Model-based GUI testing using Uppaal at NOVO Nordisk

Ulrik H. Hjort[2], Jacob Illum[1], Kim G. Larsen[1], Michael A. Petersen[2], and Arne Skou[1]

[1] Department of Computer Science, Aalborg University, Denmark
[2] NOVO Nordisk A/S, Hillerød, Denmark

Abstract. This paper details a collaboration between Aalborg University and NOVO Nordisk in developing an automatic model-based test generation tool for system testing of the graphical user interface of a medical device on an embedded platform. The tool takes as input an UML Statemachine model and generates a test suite satisfying some testing criterion, such as edge or state coverage, and converts the individual test case into a scripting language that can be automatically executed against the target. The tool has significantly reduced the time required for test construction and generation, and reduced the number of test scripts while increasing the coverage.

1 Introduction

Model-based development (MBD), [5], is a recent and very promising approach to industrial software development addressing the increasingly complex world of making correct and timely software. Although originally developed for general software systems, MBD has demonstrated great potential within embedded system and in particular safety critical systems for which failures after field deployment are unacceptable, either due to the catastrophic consequences of such failures or the impossibility of updating the software after deployment. MBD techniques address these problems by working with precise mathematical models of the software system and using these models for e.g., formal verification of correctness, automatic code generation, and/or automatic test generation.

The benefits from working with models are numerous: Models are easier to communicate and more precise than textual specifications; models allow for fast prototyping as models can be simulated effortlessly; correctness of the model can be established mathematically; with automatic code generation, manual labor is minimizes assuring faster time to market and less bugs.

The world of model checking has always been a strong proponent for building models and specifications for hardware and software systems and the research field is offering many methods for establishing model correctness. Recently, algorithms and methods from model checking have been extended to other types of model analysis such as automatic controller synthesis, [4, 6], and optimal planning and scheduling, [1, 3]. The latter turns out to be particularly applicable for automatic test suite generation, as finding a smallest possible test suite satisfying some coverage criteria is, simply, a planning or scheduling problem.

In this paper, we describe the development of a tool chain for adapting model-based model checking, planning and scheduling techniques to the application of automatic test suite generation. The technology underlying the tool chain is provided by Aalborg University, who with their widely recognized model-checking tool UPPAAL, [2], has made numerous significant contribution to the field of model checking. The case for the tool chain is testing graphical user interfaces on an embedded platform and provided by the large Danish health care company NOVO Nordisk A/S. Worldwide, NOVO Nordisk employs more than 27,000 employees spread over 81 countries. NOVO Nordisk is a world leader in diabetes care, but their business areas also stretch into haemostasis management, growth hormone therapy and hormone replacement therapy.

The remainder of the paper is structured as follow: In Section 2 we provide a short introduction to UPPAAL. In Section 3 we detail the case for the collaboration, and we finally conclude with our experiences from this work.

2 The UPPAAL model checker

UPPAAL is a tool for design, simulation and model checking (formal verification) of real-time systems modeled as networks of timed automata extended with discrete datatypes and user-defined functions. Based on more than a decade of research, UPPAAL provides very efficient algorithms and symbolic datastructures for analysing such models.

Since the release of UPPAAL in the mid-nineties, several variants have emerged, realizing the strength of both timed automata as a modeling language for real-time behavior and the efficiency of UPPAAL's symbolic model checking engine. Such variants include[3]:

– UPPAAL TIGA: Automatic synthesis of controllers for real-time systems. The modeler provides a model of the environment together with a model of all possible behavior of the controller. Using the model and providing a control objective, UPPAAL TIGA is able to automatically synthesis a controller that guarantees the objective.
– UPPAAL CORA: Optimization engine for resource-constrained problems such as planning and scheduling problems. Given a resource-constrained problem with many potential solutions, UPPAAL CORA is able to find the optimal or swiftly generate near-optimal solutions to the problem.
– UPPAAL TRON: Engine for *online* testing of real-time systems. Online testing differs from regular testing in the sense that the model of the systems is executed in parallel to actual system implementation. The purpose of the executing model is to stimulate the implementation with legal inputs and observe the behavior to establish whether the implementation adheres to the model.

[3] All the tools are available from `http://www.uppaal.com`

– UPPAAL PRO: Analysis of real-time systems showing probabilistic behavior. UPPAAL PRO takes as input a model of a real-time system with uncertainty and is able compute the probability of the model satisfying some criterion.
– TIMES: Schedulability analysis of real-time systems. TIMES is able to establish schedulability of task and resource systems where the tasks are modeled as timed automata.

3 The Case Study: GUI Testing

The purpose of this collaborative work between Aalborg University and NOVO Nordisk has been to develop a test generation tool for system testing of graphical user interfaces using the UPPAAL model checking tool.

The company is developing the hardware and software of an embedded device for medical purposes. This device has a graphical user interface to receive instructions from and provide feedback to the user. The software department at NOVO Nordisk has the assignment of system testing the GUI to determine whether all interactions work appropriately and that the expected information is displayed on the screen. The specification of the behavior is traditionally made in Microsoft Visio to provide a graphical representation that can be used in the review process as well as for implementation.

The process for the system testers is to look at the Visio drawings and manually generate a set of test cases that cover the behavior of the GUI and validate that the output is correct. These tests need to be reviewed and accepted, before they are finally converted to a scripting language that can automatically be executed against the platform. This process is depicted on the left of Figure 1.

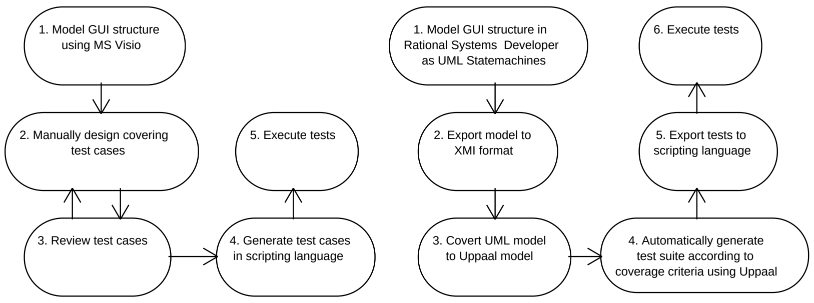
1. Model GUI structure using MS Visio
2. Manually design covering test cases
3. Review test cases
4. Generate test cases in scripting language
5. Execute tests
1. Model GUI structure in Rational Systems Developer as UML Statemachines
2. Export model to XMI format
3. Covert UML model to Uppaal model
4. Automatically generate test suite according to coverage criteria using Uppaal
5. Export tests to scripting language
6. Execute tests

Fig. 1. Old (left) and new (right) testing process.

This approach has a number of drawbacks, namely, that 1) manual creation of test cases is tedious, 2) establishing whether the test cases do in fact cover the model is difficult, 3) changes to the model require the entire process to be repeated.

By introducing model-based testing in this process, we can alleviate these drawbacks by removing the manual task of test generation and test review, knowing that the output of the automatic process is mathematically guaranteed to cover the model. And finally, since the technology is fully automated, changes to the model are reflected in the test cases by the push of a button.

To ease the transition to model-based GUI specifications, we have chosen UML Statemachines as an input model since 1) UML is an establish standard familiar to most developers and thus requires minimal re-education, 2) NOVO Nordisk has the software infrastructure in place to support the building of UML models, 3) the UML Statemachine notion is very similar to the current Visio models thus maintaining the current validation process.

To accommodate this choice, it has been necessary to adapt Uppaal to accept UML Statemachine models. This has been accomplished by using the XML-like UML exchange format called XMI, which is converted into a Uppaal model. The engines of Uppaal and Uppaal Cora are then applied to the models and used to generate a test suite with either edge or state coverage. The resulting test cases are, finally, converted into the scripting language that can be executed on the target. This new process is depicted to the right of Figure 1.

4 Conclusion

Using the automated testing tool has reduced the time used on test construction from upwards of 30 days to 3 days spent modelling and then a few minutes on actual test generation. The benefits extend into the specification process, as the model structure is used for specifications and later refined with action code to allow for automatic test generation. This removes the process of keeping specifications and test models consistent, an otherwise tedious and error-prone process. Furthermore, the testing tool has decreased the number of test case while at the same time increasing and guaranteeing full model coverage. The automatically generated test scripts have uncovered a number of bugs in the software, even "difficult" bugs that can be hard to detect, since the test generation process makes no assumptions about how the system should be used; something testers have a tendency to do.

The company has experienced that creating usable models for test generation requires time, however, once the model has been generated making changes, extensions, and doing maintenance is easy. Finally, the fact that specification changes are immediately reflected in the test suite has proved extremely helpful.

In conclusion, the collaboration has been very successful and beneficial to both company and university.

References

1. G. Behrmann, E. Brinksma, M. Hendriks, and A. Mader. Scheduling lacquer production by reachability analysis – a case study. In *Workshop on Parallel and Distributed Real-Time Systems 2005*, pages 140–. IEEE Computer Society, 2005.
2. Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *LNCS*, pages 200–236. Springer, 2004.

3. Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Optimal scheduling using priced timed automata. *ACM SIGMETRICS Perform. Eval. Rev.*, 32(4):34–40, 2005.
4. Franck Cassez, Jan J. Jessen, Kim G. Larsen, Jean-François Raskin, and Pierre-Alain Reynier. Automatic synthesis of robust and optimal controllers - an industrial case study. In *HSCC*, volume 5469 of *LNCS*, pages 90–104. Springer, 2009.
5. David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing.* John Wiley & Sons, 2003.
6. Jan Jakob Jessen, Jacob Illum Rasmussen, Kim Guldstrand Larsen, and Alexandre David. Guided controller synthesis for climate controller using uppaal tiga. In *FORMATS*, volume 4763 of *LNCS*, pages 227–240. Springer, 2007.