ALGORITHM AND IMPLEMENTATION OF MODEL BASED TESTING FOR PROTOCOL VERIFICATION

Arpita Mathur
Department of Computer Science, Lachoo Memorial College of Sc. & Tech.
A-Sector, Shastri Nagar, Jodhpur (India)

Abstract: In this paper we will explain how model based testing can be used to test the protocols that shows sequential (time based) temporal relationship between entities based on UML sequence diagram. First the dynamic model and temporal relationship are validated through model based testing. After validation this dynamic model becomes the oracle for validating the proposed protocol. An experiment was conducted on this by giving random inputs to the temporal relationship and dynamic model. Error was detected if there is difference in outputs of the two.

I. INTRODUCTION

An algorithm was developed for dynamic model based testing. Temporal relationships were validated through model based testing. Following algorithm tests the proposed protocol specification.

Step 1: Temporal relationship and dynamic model made through message sequence matrix are validated.

Step 2: Errors are injected / seeded in temporal relationship (TR).

Step 3: Random inputs are given to temporal relationship and dynamic model.

Step 4: The mismatch in output of temporal relationship and dynamic model denotes detection of error.

Step 5: Repeate Step 3 to Step 4 1000 time and record the behaviour i.e. mismatch/match.

II. RESULT AND ANALYSIS

The experiment was conducted to model a given protocol. The model prepared is able to test the protocol at early stage of the development. In this experiment the ability of model is verified through error seeding technique. The experiment is run for 5 instances of single seeded error (as shown in table 1), for each error 1000 attempts were made and this process was iterated 20 times. The data generated through this attempt formed the statistical base to analyze the behaviors of random test cases. Below are drawn charts to show relation between the numbers of attempts required to kill error and the number of errors killed successfully. X- axis represents the data sets and Y – axis represents number of attempts.

Table 1 Seeded errors

Error number	The error seeded
Error 1	t[1][0] made == instead of >
Error 2	t[3][2] made <= instead of >
Error 3	t[5][4] made don't care instead of >
Error 4	t[7][6] made >= instead of >
Error 5	t[4][3] made < instead of >

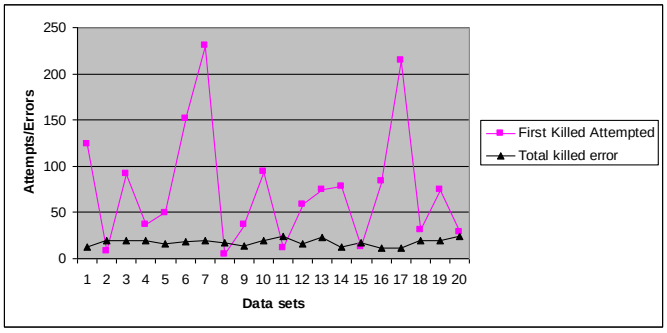

Figure 1: Error 1

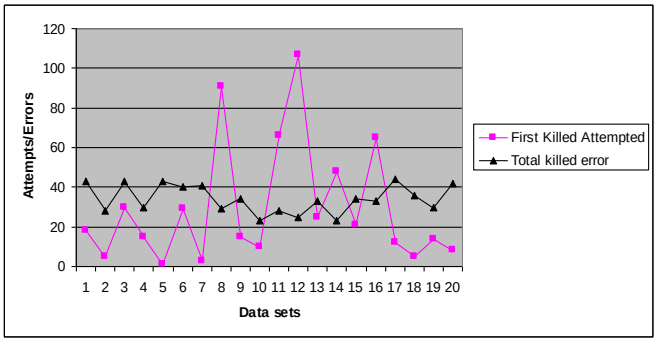

Figure 2: Error 2

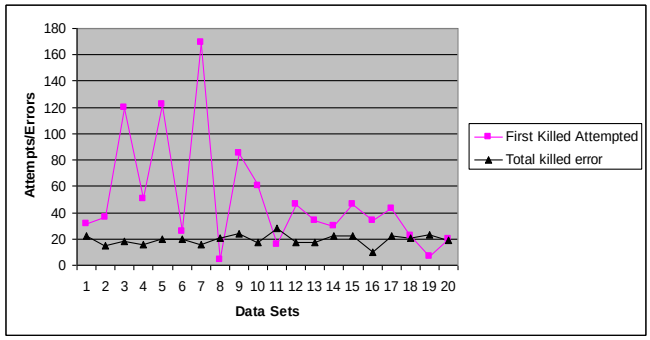

Figure 3: Error 3

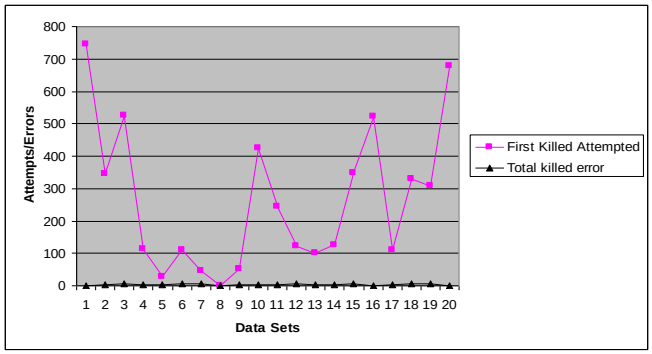

Figure 4: Error 4

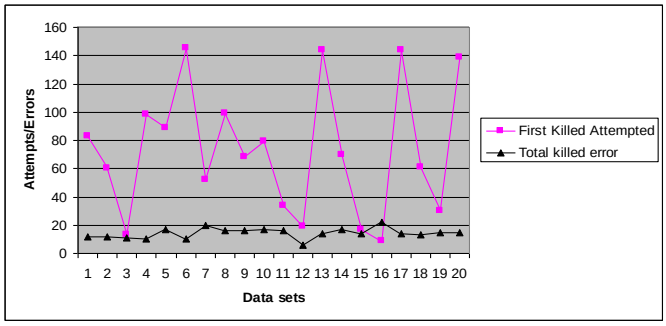

Figure 5: Error 5

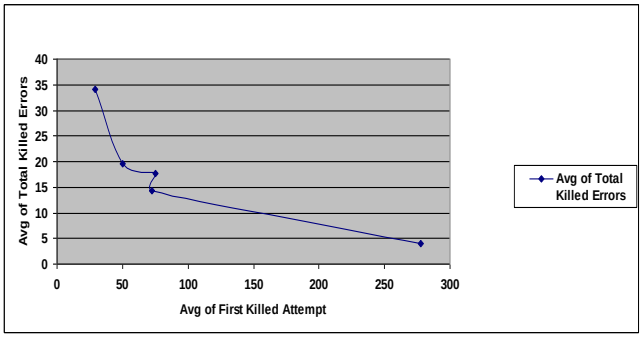

Figure 6: Relation between average of first killed attempt and average of total killed errors

III. CONCLUSION

By conducting the experiment, various results were recorded {Appendix A}. The results were visualized in line graphs as shown in fig 1 to 6. By analysis the results obtained through experiment, it is found that

- Number of time the seeded error successful caught i.e. total killed errors was almost constant for a specific error. The model behaves in the consistent manner for various set of random test cases. So confidence in randomization of input data is strengthened.
- Earliest first successful attempt to catch the seeded error was below 10 in most of the cases i.e. 4 out of 5. The dynamic model of given protocol catches error in few attempts and so the model as well as random testing is dependable.
- Number of first successful attempt in average is increased, as average of successful detection of errors is decreased, this verifies the statistical phenomena in the experiment. As per the statistics, frequent event will occur more times in a given duration or attempts. So the first successful attempt occurs early for easily detectable errors. [Figure 6]
- Out of total 100 attempts, each of 1000 random test cases, one exceptional case was observed, in which the seeded error was not caught. A further investigation shown that this exception occurred in the case of error no 4, the seeded error was ≥ (greater than or equal to) instead of >(greater than). It can easily be inferred that the error injected cover most of the correct inputs also hence malfunction in rare case. So the phenomenon of not detecting an error was due to nature of less likely occurrences.

REFERENCES

[1] J.J. Marciniak, "Encyclopedia on Software Engineering", Wiley, 2001
[2] Harry Robinson, "Intelligent Test Automation ", 2000
[3] Bill Hayduk, "Model-Based Testing for Java Applications ", 2007
[4] ITT Corporation, "Model Based Testing", 2006
[5] Grady Booch, James Rumbaugh and Ivar Jacobson, "The Unified Modeling Language User Guide", second edition, 2007
[6] Perdita Stevens & Rob Pooley, "Using UML, Software Engineering with Objects and Components", second edition, 2006.
[7] Martin Fowler & Kendall Scott, "UML Distilled, A Brief Guide to the Standard Object Modeling Language", first edition, 2005.
[8] Hans-Erik Eriksson, Magnus Penker, Brain Lyons, David Fado, "UML 2 Toolkit", OMG Press, Wiley, second edition, 2007, ISBN 10: 81-265-0466-8.
[9] Dehla Sokenou, "Generating Test Sequences from UML Sequence Diagrams and State Diagrams", GEBIT Solutions.
[10] 'Unified Modeling Language Specification", Version 2.0, OMG, 2004.
[11] Lu Luo, "A UML Documentation for an Elevator System Distributed Embedded Systems", PhD Project Report, December 2000.
[12] S. Ntafos, "On Random and Partition Testing", Procedings of International Symposium on Software Testing and Analysis (ISSTA), 1998, pp. 42–48.
[13] Ibrahim K. El-Far and James A. Whittaker, "Model-Based Software Testing", Florida Institute of Technology.

