault free components interact with each other in an incorrect way. From these interactions the number of efficient and required GUIs for any scenario can be derived thus optimizing the number of GUIs. The integrated scenarios minimizes the fault by displaying the resultant GUI for an input screen GUI when there is a correct message passing interaction among the objects. The future scope lies with the reuse of the GUIs and to generate even stronger and reliable test cases we have planned to integrate UML specification with OCL (Object Constraint Language)[10]. Also it is intended to improve performance of test case generation, by proposing precise measurement technique for data coverage. Complexity involved in finding a large number of GUI’s in actual scenario can be optimized by using some of the soft computing technologies which is its future scope.

8. FUTURE SCOPE
The future work is aimed to derive different sequence diagrams from different types of decision tables such as Extended Entry Decision Table, Mixed Entry Decision Table and Limited Entry Decision Table. Those sequence diagrams can be used for designing different Graphical User Interfaces having different conditions and its respective action. Those GUIs makes the work of testing easier. Finally it is aimed to make the entire system automated to make the system faster and error-free.

9. REFERENCES
[5]. Matthias Riebish, Ilka Philippow, Marco Gotze, "UML Based Statistical Test Case Generation”.
[7]. Marlon E. Vieira, Marcio S. Dias, Debra J. Richardson, "Object-Oriented specification-Based Testing Using UML Statechart Diagrams", University of California.


[13]. Jean Hartmann, Marlon Vieira, Herb Foster, Axel Ruder, "UML based Test generation and Execution".
