Journal of
Information Technology & Software Engineering

Muthusamy and Badurudeen, J Inform Tech Softw Eng 2014, 4:1
DOI: 10.4172/2165-7866.1000128

Research Article Open Access

A New Approach to Derive Test Cases from Sequence Diagram

Muthusamy MD* **and Badurudeen GB**

Department of Computer Science and Engineering, Sona College of Technology, Salem-636005, Tamil Nadu, India

Abstract

Testing is an important area of software engineering. There are various types of testing methodologies followed in various stages of Software Development Life Cycle (SDLC). We are proposed a novel approach for generating test cases from UML sequence diagram. Our approach consists of transforming sequence diagram in to sequence diagram graph and generating test cases from SDG. The sequence diagram is prepared based on the Use Case diagram in which describes the overall view of the system. The traceability between the models is provided by using Relational Definition Language.

Keywords: Model based testing; Sequence diagram; SDG; Test case Generation; OCL

Introduction

Testing performs an important role in software engineering, which only ensures the quality of software being produced. Most of the testing methodologies involves in the Black box approach, while compared to design level testing. Before testing, the development life cycle reduces the difficulties on implementation level.

In model based testing, the test cases are generated from the abstract representation of the model. The Abstract test suites are not executed against system under test because it is needed to be described from the corresponding abstract test suite. The effect of model based testing is it offers automation of testing. If a model is readable one then it should have a well defined behavioral interpretation. During testing the model and its behavioral specification is given to model checker. After the verification the paths were used for generating test cases. Here a little knowledge of the coding is needed for the tester instead of brief knowledge about coding in testing. They must to understand the UML models which is used for System under Test.

Model based testing provides the conformance with the UML diagrams. In the case of design level there are various diagrammatic representations used to represent the operations with operational requirements. Here we are using the Use case diagrams to get the requirements involved in the system and tracing the important scenarios to sequence diagram and converting in to sequence diagram graph. By using the sequence diagram graph test cases are generated.

Related Work

There are many of researchers proposed many methodologies for scenario level test case generation. Most probably they used black box approaches and they do not considered architectural behavioral designs. In the software development life cycle from requirement specification to actual product, the verification and validation takes place. Probably the products where verified and validated based upon their requirement specifications. More recently model based testing become as popular. Marketing proposes various methodologies in model based testing. Model paradigm contains state based notations, transition based notations, history based notations, functional notations, operational notations and criteria for test selection, in which includes data coverage, requirement based coverage, fault based criteria and about the different types of tools used in model based testing [1].

Combination of State machine diagrams and Class diagrams [2] used to generate automatic test cases using OCL expressions. This algorithm is used with ParTeG. Static analysis tools such as OCLE [Chiorean] and USE [Richter's] [3] can be used to analyze structural properties of Class Models. This approach removes manually simulating the behavior of Class Models. Instead of that it provides a light weight approach to check scenarios.

AGEDIS [4] includes an integrated environment for modeling test generation, test execution and other related activities for industries. This is widely accepted tool by industries, but it has some drawbacks. Briand and Labiche [5] describe the TOTEM (Testing Object Oriented Systems with the Unified Modeling language) system testing methodology. System test requirements are derived from early UML analysis artefacts such as, use case diagrams and sequence diagrams associated with each use case and class diagram.

For testing different aspects of object interaction, several researchers have proposed different technique based on UML interaction diagrams [6-10] , Bertolino and Basanieri proposed a method to generate test faces using the UML use case and Interaction diagrams (specifically, the Message Sequence diagram). It basically aims at integration testing to verify that the pre-tested system components interact correctly. They use category partition method and generate test cases manually following the sequences of messages between components over the Sequence Diagram.

From the use case diagram, the Use Case Dependency Graph (UDG) and Concurrent Control Flow Graph (CCFG) from corresponding sequence diagrams for test sequence generation [11]. UML state charts provide a solid basis for test generation in a form that can be easily manipulated [12]. The traceability between UML Models provides reengineering [13] it increases the test case coverage in test case generation. Bertolino and Maechetti [14] proposed and approach to generate test cases even the software's are partially model. The OCL [15] is a Generalized Model free Language and it is possible to combine programming languages and OCL in UML model processing. It is model language primarily meant for expressing constraints in model. Panthi V and Mohapatra DP [16] propose an approach to generate test cases from sequence diagram by generating extended finite State Machine for the diagram. They proposed an algorithm named as ATGSD algorithm, which focuses on the object coverage Message

***Corresponding author:** Muthusamy MD, Department of Computer Science and Engineering, Sona College of Technology, Salem-636005, Tamil Nadu, India, Tel: 91-427-409999; E-mail: dhineshmsc2011@gmail.com

Received February 19, 2014; **Accepted** May 01, 2014; **Published** June 09, 2014

Citation: Muthusamy MD, Badurudeen GB (2014) A New Approach to Derive Test Cases from Sequence Diagram. J Inform Tech Softw Eng 4: 128. doi:10.4172/2165-7866.1000128

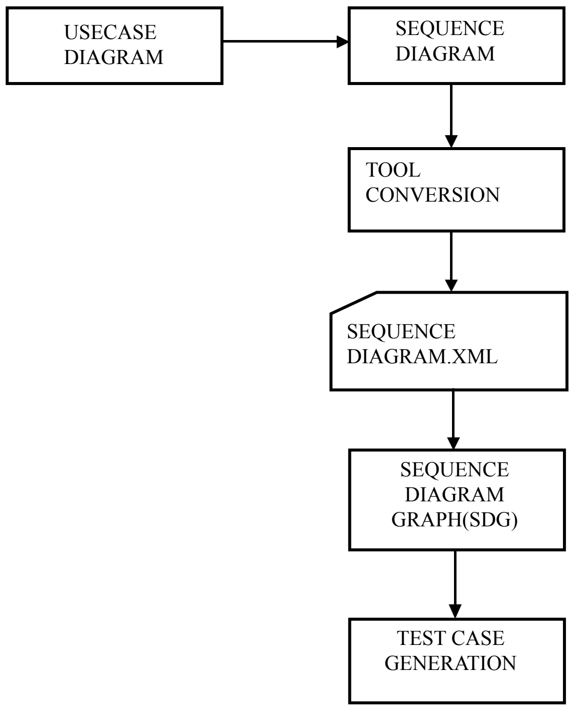

Figure 1: System Architecture.

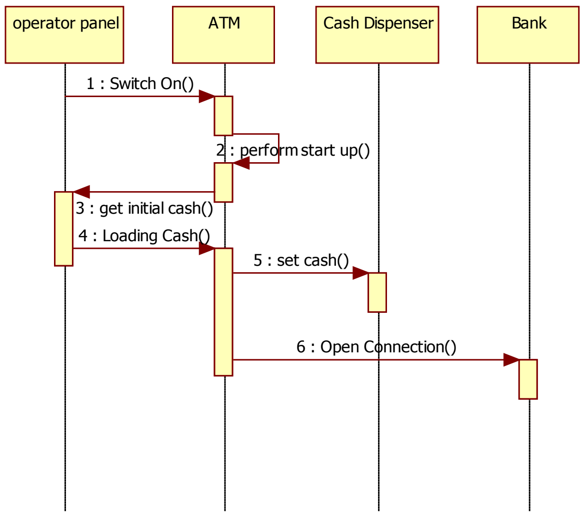

Figure 2: Sequence diagram for ATM Banking System.

sequence Path Criterion, Full Predicate Coverage and Boundary testing criteria. Traceability [17] between the models, which ever constructing from the UML can be done easily.

Proposed Approach

The proposed approach describes the following steps to generate test cases from the sequence diagram. Given a sequence diagram (SD), in which describes the detailed interaction of use cases and the actors involved in the system. We transform it in to Sequence Diagram Graph (SDG), the sequences are traced from the use case diagram. The

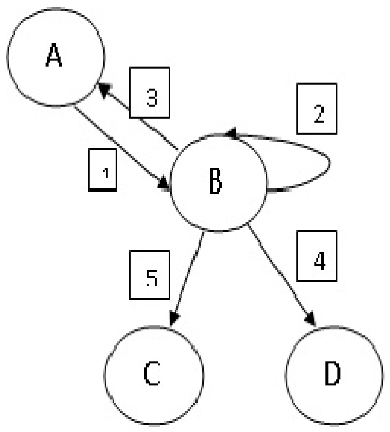

Figure 3: Sequence Dependency Graph for the sequence diagram.

sequence diagram is built with Object Constraint Language (OCL), which is a generalized language proposed for UML Models by object management group.

The OCL gives the input condition and output condition in the name of pre condition and post condition, which represents before execution of the state and after execution of the state of the behavior involved in the sequence diagram. The sequence diagram is converted into XML format. The XML file is converted into a Graphical notation called as tree structure. The structure is named as sequence dependency graph (SDG).

We then traverse the sequence diagram graph and generate test cases based on path coverage, functional coverage criteria. An algorithm is proposed for generating test cases. The algorithm used for traversing the graph is described in detail in the following section. The following architecture such that Figure 1 describes the schematic representation of proposed approach.

Architectural description

Use case diagram: The use case diagram describes the overall system, which contains the actors and use cases in the system in a sequence. The external persons or actors who will interact with each other through use cases.

Sequence diagram: The sequence diagram describes the detailed description and interaction with in the use case and actor. The interactions are written in OCL (Object Constraint Language), which is a generalized model free language. So this allows the user easily to understand about the system interactions. The sequence diagram is shown in the Figure 2 for ATM system.

Tool conversion: The sequence diagram can be exported into an XML/XMI format using an XML parser. This file contains all the XML tags that describe the sequence diagram. The diagram parameters and its values defined in XML format.

Sequence diagram graph: The sequence diagram graph defines the activities as nodes and the interactions in the form of paths. The SDG will look as an acyclic graph notation. By using this SDG the test cases are generated. The sequence diagram graph has the graphical structure like Figure 3. It shows the graphical notation of the sequence diagram of ATM system.

Test case generation: The test case is generated by visiting the nodes and edges in the SDG. We were proposed the algorithm called as Iterative Deepening Depth first search algorithm (DFS). Using this

Table 1: Test Case Generation from SDG.

Test case id	Starting node	Dependent node	Input to the node	Expected result	Actual result
1	A	B	Switch on	System loading	System loading
2	B	B	Perform start up	Performing system start up	Performing System Start up
3	B	A	Get initial cash	Loading Cash	Loading initial cash
4	A	B	Load initial cash	Getting Cash	Basic Balance Loaded
5	B	C	Set basic cash	Setting basic balance	Balance
					Loaded
6	B	D	Open Connection	System Loading	Connection Launched

algorithm the test cases are generated, which reduces the time taken to visiting the interactions which takes long time. It increases the path coverage criteria. The Table 1 shows some of the sample test cases generated from the SDG depending on the condition.

An example

We have taken the example of ATM Banking System for generating test cases. The user before entering into the system, the cash dispenser will be loaded by the authorities of respective bank with the existing amount of dispenser.

For that the how could be the test cases are generated was shown here above in the test case table. Initially the sequence diagram for the ATM Banking system will be drawn in the Eclipse software development kit. The necessary UML plug-in were installed into the Eclipse before generating the diagram. The Eclipse supports the generation of UML diagrams and it also supports the OCL expressions. The complete scenario can be generated for ATM banking system with the use of Eclipse. This diagram contains the sequence of interactions between the operator panel, ATM machine, cash dispenser and bank authorities.

The sequence of interactions shown in the ATM banking system sequence diagram can be converted into an XML format. The conversion of XML file can be done automatically by using the XML file conversion option already available in the Eclipse. The XML file represents the sequence of interactions with its functional notations. It bridges the gap between different of interactions among the sequences.

The XML file of sequence diagram will be converted to sequence dependency graph as well as sequence diagram graph. The sequences of activities in the ATM sequence diagram are represented as nodes and the interactions among those nodes are represented as paths between the nodes. The test cases are generated from the sequence dependency graph by visiting the nodes and paths.

The final output of the system will be number of nodes visited and paths visited by the given condition satisfaction criteria. During visiting the nodes some of the nodes and paths are repeated to satisfy the condition criterion. The test case generation table (Table 1) shown here is for understanding purpose. The output will not be look like as we shown in the Table 1. So that the condition looked on the path during test case generation is shown in table.

Algorithm used for test case generation

```
Procedure idvisit (G, S, Goal)
Inputs
G: Graph with nodes N and paths A
S: Set of start nodes
Goal: Boolean functions on states
Output
Paths from S to a node for which is true
Test case condition
Local
Natural failure Boolean
Bound: integer
Procedure dbsearch ($n_0$ ...$n_k$, b)
Inputs
$n_0$,.....,$n_k$,..Paths:
b: integer, b>=0
Output
Path to goal of length k+6
If (b>0) then
For each path $n_k$, n ∈ A do
dbsearch($n_0$,...,$n_k$, n b-1)
Else if (goal($n_k$)) then
return $n_o$,......$n_k$
Else if (nk has any neighbours)
natural failure←false then
bound ← 0
Repeat
natural_failure ← true
dbsearch ({S:s ∈ S} bound)
bound ← bound +1
until natural_ failure
Return
```

The test case generation is starts from the initial node s which is a subset of S. the nodes in the graph are starts from n0 ends with nk, which is called as last node. During visiting of paths the nodes has more paths to visit the bound is increased as bound←bound+1. When the condition is not satisfied, then the node will be terminated from traversing. This algorithm is called as iterative deepening depth first search algorithm, in which combines the features of basic depth first search and breadth first search algorithms to make effective test case generation. This algorithm takes less time to visit the nodes when compared to other algorithms. A largest tree comprised of several nodes and paths, it can be handled by iterative deepening depth first search algorithm effectively. The existing basic depth first search algorithm requires medium sized graphs to traverse.

This algorithm will not handle large sized graphs and it takes more time to traverse on the space.

Calculations in generating test case from sequence dependency graph (SDG)

After generating test cases from the SDG, the traversing of nodes and paths are calculated by the following formulae.

The worst case performance for explicit graphs traversed without repetition is calculated as

$$o(|E|)$$

Where E is the number of edges

The order of visiting nodes without repetition is calculated as

Where b - branching factor and d - depth

$$o(|b^d|)$$

Traversing the entire graph without repetition (visiting the longest length paths) calculated as

$$o(|v|)$$

Where v-number of vertices

This formula includes without eliminating the visiting of duplicate nodes.

Test case generation from SDG

The sequence diagram depicts the behavior of various sequences of the System under Test (SUT). Among all other UML diagrams the sequence diagram only depicts the behavior among the sequences in a timely manner. The immediate request and responses of system are modeled by sequences for test case generation. From the sequence diagram graph when an object invokes another object the system predicts whether the right sequence of messaging is followed to accomplish an operation. The SDG eventually covers all the paths from the starting node to final nodes well as message sequence paths. We are going to use the iterative deepening depth first search algorithm in the sequence diagram graph to generate test cases. The complete path coverage, Branch coverage can be predicted through this algorithm. The interaction which takes long time between the paths is found by this algorithm.

Conclusion

We are proposed a novel approach to generate test cases from sequence diagram by generating sequence diagram graph. The input of pre condition and post condition retrieved from Use case diagram and OCL expressions. These things were stored in SDG. The iterative deepening depth first search algorithm can handle the sequence of interactions among the sequences, in which takes long time.

References

1. Mark U, Alexander P, Legeard B (2010) Taxonomy of Model Based Testing Approaches, Softw. Test. Verif. Reliab, John Wiley & sons Ltd.
2. Stephan W, Schlingloff BH (2008) Deriving Input Partitions from UML Models for Automatic Test Generation, Lecture Notes in Computer Science 5002: 151-163.
3. Yu L, France RB, Ray I (2008) Scenario-based static analysis of UML class models. Lecture Notes in Computer Science 5301: 234-248,Springer-VerlagBerlin Heidelberg.
4. Hartman A, Nagin K (2004) The AGEDIS Tools for Model Based Testing. IBM Haifa Research Laboratory, ACM.
5. Briand LC, Labiche Y (2001) A UML – Based Approach to System Testing, Proceedings of the 4th International Concepts and Tools, Springer-Verlag, London.
6. Abdurazik A, Offutt J (2000) Using UML Collaboration diagrams for static checking and test generation, Proceedings of the Third International Conferences on the UML, Lecture Notes in Computer Science, Springer-Verlag GmbH, York, UK 939: 383-395.
7. Lettrari M, Klose J, Ruder A (2001) Scenario-Based Monitoring and Testing of Real Time UML Models, Proceedings of UML, Springer-Verlag.
8. Basanieri F, Bertolino A, Marchetti E (2002) The Cow Suite approach to planning and deriving test suites in UML projects, Proceedings of the 5th international Conference on the UML, Lecture Notes in Computer Science 2460: 383-397.
9. Tonella P, Potrich, (2003) A Reverse Engineering of the Interaction Diagrams from C++ code, Proceedings of IEEE International conference on Software Maintenance.
10. Fraikin F, Leonhardt T (2002) Siditec-testing based on sequence diagrams, Proceedings of 17th IEEE International conference on Software Engineering.
11. Swain SK, Mohapatra DP, Mall R (2010) Test Case Generation Based on Use Case and Sequence Diagram, International journal of Software Engineering.
12. Nebut C, Fleurey F, Traon YL (2006) Automatic Test Generation: A use Case Driven Approach, IEEE Transactions on Software Engineering.
13. Offutt J, Abdurazik A (1999) Generating Tests from UML Specifications, Lecturer Notes in Computer Science, Springer-Verlag Berlin Heidelberg.
14. Bertolino A, Marchetti E, Muccini H (2005) Introducing a Reasonably Complete and Coherent Approach for model-based Testing, Electronic notes in Theoretical computer science, Elsevier.
15. Sikarla M, Peltonen J, Selonen P (2004) Combining OCL and Programming Languages for Model Processing, Electronic Notes in Theoretical Computer Science, Elsevier 102: 175-194.
16. Panthi V, Mohapatra DP (2012) Automatic Test Case Generation Using Sequence Diagram, International Journal of Applied Information Systems 174: 277-284.
17. George M, Hellmann KPF, Knahl M, Bleimann U, Atkinson S (2012) Traceability in Model Based Testing, Future Internet, 4: 1026-1036.