Co-Evolution of Model-Based Tests for Industrial Automotive Software

Eric J. Rapos
School of Computing, Queen's University
Kingston, Ontario, Canada
Email: eric@cs.queensu.ca
PhD Advisor: James R. Cordy

I. Introduction

The concept of co-evolution refers to two (or more) objects evolving alongside each other, such that there is a relationship between the two that must be maintained. In the field of co-evolution of model-based tests, this refers to the the tests and test models evolving alongside the source models, such that the tests and test models remain correct for testing the source models. Previous work centered largely on the iterative development aspect of Model-Based Testing (MBT)[5], however further attention is needed on the prolonged maintenance of model-based tests after initial release.

Thesis Statement: Model-based test efficiency can be improved by co-evolving test models alongside system models. This can be done through studying software model evolution patterns and their effects on test models in order to apply updates directly to the tests.

II. Related Work

The work by Zech et al. [7] for the MoVE frameork, deals specifically with regression testing and the selection of the test set, however we aim to focus on the co-evolution aspect, and how exactly tests change and evolve alongside the source models. Similarly work by Farooq et al. [4] also focuses on the selection of regression tests for future testing; our work will incorporate this step but also expand to include the adaptation of existing tests. With regards to the testing work done in Simulink [1], this work looks specifically at the testing of systems, while our work aims to explore how these types of tests evolve.

III. Methodology

The work is being conducted using Matlab Simulink as our modeling language. This is primarily due to the increasing number of automotive companies (General Motors included) that are using this technology in their development. There are three phases to the proposed research: (i) Evolution Study, (ii) Algorithm Design, and (iii) Prototype Tool Development. Each phase is meant to be self-contained, with results filtering into the next.

A. Evolution Study

The first phase of the project consists of an extensive look at a number of consecutive versions of existing models and their corresponding test models (actual models and tests provided by our industrial partners) to determine how these domain models evolve. We will be extending prior work [5], now making use of MATALB Simulink Models, however we require a new set of evolutionary steps specific to this domain, which also cover a wider range of operations. The first goal will be to determine a concrete set of models that will be used for the remainder of the work. Criteria will include models with multiple versions, and test models included, and they should preferably be industrial models with real world applications.

The methodology for comparing versions of models and tests will be to make use of Simulink's built-in model model comparison, as well as our MATLAB script which makes use of the same differencing algorithm, to accurately determine differences between versions in order to create the catalog of differences. It is believed that these observed differences will fall into three broad categories: additions, modifications, and deletions (similar to those presented by Cicchetti et al. [2]). From there, it is likely that each of these differences will apply to the native elements of Simulink models, such as subsystems, connections, signals, attributes, etc., thus resulting in combinations of the two (added subsystem, deleted subsystem, modified attribute, deleted connection, etc.). Based on the catalog of changes obtained from this examination, the next part of the study will be to analyze the impacts of these model evolution steps on tests to determine how they co-evolve with the models.

B. Algorithm Design

Using the information obtained from the Evolution Study, the goal of this phase is to develop a set of algorithms for implementing the co-evolution of model-based tests. When given an existing model (M), its current test model (TM), and a number of changes applied to the model (Δ) (thus resulting in a new version of the model (M')), we will determine what changes (Δ') need to be made to the test model to ensure that the updated test model (TM') is a correct test for the newly updated model (M'), and how to apply them in the most effective way.

The aim is to develop a set of algorithms that take as input M, M' and TM, and are able to apply all necessary updates to TM to generate TM'. Recall the three categories of differences that we propose to be working with: additions, modifications, and deletions. Given an added model component, it is hypothesized that we will likely be required to add functionality to the test model, while a deletion will require the removal of functionality. This phase will solidify the rules for these types of updates, and formalize them in a set of algorithms. Furthermore, based on the portions of the test that remain unchanged, we will identify which tests do not need to be rerun during later runs, thus reducing the amount of testing required.

C. Prototype Tool Development

Using the algorithm developed in the previous phase, the third phase will be the development of a prototype tool which automates the co-evolution of model-based testing. Continuing with the Simulink use-case, the goal will be to implement our co-evolution testing framework within Simulink, writen in MATLAB code, much like our lab's tool SimNav [3]. The minimum requirement will be an interface that allows users to select two consecutive versions of a model (or set of models), along with the first version's test models; the prototype tool will then perform the updates to the tests. However, it is possible that we can make use of a Simulink version control system to manage the selection of model and test versions, such that the user need only be concerned with updating the models, and the updating of tests is done automatically in the background. Included in the prototype will also be a report to the user of the changes made to the model (reported in easy to understand terms), as well as any necessary updates to the tests, as manual interaction may be required in some cases.

In addition to the co-evolution work from this project, the goal is to create a complete testing workbench within Simulink that is capable of automatically generating test model stubs for a given system, determining differences between versions (model differencing is a well-known application [6]), and automatically co-evolving test models alongside the models (the major research contribution of this project).

IV. Evaluation

The final phase of the doctoral thesis will be an evaluation in which we test not only the correctness of the implementation, but its performance and usability as well. Evaluation will be conducted on the set of models used throughout the project.

Correctness: Evaluating the correctness of our approach will be the easiest result to obtain, as we will simply be comparing the test models generated by our tool against the existing test models. We will be looking for functional equivalence between the tests, to ensure that our generated tests test *exactly* the same behaviours as the originals.

Performance: Performance of our prototype tool will be measured on the criteria of time. Time performance will make use of two measurements to identify the amount of time required for the generation of the updated test models. First, we will look at the computation time required by the tool to generate the updated test model, and second we will look at the amount of person-time required to make use of the model and begin executing test models. These times will be compared to baseline results of manually updating existing models and executing the updated versions. The end result will be a percentage of improvement over the manual generation.

Usability: With regard to usability, we hope to provide evidence that our interface and methodology will be preferred by test engineers. To show these results, we plan to conduct usability studies, using developers from our industrial partner as subjects, in order to determine which methodology they prefer for the continued updating of test suites for their production models.

V. Limitations and Risks

The biggest risk we face in this proposed work is the availability of a substantial set of models and test models. While we intend on working with models obtained from General Motors as a result of my internship, there is a possibility the entire set will not be available. If this is the case, the open source models will serve as a complete set, however a substantial amount of models will be useful to provide additional results.

VI. Contributions

Our work will make the following contributions to the fields of model-based testing, and automotive software development:

- methodology for co-evolution of model-based tests
- catalog of evolution patterns in Simulink
- ability to identify impact of evolution on tests
- increase the efficiency of MBT test evolution process
- a test evolution workbench for industrial automotive model development

VII. Progress Made

Initial exploration of models and tests has been completed. Examination of the publicly available Automotive models, as well as those obtained from GM, has provided insight into the types of models we will be working with, all of which contributes to our first milestone of selecting models. Initial experimentation has been conducted with methods for differencing versions of Simulink models, and a Matlab script has been developed which determines if a change is an addition, modification, or deletion, and identifies the basic Simulink type of the object; this work will contribute to the construction of the evolution catalog, our second milestone, as well as the algorithm design, our third milestone. Additionally, work in the automation of test model generation has been completed. We are now capable of taking a developed library model, and automatically generating the test harnesses necessary to simulate execution using test inputs. This process will be useful in the co-evolution of tests, in that we will be able to create the updated test models with ease, allowing us to focus research efforts on the test values themselves.

References

[1] E. Bringmann and A. Kramer. Model-based testing of automotive systems. In *Proceedings of the First International Conference on Software Testing, Verification, and Validation*, ICST '08, pages 485 – 493, April 2008.

[2] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. In *Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference*, EDOC '08, pages 222 –231, September 2008.

[3] James Cordy. Submodel pattern extraction for simulink models. In *Proceedings of the 17th International Software Product Line Conference*, pages 7–10, Tokyo, Japan, August 2013.

[4] Q. Farooq, M. Iqbal, Z.I. Malik, and Matthias Riebisch. A model-based regression testing approach for evolving software systems with flexible tool support. In *17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*, pages 41–49, 2010.

[5] Eric James Rapos. Understanding the effects of model evolution through incremental test case generation for UML-RT models. Masters thesis, Queen's University, Kingston, ON, September 2012.

[6] Matthew Stephan and James R. Cordy. A survey of methods and applications of model comparison. Technical Report 2011-582, School of Computing, Queen's University, Kingston, Ontario, Canada, 2011.

[7] Philipp Zech, Michael Felderer, Philipp Kalb, and Ruth Breu. A generic platform for model-based regression testing. *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, 7609:112–126, 2012.