A Short Report from an Experiment on Improving Functional Test Coverage Maximization in the Automotive Industry

Rashid Darwish, Lynnie Nakyanzi Gwosuta and Richard Torkar
Chalmers and the University of Gothenburg, Gothenburg, Sweden
Email:{rashid.darwish, gnlynnie}@gmail.com, torkarr@chalmers.se

***Abstract*—In the automotive industry, as the complexity of electronic control units (ECUs) increase, there is a need for the creation of models that facilitate early tests to ensure functionality; but there is little guidance on how to write these tests in order to achieve maximum coverage. Our prototype CANoe$^+$, which builds on the CANoe and GraphWalker tools, was evaluated against CANoe with regard to coverage maximization of generated test cases.**

I. Introduction

As software in Electronic Control Units (ECUs) gets more and more complex, there is an ever- increasing need for efficient testing processes in the automotive industry. Automated model-based software testing has been proposed as one solution, but due to the complexity of the systems being built, there is a challenge to generate test cases automatically that are effective and have a high functional test coverage [4].

With this in mind, creating a method that enables the developers to model the behavior of the desired system using graph theory techniques and generate automated test cases to intelligently test the system functionality is most sought after. Addressing this challenge will provide an insight into the possibilities and limitations of using model-based testing with graph theory techniques to generate effective test cases that have a maximum functional test coverage.

The knowledge will be applied in the automotive industry by capitalizing on the advantages of two tools, i.e., using CANoe [2] and GraphWalker [5] to generate and execute test cases with maximum functional test coverage. By maximum functional test coverage we mean how much of the system's functionality is exercised by the generated test case.

GraphWalker is a model based testing tool that uses the command line to read models in the form of finite state diagrams or directed graphs and generates tests from the models, either offline or online whereas CANoe is the most widely used comprehensive software tool for development, test and analysis of entire ECU networks and individual ECUs in the automotive industry today.

This will, in the end, verify the system requirements by the use of models and validate that the system under test meets the customer's needs. Additionally, our assumption is that in the long run this approach will reduce the costs of regression testing and the developer efforts will be channeled to, e.g., exploratory and negative testing.

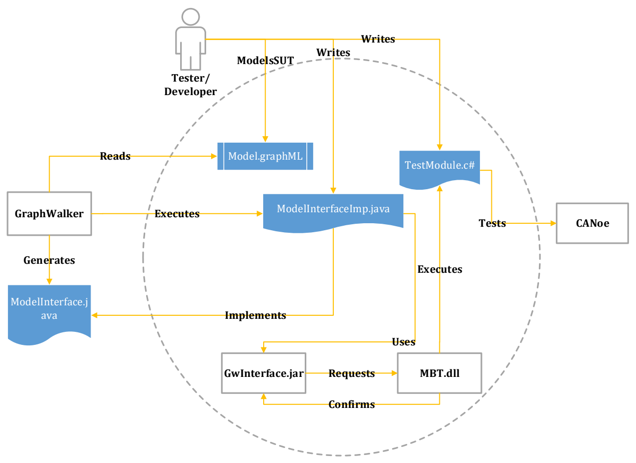

Fig. 1. Context diagram for CANoe$^+$

II. Proposed Solution

As CANoe presently executes automated test cases in a sequential manner with no notion of randomness, CANoe$^+$ is the proposed solution to solve the challenge at hand. CANoe$^+$ is the result of integrating the GraphWalker open source tool with CANoe where these two tools then communicate through an interface in order to generate test cases from the model. The test cases are then executed in CANoe to test the system functionality. However, in addition to the above tools, the solution includes a model and its implementation. There are three major steps involved in writing tests with CANoe$^+$ as can be seen in Fig. 1;

CANoe$^+$ models the functionality that is to be tested using the yEd desktop application [6]; a model of the expected behavior of the system under test is drawn as an extended finite state machine with states and transitions. The saved model is provided to the GraphWalker tool as an input of the expected states, transitions and the values that need to be tested.

GraphWalker is connected to CANoe by a communication bridge and for the purpose of this communication, the generated interface is implemented. GraphWalker provides various traversing strategies, we used 100% edge traversing strategy to cover the complete model. A model entails one or more functionality, if we cover the complete model then we obtain

100% functional coverage by the generated test cases for the intended functionality.

CANoe includes a simulated CAN network, and a database which holds the values of the network. The simulated CAN network in our context is the system under test. CANoe is responsible for the test framework and it is the test driver that we use to test the system under test. We used a .NET test module and its libraries provided by CANoe to implement the test module. The test module is the implementation of the adapter to the SUT, which should correspond 100% with the interface generated by GraphWalker in the previous step.

GraphWalker uses random functions to generate random test sequences from the model and hence, when it is executed, sends a request for a given function to be executed. While sending the request, it has the expected value appended to it. When the function to be executed in the test module is found, a comparison is made between the expected value from the model and the actual value from the CANoe database. A confirmation is sent back to GraphWalker to affirm that the functionality was executed and then CANoe produces a test report showing the verdict of the execution as pass or fail.

III. Analysis of Results

The prototype that was evaluated made use of sample ECU functionality that was simulated in the CANoe software. Arcuri and Briand [1] notes that the probability distribution of a randomized algorithm can be analyzed by running such an algorithm several times in an independent way, and then collecting the appropriate data about its results and performance. Due to the reason that CANoe$^+$ contains a notion of randomness, we evaluated the prototype ourselves as opposed to using human subjects. Coverage data of the two tools was collected and analyzed. We state our hypotheses as follows:

Null Hypothesis: CANoe is significantly better in functional coverage compared to CANoe$^+$, i.e., $H_0 : Cov_c > Cov_{c+}$

and,

Experimental (alternative) Hypothesis: CANoe$^+$ is significantly better in functional coverage compared to CANoe, i.e., $H_1 : Cov_c < Cov_{c+}$

A one-tailed test was used because our hypothesis is that our improvement, i.e., CANoe$^+$, would be better.

The sample functions were executed in 240 runs for each tool while recording the number of passed, failed and total test cases with respect to 100% edge traversing strategy of the model. Within these runs, fault injections were introduced of which data was collected. The collected data was analyzed in five scenarios as can be read from the full description [3]. One scenario presented here is the custom fault where a fault was injected in a door lock functionality that is responsible for unlocking a door. This fault would only be triggered if the unlocking feature of the door was executed two times successively, if any other action was executed between the clicks, the fault would not be triggered and the system under test would execute normally with no failure.

A. Evaluation of Results

Here we evaluate the custom fault scenario that showed a significant difference. This was an interesting observation as in CANoe$^+$ the affected functionality failed during all the runs but surprisingly CANoe reported 100% passed test cases without a single fault reported. This can be attributed to the fact that CANoe executes sequentially hence is not designed to reveal such faults.

A Shapiro-Wilk test for normality was applied ($p < .001$, $\alpha < 0.05$) rejecting the null hypothesis. As a consequence, the Mann-Whitney U test was used which rejected our null hypothesis ($U = 30, Z = -9.9801, p < 0.05, r = 0.91$). The test was used to evaluate the difference in the functional test coverage of the two tools and a significant difference was discovered.

The effect size calculations resulted in $\hat{A}_{12} = 0.0083$, hence indicating a large effect size. In short, in more than 99% of the cases CANoe$^+$ will, from a probabilistic point of view, be the winner when compared to CANoe.

IV. Conclusion

Our contribution is the prototype of the tool CANoe$^+$, which was developed with the aim to investigate if there is an increase in the functional coverage of model-based test cases with our new tool (CANoe$^+$) as compared to the current way of working, i.e., using only CANoe. Sample functions were developed and simulated in the CANoe software.

The sample functions were executed with the use of the two tools to determine which was best at uncovering faults and the fault finding capability. The functions were executed multiple times, i.e., 480 runs, while injecting faults to assess the fault finding capabilities for each of the tools. For each run, the number of failed test cases were recorded and later used in the analysis. The paper was reported as a controlled experiment, and the collected data was analyzed using analytical statistics.

We were able to reject the null hypothesis in favor of the alternative hypothesis as the results indicated the superiority of model-based testing approaches like CANoe$^+$ over testing methods like CANoe.

All data and source code is available at https://bitbucket.org/canoeplus/canoeplus.

References

[1] A. Arcuri and L. Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 1–10, New York, NY, USA, 2011. ACM.

[2] CANoe. https://vector.com/vi_ecutest_en.html. Accessed: 2016-12-07.

[3] R. Darwish and N. L. Gwosuta. A controlled experiment on coverage maximization of automated model-based software test cases in the automotive industry. Master's thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Sweden, 2016. 73.

[4] D. Fodor and K. Enisz. Vehicle dynamics based ABS ECU verification on real-time hardware-in-the-loop simulator. In *16th International Power Electronics and Motion Control Conference and Exposition (PEMC)*, pages 1247–1251, Sept 2014.

[5] GraphWalker. http://graphwalker.github.io. Accessed: 2016-12-07.

[6] yED. https://www.yworks.com/products/yed. Accessed: 2016-12-29.