Test Case Generation from UML Models

Palak Sharma, ps2671@rit.edu
SE Department, Rochester Institute of Technology,
Rochester, NY, USA-14623

INTRODUCTION

Testing is a crucial phrase of any software development project. Writing efficient test cases is difficult as developers can miss few use cases. To keep track that all the desired functionalities are tested, it is a good idea to automate the process of test case generation. One of the best practices to capture system functionalities are using UML modeling. So if the test cases are generated using different type of UML models such as state diagrams, activity diagrams, and sequence diagrams, the test cases will be more robust and will capture the test cases for all the information modelled in the UML diagrams. In order to understand this topic in more detail, the focus of this research survey is automation of test case generation using different UML Models.

For conducting this survey, four papers are chosen which propose test case generation using different UML diagrams. [1] proposes a tool called "ModelJunit" which is use for test case generation based on UML state diagrams using depth first search algorithm. [4] makes use of "ModelJunit" developed in [2] to generate test cases based on UML sequence diagrams and achieve efficient test coverage. [3] uses genetic algorithm to generate test cases from UML Activity diagrams which can be especially useful during integration and system testing. In [4], the authors use a combination of Sequence diagrams and Interaction Overview diagram to generate the desired test cases. The aim is to find the fault or error location at an early stage so as to optimize resources and planning. A main idea in all these approaches is to increase the test case coverage and reduce the number of redundant test cases which can be achieved by optimizing and prioritizing the generated test cases.

METHODOLOGY

- ### Proposed Framework

A common theme amongst all the papers is the methodology used to generate the test cases from the models. The generalized framework derived from the methodology used in all the four papers is shown in Figure 1. Steps to generate the test cases from the models are as follows:

1) Create the desired UML model from the system source code. Ex – State diagram [1], Sequence diagram [2,4], Activity diagram [3] and Interaction diagram [4].
2) Using the UML diagram, convert it into its corresponding graph.

3) Traverse the generated graph to select a predicate function. Algorithm such as Depth first search can be used to extensively traverse through the graph.
4) Convert the predicate to source code using ModelJunit [1,2].
5) For optimizing the generated test cases, [3] makes use of Genetic algorithm whereas [4] makes use of dominance concept of graph.
6) On applying optimization algorithms on the generated test cases in Step 4, non-redundant test cases are obtained.

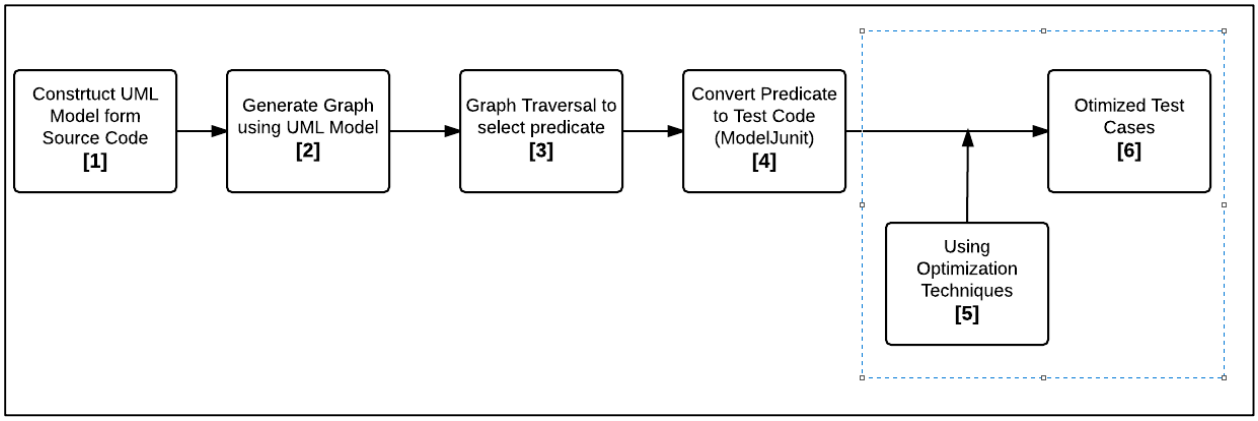

Figure 1 – Common Framework

- **Model Conversion Process**

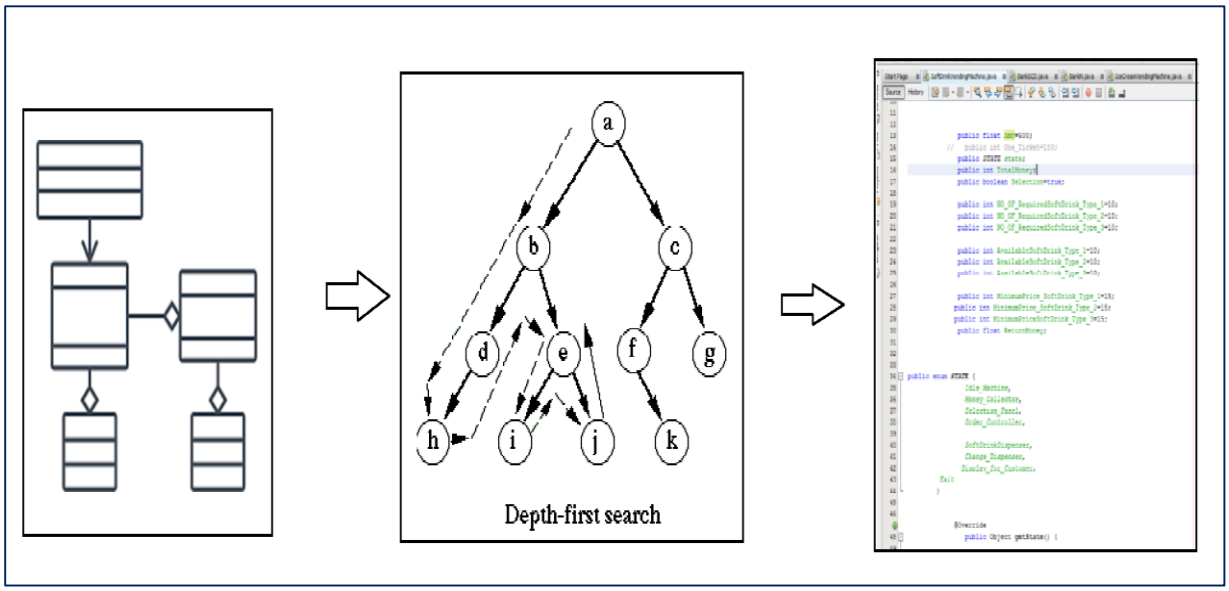

Figure 2 - Model Conversion Process

Figure 2 shows a sample for the methodology employed in the proposed framework. Depth-first search algorithm is used for graph traversing as it requires linear memory (i.e. only storing

the current node in traversed path) and so the results may be derived without examining the whole search space.

OVERVIEW

- ## Algorithms

    - **Depth First Search**
      The graph traversal algorithm used in [1, 2, 3, 4] is Depth First Algorithm. DFS is an uninformed search that progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it has not finished exploring. In a non-recursive implementation, all freshly expanded nodes are added to a stack for exploration. IT has advantage of memory over other search algorithms.

    - **Genetic Algorithm**
      As graph traversal may lead to redundant nodes and therefore generate redundant test cases, so in order to optimize the test case generation, [3] makes use of Genetic Algorithm. A genetic algorithm (GA) [5] is a method for solving both constrained and unconstrained optimization problems based on a natural selection process that mimics biological evolution. The algorithm repeatedly modifies a population of individual solutions based on principles of mutation and crossover. The generated offspring element is verified using a fitness function to select the best fit off springs.

    - **Dominance**
      For optimizing the test case generation process, [4] makes use of Dominance concept in graph traversal. A node n dominates a node m if every path P from the entry node n0 to m contains n.*" A (rooted) tree DT(G) = (V, E) is a digraph in which one distinguished node n0, called the root, is the head of no arcs; in which each node n excepts the root n0 is a head of just one arc and there exists a (unique) path from the root node n0 to each node n called dominance path" [4].*

- ## Advantages

1) The proposed techniques used to generate test cases from UML Models can be extremely useful in Test Driven Development as the test plans are created even before the implementation starts.

2) UML Models are created during the design phase of the Software Development Life Cycle. If the proposed frameworks are used to generate test cases after the design phase, the system test cases are more closely connected to the system design. Thus supporting the concept of validation and verification.

3) The generated test cases are automatically generated, thus are less prone to human errors and mistakes.

4) A major application of this techniques can be in mission or life critical systems in which the requirements are concise and precise and designing phase is completed well before the implementation phase.

LIMITATIONS & CHALLENGES

Some of the limitations associated with the current state of art related to field of automatic test case generation from UML Models [1, 2, 3, 4] are as follows:

- The case studies used in all the papers are limited in scope and do not emulate real time systems.
- The evaluation techniques used for analyzing the results of the proposed framework are analyzed statistically and not by end users such as project managers, developers or quality engineers.
- The papers don't answer the question of how useful are these techniques for automatic test generation for real time applications.
- A major threat for the proposed framework is the reliability of the UML Model. For effective and accurate test case generation, the requirements should be precise and so does the models. The models should represent the system concretely.
- Another unanswered question is the impact of change on the test cases." How does the change in models impact the test cases?"

CONCLUSION

The proposed approaches are novel and elementary in nature. Extensive investigation should be done to test the generated test cases by using real time industrial systems with an actual project team. In order for larger employment of test case generation techniques in real world, empirical studies should be carried out which calculates the cost in terms of resources, time and money.

REFERENCES

[1]Swain, Ranjita, et al. "Automatic test case generation from UML state chart diagram." *International Journal of Computer Applications* 42.7 (2012): 26-36.

[2] Panthi, Vikas, and Durga Prasad Mohapatra. "Automatic test case generation using sequence diagram." *Proceedings of International Conference on Advances in Computing*. Springer India, 2013.

[3] Jena, Ajay Kumar, Santosh Kumar Swain, and Durga Prasad Mohapatra. "A novel approach for test case generation from UML activity diagram." *Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*. IEEE, 2014.

[4] Jena, Ajay Kumar, Santosh Kumar Swain, and Durga Prasad Mohapatra. "Model Based Test Case Generation from UML Sequence and Interaction Overview Diagrams." *Computational Intelligence in Data Mining-Volume 2*. Springer India, 2015. 247-257.

[5] "Genetic Algorithm." Genetic Algorithm - MATLAB & Simulink. Retrieved from *https://www.mathworks.com/discovery/genetic-algorithm.html* N.p., n.d. Web. 09 Dec. 2016.

[6] "DEPTH FIRST SEARCH". DEPTH FIRST SEARCH. Retrieved from *http://intelligence.worldofcomputing.net/ai-search/depth-first-search.html#.WEsTLOYrI2w* N.p., n.d. Web. 09 Dec. 2016.