Volume 3, Issue 4, April 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Test Path Generation Using Uml Sequence Diagram

S. Shanmuga Priya
PG Student, CSE
J.J. College of Engineering and Technology,
Trichy, India.

P. D. Sheba Kezia Malarchelvi
Professor, CSE
J.J. College of Engineering and Technology,
Trichy, India.

Abstract — Software testing, the inevitable phase in the Software Development Life Cycle (SDLC) plays a vital role in deciding the delivery of the product as well as to ensure the quality of the product. Testing can be performed on requirement, design and code. However, if testing is followed in the initial phase in SDLC most of the errors can be eliminated and can be prevented without disseminating to the next phase. So, testing must not be isolated to a single phase alone in SDLC. Test case generation is the challenging part in software testing process. The proposed work presents a model based testing approach from which the test paths are automated and obtained before or during the development process and so, when the application code is available, the test cases can be executed that aids in fixing the errors at initial phase. Unified Modeling Language (UML) Sequence Diagram is considered for designing and a case study of Medical Consultation System is taken for the proposed work.

Keywords— Sequence Diagram, Test Case, UML, Model Based Testing, Test Path.

I. INTRODUCTION

Almost everything we utilize in day today life has an element of software in it. An organization that develops any form of software product or service must put in every effort to drastically reduce and eliminate any defects in each delivered product or service. Users are intolerant to the hit-and-miss approach that characterized software products. From the view of a software development organization also, it may not be economically feasible to deliver products with defects are unlikely to remain latent for long. The consequences and impact of every single defect needs testing, especially for mission critical applications. It may be acceptable to say that 99.9% of defects are fixed in a product for a release, and only 0.1% defects are outstanding [1]. For such desired activity to put in existence, a complete software testing is the only plausible solution that is put forth by most of the IT sectors. The nature of usage of a product or service is becoming increasingly unpredictable. When specially made software is developed for a specific functions for a specific organization (for example, a payroll package), the natural practice of the product can be predictable, as users can only exercise the definite functionality provided in the software. On the other hand, consider a generic application hosted on the Internet. The application developers have no control over how someone will make use of the application. They may exercise untested functionality; they may have improper hardware or software environments; or they may not be fully trained on the application and thus simply use the product in an incorrect or unintended manner that could result in serious cause. The major need for software testing is it potentially close the gap between what is specified to be produced, meaning the documented requirements and internal IT standards, and what is actually built. The figure 1 shows how defects from early phase add to the cost of the software project. This implies that defects, when introduced in the requirements phase that might occur due to the misunderstanding or inconsistent requirements may inquire additional costs in detecting and correcting those errors.

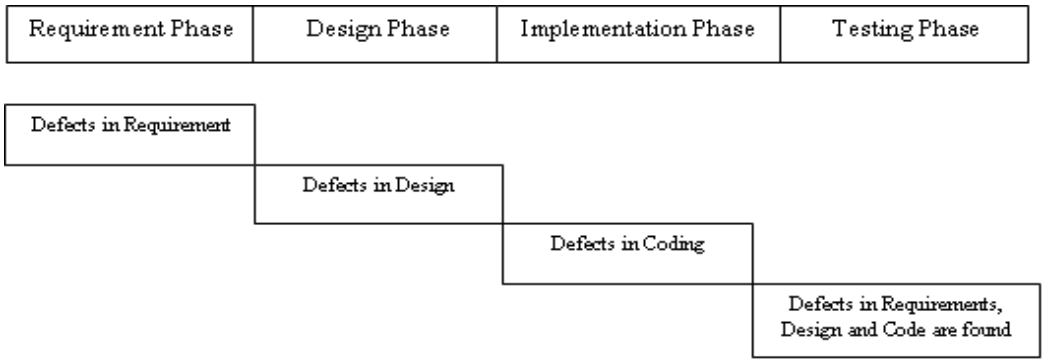

Fig. 1 Defects from early phase adding to the cost

A software project is made up of a series of phases. Generally, software projects comprise the phases as: i) Requirements gathering and analysis, ii) Planning, iii) Design, iv) Development or Coding, v) Testing, and vi) Deployment and Maintenance. Each and every phase has its own functionalities, milestones and deliverables to be achieved, in order to

ensure a product that meet the needs of the customer. Everything else is secondary. Software testing is an activity that should be done throughout the whole development process [2]. Figure 2 shows the traceability matrix structure, which insists that the testing is associated with requirements, design and coding phases of a software project.

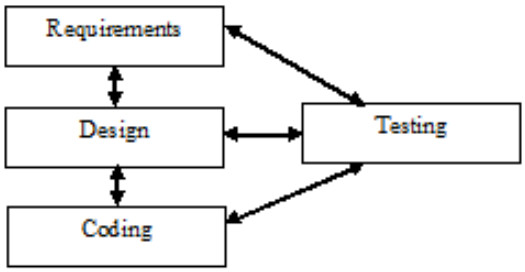

Fig. 2 Traceability matrix structure

The fundamental and time consuming part in testing is test case development. Hence it could be automated in order to save time. There are various techniques used in automatic test case generation which includes SBST (Search Based Software Test case generation), FSM (Finite State Machine), EFSM (Extended Finite State Machine), MBT (Model Based Testing), Petri Nets etc. In industries, Model Based Testing is gaining its popularity. MBT is a testing technique which considers the run –time behaviour of the system under test against the formal specification made. Behavior of the system can be described by means of input sequence, actions performed, conditions used, expected output and data flow from input to output. The approach followed in Model Based Testing is: i) Create an artefact model that is to be tested. ii) Explore the model to access its properties. iii) Establish achievable criteria for test coverage based on the created model. iv) Generate the sequence of test set. v) Execute the tests. vi) Compare the actual output with the expected output. vii) Use coverage criteria to determine adequacy of testing. The major benefits of MBT are that the developer or tester could be able to construct a mental model of the system under construction which could be transformed into implementation or test result. UML model is gaining its popularity in the industrial sector due to its easy representation. The proposed work deals with an approach that automates the test path generation from UML sequence diagram.
The organization of this paper is Section II gives Related Work, Section III gives Proposed Work, Section IV gives Case Study of Medical Consultation System considered for this work, Section V gives Simulation Result and Section VI gives the Conclusion

II. RELATED WORK

In [3] Jeevarathinam and Antony Selvadoss Thanamani proposed an approach to generate test cases automatically from software specification. In [4], [5] methods were proposed to automatically generate test cases from the UML structure diagram. In [5], Shanthi and Mohan Kumar proposed a technique to generate test cases for object oriented software. The data mining technique was applied to generate optimal test cases. The proposed approach used UML Class diagram. From the diagram, the information were extracted and mapped into a tree structure to derive test cases. In [6] – [22], UML behavioral diagram such as use case diagram, activity diagram, state chart diagram, sequence diagram were taken to generate test cases. From the considered UML diagram, an appropriate intermediate graph was constructed. Then the graph was traversed in depth first or breadth first search manner to produce the test cases. In [23] - [30], a combinational approach was followed by considering both UML structure diagram and behavioral diagram to generate test cases. From the considered UML diagram, an appropriate intermediate graph was constructed from the diagrams taken. Constructed individual graphs were merged into single graph and was traversed in depth first or breadth first search manner to produce the test cases.

III. PROPOSED WORK

Path testing is a structural testing method traditionally followed in testing a system under test. The objective of path testing is to ensure that each and every independent path through the program is executed at least once. An independent program path is defined as one that traverses at least one new edge in the flow graph obtained every time of traversal. Both basic and alternate paths must be executed in order to have a complete coverage. If the set of paths are properly chosen during testing, then it's the tester have achieved some measure of test thoroughness. Many such paths exist in testing a code. The proposed work gives the automated test paths derived from the UML model during design. The available test paths could give an idea to the software developer that he must ensure that those paths are properly coded during coding. The software tester might get an idea that the test cases to be developed must cover these generated paths in order to ensure complete coverage of the code. Sequence diagrams bring out the interactions among classes in terms of an exchange of messages over time. The lifeline in sequence diagram is represented vertically which depicts the sequence of events that are exchanged between the participants (can be instance of a class or an actor) during their course of interaction. An actor is an effective participant acting external to the system that is to be developed. There are two kinds of messages that could be shared between the participants which can be synchronous message or asynchronous message. The synchronous message is one in which the sender waits until it gets response before it proceeds further where as asynchronous message does not wait for the response. Figure 3 shows the flow diagram of the proposed system.

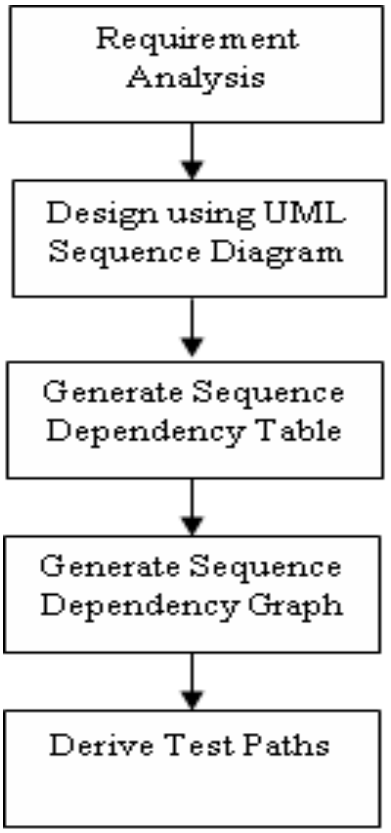

Fig. 3 Flow diagram of the proposed system

The proposed method has the following steps to derive the test path for medical consultancy system taken as case study:

i) Sequence Diagram for medial consultation system is designed using Rational Rose, IBM product and is saved with an extension as .mdl.
ii) Parse the .mdl file using java swing to get the necessary information that produces the Sequence Dependency Table (SDT).
iii) From the generated Sequence Dependency Table (SDT), using the objects as nodes, construct Sequence Dependency Graph (SDG).
iv) Traverse the Sequence Dependency Graph in Depth First Search (DFS) manner to derive the test paths.

IV. CASE STUDY : MEDICAL CONSULTATION SYSTEM

A. Problem Statement

The medical consultation system can be thought of as a computerized consulting service that can be an information guidance system. This service provides a means for the patient to get medical help they need without stepping to the doctor's door. The proposed medical consultation system allows patients and doctors to communicate with each other online. It is a convenient way to handle easy, non-urgent consultations and routine service requests either through computer. With this system, a patient (or the attendee of the patient on his/her behalf) can access the service from any computer. The patient must have a confidential account where he/she must provide username and password (if he/she is a registered user). On successful login the patient can produce the details such as the patient name, age, gender, last consulted date, doctor to whom they wish to consult (mandatory field), the symptoms they have (mandatory field). If the requested doctor is available, the doctor can refer to the patient's previous history from the diagnosis system and refer with the present symptoms given by the patient. If the details provided by the patient is adequate to diagnose the problem, the doctor is supposed to recommend treatment to the patient for cure else the doctor can suggest the patient to take additional prescribed tests and submitting the reports in person for further consultations.

B. Constructing Sequence Diagram

Figure 4 shows the sequence diagram of the proposed system designed using Rational Rose, IBM product and saved it with the extension .mdl. Here, the numbers represents the sequence of messages exchanged between the objects in the communicated order. Vertical dashed lines represent the sequence of events that occurs in the system.

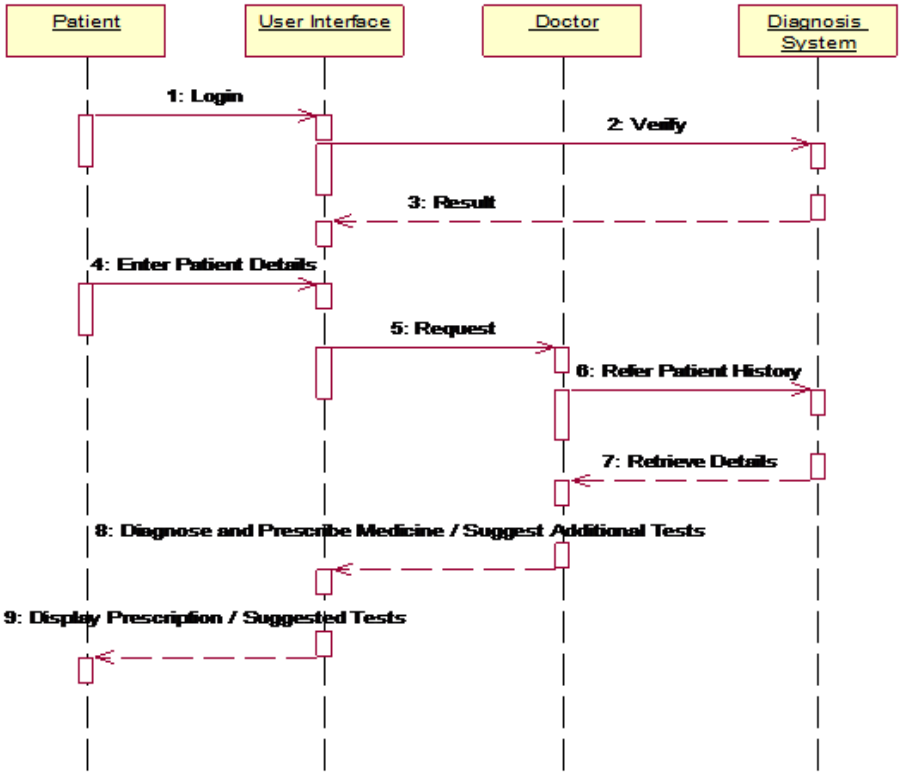

Fig. 4 Sequence diagram for medical consultation system

i) Login: Allows the patient to logon to the system by providing valid username and password.
ii) Verify: Verifies whether the entered username and password are valid by checking against the database.
iii) Result: Gives the result to the user interface. If the produced username and password gets matched the system authenticates the user and allows him/her to proceed further. Else the access will be denied indicating as invalid user.
iv) Enter Patient Details: The patient after successful login is expected to enter the personal details such as patient name, age, gender, last consulted date, doctor to whom they wish to consult (mandatory field), the symptoms they have (mandatory field).
v) Request: The system on getting the details of the patient connects to the doctor. If the doctor is available, he can respond to the patient, else, the patient will be notified with the unavailability of the doctor.
vi) Refer Patient History: If the doctor is available, he/she can send a request to the diagnosis system to refer to the patient's history to make diagnosis with the given symptoms.
vii) Retrieve Details: The information about the patient is retrieved.
viii) Diagnose and Prescribe Medicine / Prescribe Additional Tests: The doctor, after referring to the previous history of the patient and with the present entered symptoms, if adequate, can diagnose the disease severity and could prescribe medicines. Else if the symptoms are not adequate to diagnose the doctor could prescribe additional tests to be taken.
ix) Display Prescription/Suggest Additional Tests: The doctor's prescription can be sent to the user interface for displaying it to the patient.

C. Generating Sequence Dependency Table

From the constructed sequence diagram, information is extracted and Sequence Dependent Table is generated. Table 1 shows the Sequence Dependency Table (SDT) which contains six columns holding the information as follows:

i) Symbol: It is an alphabetic letter which is given for every activity involved.
ii) Activity Name: This field describes the activity carried out.
iii) Sequence Number: It gives the sequence number which gives the trace of flow that occurs when the messages were exchanged between the involving entities.
iv) Dependency: The Symbol(s) of each sequence that the current activity depends upon is given by the dependency. For example, the activity 'C' depends on Activity 'A' which means the result that is to be returned depends on the valid user name and password supplied by the Patient.
v) Input: The input gives the precondition that must hold.
vi) Expected Output: This gives the expected output of the current event that occurred.

TABLE I. Sequence dependency table

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	-	-

D. Generating Sequence Dependency Graph

From the sequence diagram, the objects and messages are collected. For every conditional message that occur one node and two edges are allocated, one edge for the basis flow (true) and other one for the alternate flow (false). For every other message present one node is allocated with one edge. Finally, the nodes and edges are connected in the sequence they occur. This gives the Sequence Dependency Graph. Figure 5 shows the Sequence Dependency Graph generated from the SDT.

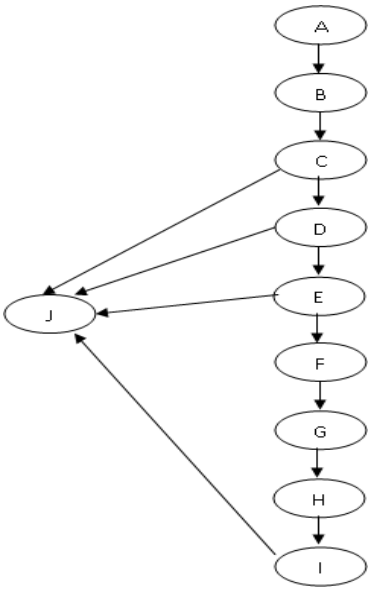

Fig. 5 Sequence dependency graph

E. Deriving Test Paths

From the generated Sequence Dependency Graph, it is traversed in Depth First Search manner to obtain the test paths. When derived the test paths, it could be easy for the software tester to generate test cases exercising these paths in order to ensure a complete coverage of the code. The generated test paths are:

A→B→C→J
A→B→C→D→J
A→B→C→D→E→J
A→B→C→D→E→F→G→H→I→J

V. SIMULATION RESULTS

The simulation of the test path generation using UML sequence diagram is done using java programming. The following screen shots show the various implementation details which gives the illustration of test path generation. Figure 6 shows the applet viewer depicting the generation of Sequence Dependency Table. First four fields as shown in the Table 1 are taken into consideration which is more than enough to generate the test paths as well the Sequence Dependency Graph.

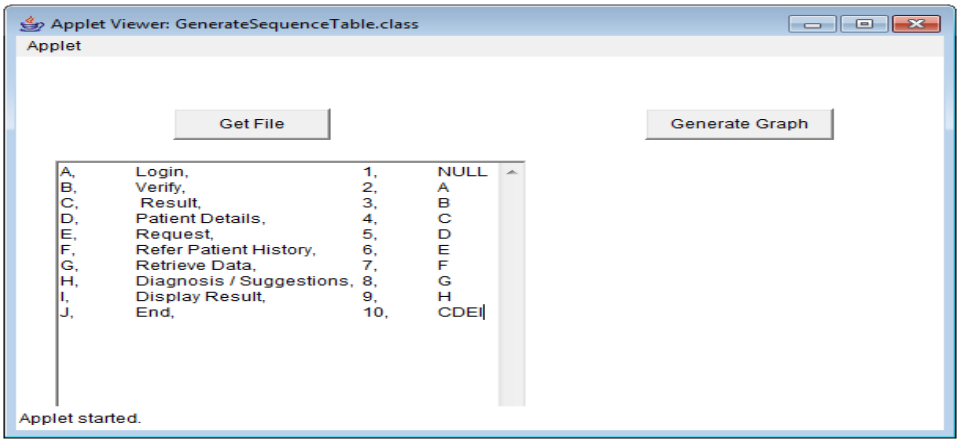

Fig. 6 Displaying sequence dependency table

Figure 7 shows the generated Sequence Dependency Graph from the Sequence Dependency Table, where for every conditional message one node occur and two edges emerge from it. In that two edges, one is to give the basis path i.e., the true flow and other gives the alternate flow i.e., the false flow. For every other message present one node is allocated with one edge. Finally, the nodes and edges are connected in the sequential order they occur in the sequence diagram.

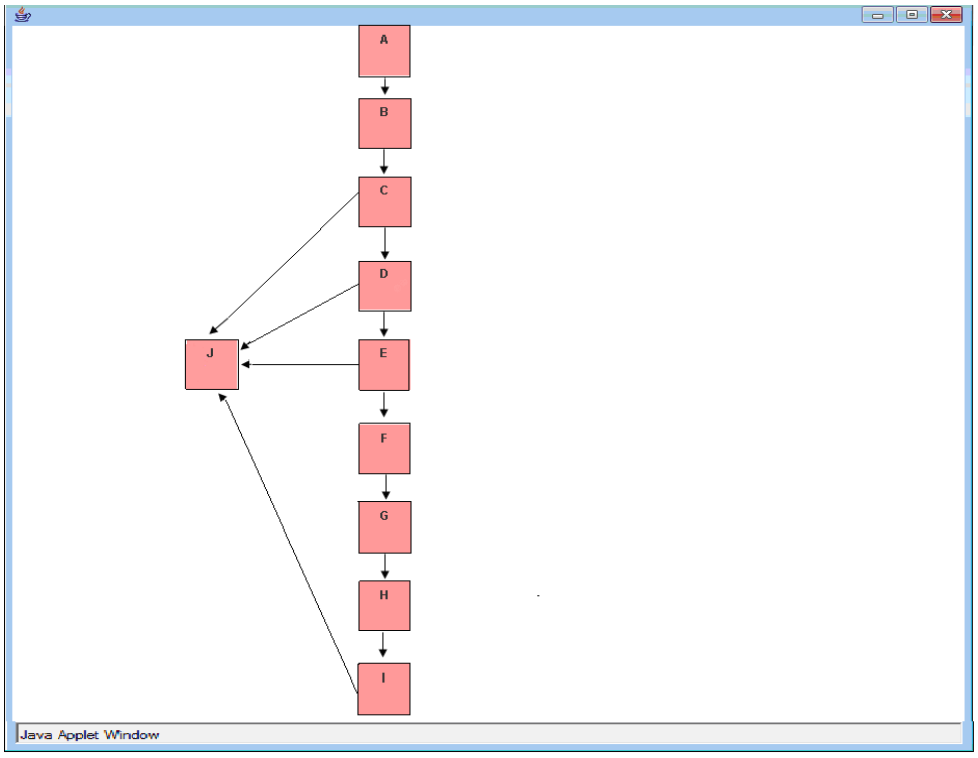

Fig. 7 Displaying sequence dependency graph

Figure 8 shows the generated test paths from the Sequence Dependency Graph by traversing the graph using Depth First Search technique. The independent paths that are obtained during the traversal are listed out which gives the various test paths for which the test cases could be generated while testing the system.

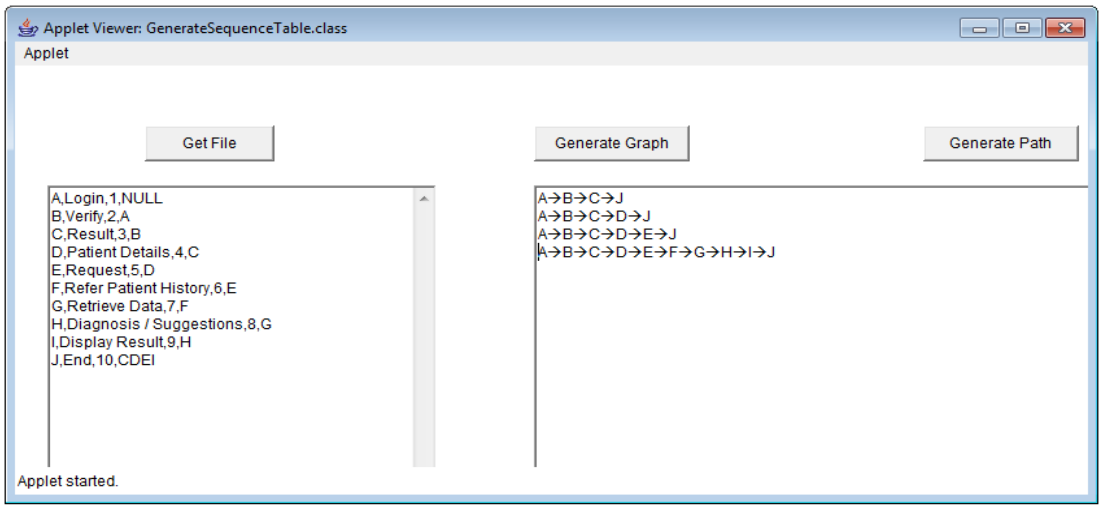

Fig. 8 Displaying generated test paths

VI. CONCLUSIONS

In this paper the test path generation by using UML Sequence diagram has been presented. During test case development, there might be enormous test paths available, and the software tester could get a confusion which path must be considered and which one to leave with. So, it is essential to get those basic paths covered in order to ensure complete coverage. This proposed approach helps in generating the test path for which the test cases could be produced by the software tester during testing to check whether the system works as intended. More over, the proposed approach assists the developer to significantly improve their design quality. They could find the faults in the implementation early and prevent those getting disseminated to the successive phases. As exhaustive testing is not possible due to time constraint, the test cases chosen to test a system can be minimal but should ensure maximum coverage required to test an application. In future, a method could be proposed to automate the generation of minimum number of test cases with the generated test paths.

REFERENCES

[1] http://my.safaribooksonline.com/book/software-engineering-and-development/software-testing.
[2] A. Bertolino, "Chapter 5: Software Testing", in IEEE SWEBOK Trial Version 1.00, May 2001.
[3] Jeevarathinam, Antony Selvadoss Thanamani, "Towards Test Case Generation from Software Specification", International Journal of Engineering Science and Technology, Vol. 2(11), pp. 6578-6584, 2010.
[4] M. Prasanna and K.R. Chandran, " Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm", International Journal on Advance Soft Computing Application, Vol. 1, No. 1, July 2009.
[5] A.V.K. Shanthi and G. Mohan Kumar, "Test Cases Generation for Object Oriented Software", Indian Journal of Computer Science and Engineering (IJCSE), Vol. 2, No. 4, Aug -Sep 2011.
[6] Noraida Ismail, Rosziati Ibrahim, Noraini Ibrahim, "Automatic Generation of Test Cases from Use-Case Diagram", Proceedings of the International Conference on Electrical Engineering and Informatics Institute Technology, Bandung, Indonesia June 17-19, 2007.
[7] Puneet Patel and Nitin N. Patel, "Test Case Formation using UML Activity Diagram", World Journal of Science and Technology, pp. 57-62, 2012.
[8] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba, "A Proposed Test Case Generation Technique Based on Activity Diagrams", International Journal of Engineering and Technology, Vol. 11, No. 03, June 2011.
[9] Debasish Kundu and Debasis Samanta, "A Novel Approach to Generate Test Cases from UML Activity Diagrams", Journal of Object Technology, Vol. 8, No. 3, pp. 65-83, June 2009.
[10] Luis Fernandez-Sanz, Sanjay Misra, "Practical Application of UML Activity Diagram for the Generation of Test Cases", Proceedings of the Romanian Academy, Series A, Vol. 13, No. 3, pp. 251-260, 2012.
[11] Andreas Heinecke, Tobias Bruckmann, Tobias Griebe, Volker Gruhn, "Generating Test Plans for Acceptance Tests from UML Activity Diagrams", 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, Vol. 14, pp. 425-654, 2010.
[12] Zhang Mei, Liu Chao, Sun Chang-ai, "Automated Test Case Generation Based on UML Activity Diagram Model", Journal of Beijing University of Aeronautics and Astronautics(in Chinese), Vol. 27, No. 4, pp. 433-437, 2010.

[13] Hyungchoul Kim, Sungwon Kang, Jongmoon Baik, Inyoung Ko, "Test Cases Generation from UML Activity Diagrams", Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing.

[14] Suppandeep Sandhu, Amardeep Singh, "A Systematic Approach for Software Test Cases Generation using Gray Box Testing with UML Activity Diagrams", International Journal of Computer Science And Technology Vol. 2, Issue 4, Oct-Dec, 2011.

[15] Mingsong Chen, Xiaokang Qiu, Wei Xu, Linzhang Wang, Jianhua Zhao And Xuandong Li, "UML Activity Diagram-Based Automatic Test Case Generation For Java Programs", The Computer Journal Advance Access, August 25, 2007.

[16] Johannes Ryser, Martin Glinz, "A Scenario-Based Approach to Validating and Testing Software Systems Using Statecharts", Presented at the 12th International Conference on Software and Systems Engineering and their Applications ICSSEA'99. Proceedings: CNAM, Paris, France.

[17] Chartchai Doungsa-ard, Keshav Dahal, Alamgir Hossain, and Taratip Suwannasart, "An Automatic Test Data Generation from UML State Diagram using Genetic Algorithm", The proceedings of the Second International Conference on Software Engineering Advances, 2007.

[18] S. Kansomkeat and W. Rivepiboon, "Automated-generating test case using UML statechart Diagrams", In Proc. SAICSIT 2003, ACM, pp.296 – 300, 2003.

[19] M. Prasanna, K.R. Chandran, K. Thiruvenkadam, "Automatic Test Case Generation for UML Collaboration Diagrams", http://www.jr.ietejournals.org/text.asp?2011/57/1/77/78373.

[20] P. Samuel, R. Mall and A. K. Bothra, "Automatic Test Case Generation Using Unified Modeling Language (UML) State Diagrams", IET Software, 2008.

[21] A. V. K. Shanthi and G. Mohan Kumar, "Automated Test Cases Generation from UML Sequence Diagram", International Conference on Software and Computer Applications (ICSCA 2012), Vol. 41, Singapore, 2012.

[22] Emanuela, G. Cartaxo, Francisco, G. O. Neto and Patrıcia D. L. Machado, "Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems", Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Montréal, Canada, 7-10, October 2007.

[23] Saru Dhir, "Impact of UML Techniques in Test Case Generation", International Journal of Engineering Science and Advanced Technology, Vol. 2, Issue – 2, pp. 214-217, March – April 2012.

[24] Monalisha Khandai, Arup Abhinna Acharya, Durga Prasad Mohapatra, "A Survey on Test Case Generation from UML Model", International Journal of Computer Science and Informational Technologies, Vol. 2(3), 2011.

[25] Vinaya Sawant, Ketan Shah, "Automatic Generation of Test Cases from UML Models", International conference on Technology Systems and Management (ICTSM), Proceedings published by International Journal of Computer Applications (IJCA), 2011.

[26] Arpita Tewari and A. K. Misra, "A Novel Approach to Generate Test Cases using UML Diagrams", International Journal of Software Engineering, Vol. 2, No. 2, pp. 109 – 124, 2011.

[27] Supaporn Kansomkeat, Jeff Offutt, Aynur Abdurazek, Andrea Baldin, "A Comparative Evaluation of Tests Generated from Different UML Diagrams", Ninth ACIS International Conference on Software Engineering, 2008.

[28] Dehla Sokenou, "Generating Test Sequences from UML Sequence Diagrams and State Diagrams", Proceedings of GI Jahrestagung (2), pp. 236-240, 2006.

[29] Yiwen Wang and Mao Zheng, "Test Case Generation from UML Models", 45th Annual Midwest Instruction and Computing Symposium, Cedar Falls, Iowa, April 2012.

[30] Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall, "Test Case Generation Based on Use case and Sequence Diagram", International Journal of Software Engineering, IJSE, Vol.3, No.2, July 2010.

