A Method for Automated Test Cases Generation from Sequence Diagrams and Object Constraint Language for Concurrent Programs

Thi Dao Vu, Pham Ngoc Hung and Viet Ha Nguyen
Faculty of Information Technology, VNU- Vietnam National University
E3 building 144 Xuan Thuy str., Cau Giay dist., Hanoi, Vietnam

Model- based testing plays a significant role in practice and a lot of researches on it has been investigated in recent years due to great benefits. Current research focuses on generating executable test cases from Unified Modeling Language (UML) sequence diagrams and object constraint language (OCL). In this paper we propose an automated test cases generation method from sequence diagrams, class diagrams, and OCL. The method supports UML 2.0 sequence diagrams including eight kinds of combined fragments describing control follow of systems. Test cases are generated with respect to the given concurrency coverage criteria. With strong concurrency coverage, generating exhaustive test cases for all concurrent interleaving sequences is exponential in size. The key idea of this method is to create selection of possible test scenarios in special case of exploring the message sequence with their possible interleaving in parallel or weak sequencing fragments. Test data for testing loop fragments is also generated. We implemented a tool to automate the proposed method and studied its feasibility and effectiveness. Experimental results show that the method can generate test cases on demand to satisfy a given concurrency coverage criterion and can detect up to 74.5% of seeded faults.

The objectives of this research focuses on four main contents:

(i) propose the method for generating test scenarios with respect to concurrency coverage criteria for testing concurrent behaviors.

(ii) propose test data generation procedure for each test scenario. The proposed procedure solves this test data for testing loops.

(iii) develop a tool to automate the proposed method with analyzing sequence diagrams in xmi file.

(iv) conduct case studies to validate the feasibility and effectiveness of the proposed method.

- Control-flow graph generation: Given UML 2.0 sequence diagrams describes behavior of SUT while class diagrams declares all method signatures and class attributes. Control-flow graph (CFG) generation from sequence diagram is used by a proposed recursive algorithm, and constraints of variables are derived from class diagram to generate test data. The generation of sequence diagram data structure creates queue which includes message, fragment and operand. The proposed iterative process is to generate different kinds of nodes from the queue. At each iteration, it analyzes each element of queue to create corresponding exit node, connect edge from current node to exit node, then exit node is considered current node. The method supports UML 2.0 sequence diagrams including eight kinds of combined fragments describing control-flow of systems.

- Test scenarios generation: Input of the test scenarios generation is CFG. The test scenarios denote abstract test cases which represent possible traces of executions. The output from the scenario generation is a finite set of scenarios which are complete paths starting from the initial node to the final node. Basic paths generated using DFS or BFS algorithm [6, 1] are suitable for node coverage and edge coverage of graph, but do not address the issues of the synchronization and data safety. When using that algorithm, we can not explore the message sequence with their possible interleaving of operands in par or seq fragment. The proposed algorithm generates test scenarios from CFG to solve that problem.

Algorithm: Generating the test scenarios
Input: Control-flow Graph G with initial node *in* and final nodes are f_{ni}
Output:T is a collection of test scenarios, t is a test path

```
1: T=∅, t=∅;
2: curNode=in; //current node starts from in
3: repeat
4:      move to next node
5:      if (curNode==BN) then
6:           t.append(BN);
7:      end if
8:      if (curNode==DN and decision==TRUE)then
9:           Append t with true part BN to merge node
10:     else
11:          Append t with false part BN to merge node
12:     end if
13:     If (curNode==FN) then
14:          active all sub paths of FN;
15:          repeat
16:              select random sub path;
17:              append t with node up to before or after node
                 having isAsyn property //message having isAsyn (true) is a
                 switch point
18:          until (all sub paths are empty)
19:     end if
20:     If (curNode==fni)
21:          T = T +{t};
22:     end if
23: until Graph end
```

Test data generation: The proposed method solves test data generation for testing loops by finding values in the test scenarios, using one predicate at a time and reducing domains of variables step by step. We develop the dynamic domain reduction procedure [6]. In the test data generation procedure, loops are handled dynamically. The procedure finds all the scenarios that contain at most one loop structure. It then marks those DNs that affect whether another iteration of the loop is made.Then as the test scenario is traversed,when the DN is encountered, the loop constraint and variables are checked dynamically to decide whether to continue with another iteration. Comparing with [6], if variables always satisfy in next iteration, our procedure exits the loops to generate test data when the DN is encountered in case of *1, 2, random n, max and min* loops.

Data safety error uncover capability: A test scenarios generated by algorithm should be able to uncover data safety errors. We use and compare DFS, BFS and our algorithm in generating the test scenarios from CFG. DFS algorithm generates test scenarios including test sequences that are not capable of finding data safety errors because it does not allow interleaving between the messages of two operands in par fragment. BFS algorithm does not generate the test sequences in testing loops while our method uses with two parts, false part and true part that means zero and more than one loops.

Fault -detection capability:

Table 1: The mutation score results for each test scenario

Level	Number of test cases	Test scenario 1	Test scenario 2	Test scenario 3	Test scenario 4
Method-level	*size* = 2	51.2%	40.5%	45.7%	50.2%
	size = 10/20/30	56.5%	55.7%	47.7%	60.4%
Class-level	*size* = 2/10/20/30	74.5%	74.5%	81.5%	81.5%

Table 2: The mutation score results of our method and random method

			Our method		Random Method	
Number of test cases	Level	Total mutants	Total killed mutants	MS	Killed mutants	MS
$size = 2$	Method-level	351	261	74.3%	139	39.6%
	Class-level	26	26	100%	23	88.4%
	Total	377	287	76.2%	162	42.9%
$size = 10/20/30$	Method-level	351	266	75.7%	153	43.5%
	Class-level	26	26	100%	26	100%
	Total	377	292	77.5%	179	47.4%

The paper presented the automated test data generation method based UML sequence diagrams, class diagrams and OCL. The method supports UML 2.0 sequence diagrams including eight kinds of combined fragments due to improved control follow graph generation technique. The key idea of this method is to generate selection of possible test scenarios in special case of exploring the message sequence with their possible interleaving in par or seq fragments. The test scenarios generation method also avoids test explosion by selecting switch point. Therefore, concurrency errors of systems can be found. In addition, the new point is to generate test data in testing loop fragments when comparing with current approaches, test data is generated in case of body of loop that is only executed once. The method supports different coverage criteria and can therefore test concurrent processes quite effectively. Finally, we have implemented a tool to automate the proposed method and conducted the case study to validate its feasibility and effectiveness.

We are investigating to determine infeasible or feasible test scenarios when there is no input data for them to be executed. We also are going to extend the proposed method for other UML diagrams (e.g., state-chart diagrams, activity diagrams). Moreover, we would like to further investigate and evaluate the fault-detection effectiveness, and costs, of the concurrency coverage criteria.

[1] M. Dhineshkumar and Galeebathullah (2014) An approach to generate test cases from sequence diagram, In Proceedings of the 2014 International Conference on Intelligent Computing Applications, ICICA 14, IEEE Computer Society, Washington, DC, USA, pp. 345- 349.

[2] Object Management Group (2006) The Unified Modeling Language UML 2.0 Technical Report formal/06-04-04, The Object Management Group (OMG).

[3] M. Khandai, A. Acharya, and D. Mohapatra (2011). A novel approach of test case generation for concurrent systems using uml sequence diagram, In Electronics Computer Technology (ICECT), 3rd International Conference, vol 1, pp. 157-161.

[4] Bao-Lin Li, Zhi-shu Li, Li Qing, Yan-Hong Chen (2007) Test case automate generation from uml sequence diagram and ocl expression, In Proceedings of the 2007 International Conference on Computational Intelligence and Security, CIS 07, IEEE Computer Society, Washington, DC, USA, pp. 1048-1052.

[5] H. Minh-Duong, L. Khanh-Trinh, and P.N. Hung (2013) An assume-guarantee model checker for component-based systems, In The 10th IEEE-RIVF International Conference on Computing and Communication Technologies, pp. 22-26.

[6] A. Nayak and D. Samanta (2010) Automatic Test Data Synthesis using UML Sequence Diagrams, Journal of Object Technology, vol. 09,no.2, pp. 115-144.

[7] M. Shirole and R. Kumar (2012) Testing for concurrency in uml diagrams, SIGSOFT Softw. Eng. Notes, vol 37, no.5, pp. 1-8.

[8] Mark Utting and Bruno Legeard (2006) Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[9] Thi Dao Vu, Pham Ngoc Hung and Viet Ha Nguyen (2015) A Method for Automated Test Data Generation from Sequence Diagrams and Object Constraint Language, In Proceedings of the Sixth International Symposium on Information and Communication Technology, ACM, Hue City, Viet Nam, pp.335-341.