International Journal of Scientific Research in Computer Science and Engineering

Research Paper

ISSN 2320-7639

Volume-1, Issue-1, Jan- Feb-2013

Available online at www.isroset.org

Test Case Generation Using Discrete Particle Swarm Optimization Algorithm

Chandraprakash Patidar

Department of Information Technology, Institute of Engineering & Technology DAVV Indore, M.P. - INDIA
chandraprakash_patidar@yahoo.co.in

Available online at www.isroset.org

Received: 04 Jan 2013 Revised: 16 Jan 2013 Accepted: 15 Feb 2013 Published: 28 Feb 2013

Abstract— Today's era is of software. To deliver the best software to the customer it is very important to test the software before delivery. So the software testing becomes an independent field of software engineering. The crucial section of software testing is to generate the test cases. In this paper I proposed a model which is based on sequence diagram. That generate more dynamic test cases because of sequence diagram. Whereas a testing which is based on use case diagram is generates only static test cases. In this paper information will be extracted from sequence diagram. For this first of all sequecne diagram is created on the given problem. After this dependency table will be generated. Dependency graph will be generated from dependency table. At last test cases will be generated from dependency graph. These test cases will be optimized by Discrete Particle Swarm Intelligence Algorithm.

Keywords—Sequence diagram, Test Case Discrete Particle Swarm Optimization, Visual Paradigm 12.0.

I. INTRODUCTION

Sequence diagram is typical control flow diagram. It concentrates on control flow through multiple interacting instances. For testing, the sequence diagram may be represented as abstract control flow graph that span multiple entities. With that respect we can apply all typical traditional control flow graph based test coverage criteria. Since UML diagrams are more abstract than traditional control flow graphs, the test targets may be more abstract. Binder identifies some typical problems that may be discovered through sequence diagram based testing [1]:

- Incorrect or missing output
- Action missing on external interface
- Missing function/feature (interface) in a participating object
- Correct message passed to the wrong object
- Incorrect message passed to the right object
- Message sent to destroyed object
- Correct exception raised, but caught by the wrong object
- Incorrect exception raised to the right object

Discrete Particle Swarm Optimization Algorithm [2]:

```
Initialize swarm P    t=0;
Initialize velocity $v_k^t$ and position $P_k^t$
Initialize parameters
Evaluate particles
Find the local best ${}^eP_k^t$ and global best $G_b^t$
Do
{
for(K=1, N)
Update Velocity $v_k^{t+1}$;
Update Position $P_k^{t+1}$;
Evaluate all Particles;
Update ${}^eP_k^{t+1}$ and $G^{t+1}$, (K=1, N);
$t \rightarrow t+1$;
}(while $t<t_{max}$)
```

The Particle Velocity and position are continuously updated using equation 1 and 2.

$$v_k^{t+1} = C_1U_1\ v_k^t + C_2U_2\ \text{rand}()\ ({}^eP_k^{t+1} - P_k^{t+1}) + C_3U_3\ \text{rand}()\ (G_k - P_k^{t+1}) \quad \text{---------}\ (1)$$

$$P_k^{t+1} = P_k^t + v_k^t \quad \text{---------}\ (2)$$

Where C_1, C_2 and C_3 are acceleration constants. The acceleration constants C_1, C_2 and C_3 in equation 2 guide every particle toward local best and the global best solution during the search process. Low acceleration value results in walking far from the target, namely local best and the global best. High value results in premature convergence of the search process.

The present study is organized as following: Section 2 reviews the activities for software testing by Sequence diagram and Discrete Particle Swarm Optimization algorithm. The first part of section 3 explains completely the concepts and algorithm stages for full explanations and also deals with the explained approach concepts, analyzes the TC generation method by this approach and an example for better understanding is evaluated. Also the last part in section 3 compares the results with test case generation using Genetic algorithm. Finally the last section provides the conclusion.

II. RELATED WORK

The literature survey is done in order to collect information about the basic system and the various algorithms and technologies that can be used for the further references while developing the project. In this chapter we present the research papers related to the Model i.e. sequence diagram based testing. Emanuela G. Cartaxo, Francisco G. O. Neto and Patrícia D. L. Machado [3] presented a systematic procedure based on model based testing technique in which test cases are generated from UML Sequence diagrams translated into Labeled Transition Systems (LTSs). Labeled transition systems provide a global indivisible description of the set of all possible behaviors of the system; a path on the LTS can be taken as test sequence. Therefore, Labeled Transition Systems are highly testable models. In figure 2.1, the elements that are present in a Labeled Transition System are shown: (a) initial state (Figure 2.1(a)) that represents an initial state of the system; (b) labeled transition (Figure 2.1(b)) that represents an action that occurs and change the state of the system; and (c) state (Figure 2.1(c)) that represents a state of the system. An Labeled Transition Systems is a 4-tuple S = (Q, A, T, q_0), where

- Q is finite, nonempty set of states;
- A is a finite, nonempty set of labels (denoting actions);
- T, the transition relation, is a subset of Q×A×Q;
- q_0 is the initial state.

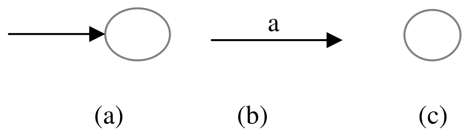

Figure 1: Labeled Transition System Elements.

In this paper Redundancy in Test Case Generation which will be removed through my approach.

S. Shanmuga Priya and P. D. Sheba Kezia Malarchelvi [4] presented a model based testing approach in which test paths are generated by traversing the sequence dependency graph using depth first search technique; which is created using the information extracted from UML Sequence diagrams in the form of sequence dependency table using depth first search.

Symbol	Activity Name	Sequence Number	Dependency	Input	Expected Output
A	Login	1	-	Patient ID and Password	Valid Patient ID and Password
B	Verify	2	A	-	Validate Patient ID and Password/Invalid Patient ID or Password
C	Result	3	B	-	Take to the next screen on entering valid Patient ID and Password
					End on entering invalid patient ID or password
D	Patient Details	4	C	Patient enters Patient Name, Age, Gender, Last Consulted Date, Symptoms, Doctor Name	Checks for Details (Valid)
					Invalid Details (End)
E	Request	5	D	-	Proceeds if doctor is available
					End if doctor is not available
F	Refer Patient History	6	E	-	Checks for Details
G	Retrieve Data	7	F	-	Retrieve and Display Patient History
H	Diagnosis/ Suggestions	8	G	-	Doctor make diagnosis and prescribe medicine
					Suggest to take some other tests for further diagnosis
I	Display Result	9	H	-	Patient takes prescription/ Suggested Medical Test
J	End	-	C, D, E, I	-	-

Table 1: Sequence Dependency Table

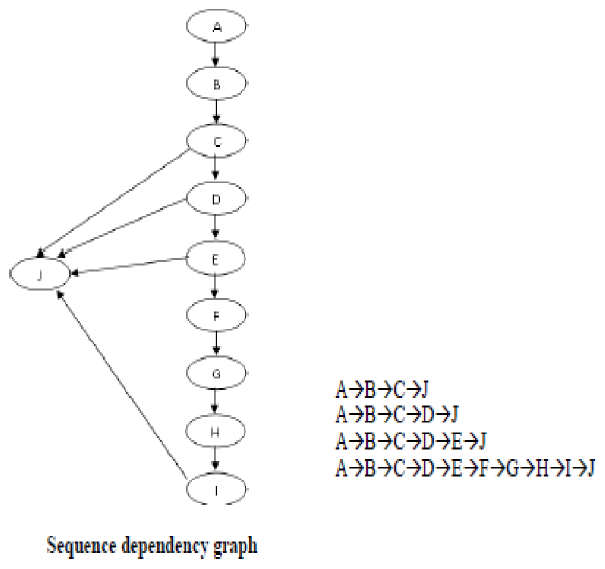

Figure 2: Generated Test Paths using Depth First Search.

In this paper Manual Test Case generation with test paths which will be automated in our approach.

Bandyopadhyay, A. and Ghosh, S. presented in [5] a novel testing approach using sequence models to extract message paths and than using state machines for generating multiple execution paths from a message path for analyzing the effect of messages on state transitions of the system [3]. Here also Manual Test Cases are generated with test paths**.**

Vivek Kumar and Aarti Gautam Dinker presented in [6] a model based testing approach using UML use case diagram for Web Application including ordered profile use case diagrams called Use Case Transition Models. In their approach testing strategy derives test cases using full predicate coverage criteria. In this paper Static Test Cases are Generated and which are made more dynamic in our approach.

Manpreet Kaur and Rupinder Singh presented in [7] the conditional predicates linked with the messages in Sequence Diagram and created a Slice with each conditional Predicate and than with respect to each slice test cases are generated manually for satisfying slice condition. Here the Slicing technique is used to disintegrate the structure of the system model into sub-models without affecting their original structure and functionality. In this paper also Test Cases are generated manually with respect to each constructed slice by satisfying slice condition, these are automated in our approach.

Ashlata Nayak and Debasis Samanta presented a two phase approach for automating the test Scenario Generation Process using Sequence Diagram. A Control flow analysis of a sequence diagram is presented in the first phase and generates as outcome a directed graph representation i.e. Scenario Graph and this graph will be the input for second phase and will generate test scenarios as the final output [8].In this paper Test Scenarios are not optimized and will be optimized in our approach.

Swain et al. Presented test suite generation by constructing Use Case Dependency Graph from Use Case Diagram and Concurrent Control flow graph from corresponding Sequence Diagram. And this test suite aims to cover different faults associated with Object Oriented Systems for which Use Case Dependencies and Inter and Intra-Sequence Diagrams are considered [9].

Sven Sieverding, Christian Ellen and Peter Battram Presented an approach to analyze sequence diagram based test cases. This is achieved by using test case sequence diagram and for translating this test case sequence diagram to Timed Arc Petri Nets transition rules are presented. These timed arc Petri nets can be merged into one single Petri net. By these merged Petri nets they analyzed consistency of test cases in terms of ordering and timing behavior [10].

A. V. K. Shanthi and G. Mohan Kumar presented optimized test case generation by UML sequence diagram using genetic algorithm [11].
Clémentine Nebut, Frank Fleurey, Yves Le Traon, and Jean-Marc Jézéquel [12] presented the test case generation using Use Case, Use Case contracts and Use Case Scenarios. They also show results of their approach on three case studies named Automated Teller Machine, A FTP Server and The Sever of Virtual Meetings. Which can be refined using sequence diagram which is a more dynamic representation than Use case diagram and they also mentioned that more efficient criteria for test generation and test efficiency measures should be developed and which is done by our approach and so this research gap overcome in our project by sequence diagram and optimal test case generation algorithm Discrete Particle Swarm Optimization.

Azam Andalib and Seyed Morteza [13] presented the algorithm which optimizes the test cases by Discrete Particle Swarm Optimization Algorithm. Here test cases are generated from code which can be generated early in the software development life cycle using Sequence Diagram.

III. OBJECTIVE

The objective of this project is to resolve the problems with other existing software testing, test case generation algorithms. The objective of this paper using Sequence Diagram Based Testing with discrete particle swarm optimization algorithm approach is as follows:

1. Automated Test Case Generation: This project focuses on automated test case generation from UML Sequence Diagram using Discrete Particle Swarm Optimization Algorithm. The proposed approach introduces an algorithm that automatically creates a dependency table then it generates a dependency graph from which test cases can be easily generated and then those test cases can be optimized using discrete particle swarm optimization algorithm.
2. Dynamic Test Case Generation: This Paper focuses on generating dynamic test cases by using sequence diagram as base element for generating test cases. This is possible by proposed approach because using sequence diagram dynamic actions like interaction among objects can be taken into consideration when generating test cases.
3. Optimized Test Case Generation: In this paper Test cases are optimized using discrete particle swarm optimization algorithm. Which is applied on the test cases generated by dependency graph (control flow graph) generated by extracted information from sequence diagram in the .xml file.

IV. PROPOSED APPROACH

Firstly information is extracted from sequence diagram using visual paradigm tool that will export .xml file from .mdl file than dependency table will be generated based on extracted information. Now from dependency table control flow graph will be generated and from control flow graph test paths can be identified. On these test paths Discrete Particle Swarm Optimization Algorithm will be applied for test case generation. The set of optimized test case sequences are now applied on the source code (through any automated test suit execution program) and the number of faults found are analyzed against the existing algorithms in the literature.

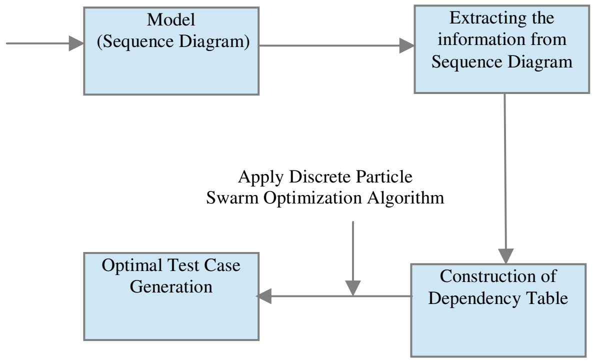
Model
(Sequence Diagram)
Extracting the information from Sequence Diagram
Apply Discrete Particle Swarm Optimization Algorithm
Optimal Test Case Generation
Construction of Dependency Table

Figure 3: Architecture of the System.

We are taking ATM case study as example to clear our approach. Firstly in our approach we have to draw sequence diagram for our example than we have to export the Sequence diagram into xml file for extracting the information which will be helpful in constructing sequence dependency table. And from this sequence dependency table we will be able to draw the control flow graph and from that control flow graph we will be able to generate test cases. Following this the test cases can be optimized by using Discrete Particle Swarm Optimization Algorithm. Different diagrams and tables as per procedure defined in our approach are as follows that include Sequence diagram, Sequence diagram to xml exported file using visual paradigm 12.0 tool, Sequence dependency table and control flow graph:

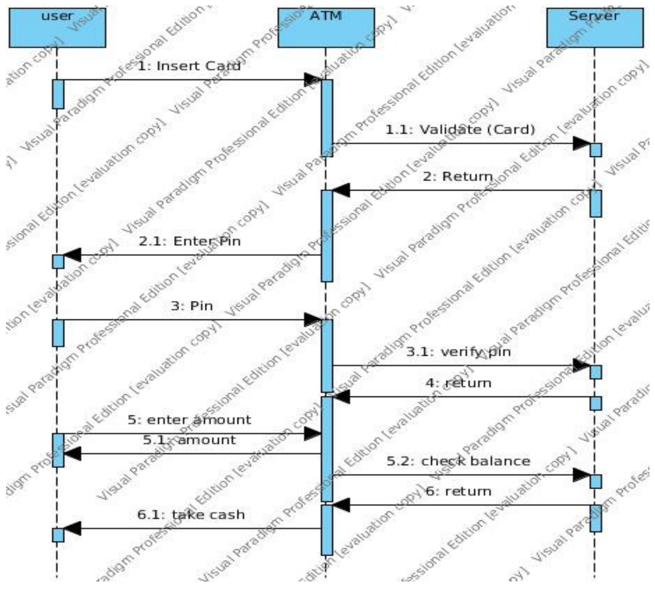

Figure 4: Sequence Diagram for ATM Case study.

Symbol	Activity Name	Sequence Number	Dependency	Input	Expected Output
A	Insert Card	1			
B	Validate (Card)	2	A	Card	True (enter the pin)
					False (end)
C	Enter Pin	3	B	Pin	Pin
D	Pin	4	C	User types the pin	Receives the pin and send it to the server
E	Verify (pin)	5	D	Pin	True (enter amount)
					False (end)
F	Enter amount	6	E	Amount	Amount
G	Amount	7	F	User types the amount	Receives the amount and send it to the server
H	Check Balance (amount)	8	G	Amount	True (take cash)
					False (end)
I	Take cash	9	H	Cash	End
J	End	10	B, E, H, I		

Table 2: Sequence Dependency Table

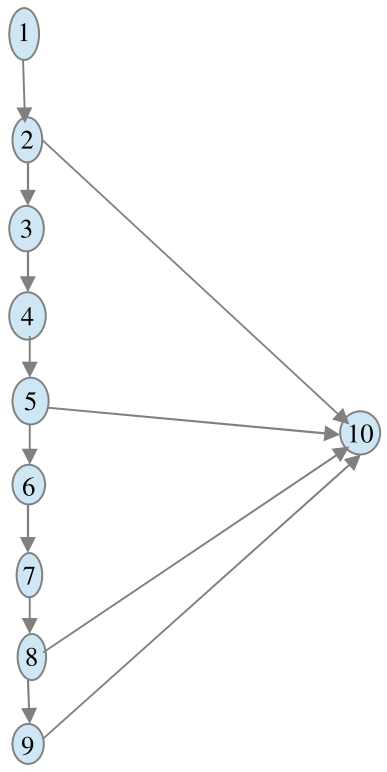

Figure 6: Control Flow Graph

According to McCabe theory as shown in [14] four test cases are possible from above control flow graph. All Possible Test Paths are as follows:

P1: 1-2-10

P2: 1-2-3-4-5-10

P3: 1-2-3-4-5-6-7-8-10

P4: 1-2-3-4-5-6-7-8-9-10

Binary and Decimal Value of Pathways

P1: F = (0001): 1

P2: F = (0010): 2

P3: F = (0011): 3

P4: F = (0100): 4

Ideal Value of fitness function is
$F_{1\ \mathrm{ideal}}$ = (0100): 4

V. RESULTS

As we know from [12] that the path with maximum fitness value will be the optimum path; we can conclude by above shown paths and fitness value that the path 4 with fitness value 4 will be the optimum path and should be generated when we test the data given by user on the System under test. As the purpose of performing this algorithm to achieve maximum fitness value, the aim of the particles is achieving fitness value 4, it can be said the path 4 is covered. By achieving such testing data 100% of path and 75% of the program regions are covered as in the case of considered case study.
Comparison of the Proposed Method with Test case generation using Genetic Algorithm:

Discrete Particle Swarm Optimization algorithm is much simpler than Genetic algorithm as it does not have the Genetic algorithms “cross” and “mutation” operation. Generally as in Genetic algorithm there is no selection operation that means none of the particles are eliminated and only the value of each particle is changed.

VI. ADVANTAGES AND DISADVANTAGES

In this approach test cases are generated early in the software development life cycle and are more optimized than currently present test case generation approaches but as there is further scope of enhancement in optimizing the test case so this can be considered as a limitation of the given approach.

VII. FUTURE SCOPE

In future optimization in generating test cases can be increased by applying the Hybrid Test Case Generation algorithm using both Genetic algorithm and Discrete Particle Swarm Optimization Algorithm as already shown in many papers from generating test cases from Code that can further enhanced and the same approach can be applied to Model i. e. Sequence diagram based test case generation.

VIII. CONCLUSION

In this paper a method for automatic and more dynamic generation of testing data based on Model i. e. Sequence Diagram is presented. Here the dependency is not on the flow but we assigned different discrete values to different paths based on loops present or not and than our fitness value will be calculated based on these discrete values and so if the fitness value will be the maximum value among these values or will differ as per if the loops are present in the case considered as in [12].

REFERENCES

[1]. R. Binder, “Testing Object Oriented Systems: Models, patterns and tools”; Addison-Wesley, 2000.

[2]. S. G. Ponnambalam, N. Jawahar and S. Chandrasekaran, “Discrete Particle Swarm Optimization Algorithm for Flowshop Scheduling”.

[3].Emanuela G. Cartaxo, Francisco G. O. Neto and Patrícia D. L. Machado; “Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems”, SMC 2007: 1292-1297.

[4]. S. Shanmuga Priya and P. D. Sheba Kezia Malarchelvi; “Test Path Generation Using Uml Sequence Diagram”, IJARCSSE, 2013.

[5]. Bandyopadhyay, A. and Ghosh, S.; “Using UML Sequence Diagrams and State Machines for Test Input Generation”, 19th International Symposium on Software Reliability Engineering, ISSRE, 2008.

[6]. Vivek Kumar, Aarti Gautam; “Improving Testing Architecture for MVC Based Application”, IJARCSSE, February 2014.

[7]. Manpreet Kaur, Rupinder Singh; “Generation of Test Cases from Sliced Sequence Diagram”, IJCAR, July 2014.

[8]. Ashalatha Nayak and Debasis Samanta; “Synthesis of Test Scenarios Using UML Sequence Diagrams”, ISRN, February 2012.

[9]. Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall; “Test Case Generation Based on Use case and Sequence Diagram”, IJSE, July 2010.

[10]. Sven Sieverding, Christian Ellen and Peter Battram; “Sequence Diagram Test Case Specification and Virtual Integration Analysis using Timed-Arc Petri Nets”, FESCA, 2013.

[11]. A. V. K. Shanthi and G. Mohan Kumar; “Automated Test Cases Generation from UML Sequence Diagram”, ICSCA 2012.

[12]. Azam Andalib and Seyed Morteza, “A New Approach for Test Case Generation by Discrete Particle Swarm Optimization Algorithm”; ICEE, 2014.

[13]. Clémentine Nebut, Frank Fleurey, Yves Le Traon, and Jean-Marc Jézéquel; “Automatic Test Generation: A Use Case Driven Approach”, IEEE Transactions on Software Engineering, Vol. 32, No. 3, March 2006.

[14]. Watson, A. H., T. H. McCabe, and D. R. Wallance, “Structured testing: A testing methodology using the cyclomatic complexity metric”, NIST special Publicaton, 1996.