Model-Based Testing for Avionics Systems*

Jörg Brauer and Uwe Schulze

Verified Systems International GmbH, Bremen, Germany
lastname@verified.de

Abstract. Model-based testing is considered state-of-the-art in verification and validation of safety-critical systems. This paper discusses some experiences of applying the model-based testing tool RTT-MBT for the evacuation function of an aircraft cabin controller. One challenge of this project was the parametric design of the software, which allows to tailor the software to a certain aircraft configuration via application parameters. Further challenges consisted of connecting hardware signals of the system under test to abstract model variables, and handling incremental test model development during an ongoing test campaign. We discuss solutions that we developed to successfully conduct this test campaign.

1 Introduction

Over the past two decades, we have observed a wide-spread adoption of model-based development techniques in industry. In parallel, the scientific community has provided promising concepts and powerful solutions for the area of model-based testing, in particular with respect to the automation of the test case generation process. SMT solvers, to name just one example, have advanced to the state where they can easily solve problems involving thousands of variables, and thus support the automated generation of test data for large test models. This technological progress is a prerequisite for effective application of model-based testing in industry. In practise, however, there is often some mismatch between the scientific progress on the one hand and the industrial requirements on the other hand. In particular, we have observed that the successful and not so successful model-based testing projects often just differ in the supportive features offered by the model-based testing frameworks.

This paper is about effective model-based testing for the avionics domain, and the peculiarities that need to be taken care of, which in the end often make the difference between success and failure of a project. For the cabin control system of several Airbus aircrafts, our company has provided several hardware-in-the-loop (HIL) test benches which traditionally execute hand-written tests. More recently, the existing test infrastructure was extended by tests generated from test models using RTT-MBT [3], which is based on simulation and SMT solving [1, 2]. Model-based testing for the cabin control system poses some interesting challenges that need to be taken care of:

* The work presented in this contribution has been partially funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) in the context of project STEVE, grant application 20Y1301P.

- The runtime behavior of the system is highly configurable via application parameters that need to be selected by the test case generator with the test data. Trying out all admissible combinations of parameters is infeasible.
- The system uses numerous different I/O interfaces.
- The test case generator needs assistance for the creation of meaningful, descriptive test cases which not just cover the requirements using some arbitrary inputs, but also trigger the standard use cases of the system.

2 Core Challenges

The emergency evacuation function of the cabin controller has some interesting properties. First and foremost, the software is highly configurable at runtime, using a configuration file, which allows to adapt the software to the layout of and the devices installed in an aircraft. The logic implemented by the evacuation function itself strongly depends on these parameters. This characteristic has been investigated in recent years under the term product line testing. For instance, contemporary aircrafts have a set of attendant panels, the number of which is configurable. Attendant panels can be configured to indicate an evacuation situation differently: either a flashing light or a steady light can be used for this purpose. A test which examines the behavior of the evacuation function with respect to the attendant panels thus needs to be run with a configuration file that matches the prerequisites for the test. If the test objective is to examine whether the flashing behaviour is correctly implemented, a configuration needs to be created which (1) installs at least one attendant panel and (2) assigns the flashing mode to the panel indication lights used by the evacuation function. For testing the steady indication, another configuration and another test is needed.

2.1 Application Parameters

In our target system, the parameter configuration consists of roughly 100 C-structures. The parameters defined in the structures can themselves be structured and can define dependencies on other system parameters. This provides a very powerful way to customize the system, but also makes it virtually impossible to take all possible parameter settings into account when defining the test model. The complexity of the system parameters is also a serious challenge in manual test campaigns, because manually calculating suitable system parameters is complicated and error-prone. Normally only a small subset of the parameters is directly related to the functionality addressed by a test model. In case of the evacuation function, only 16 parameters defined in two of the structures directly affect the behavior of the function. These parameters must be adjusted to enable certain parts of the system behavior under consideration.

Other parameters are still relevant, but do not need to be changed to reach certain test goals. They do, however, define constraints to the system, and these must be taken into account when designing the test model. For these parameters, constants were defined in the model to represent the selected settings, and a

fixed definition for this part of the parameter state space was used for all test generations of the test suite. The configuration parameters cannot be changed at runtime without restarting the system. This information is reflected in a model via a UML stereotype called "parameter", which is assigned to the respective model variables. The transition relation used by the test generator is augmented with constraints to ensure that the parameters do not change during a single test. This approach results in a significant improvement compared to manual test development with manual calculation of the configuration parameters.

2.2 Interface Modules

Despite the improvements on SMT solving, generating test cases may still be computationally infeasible on the concrete semantics of an application, which naturally leads to the question of abstraction. Modelling the application behavior on the granularity of hardware interfaces leads to state explosion, which entails that some kind of abstraction layer has to be introduced, which maps model inputs to concrete hardware signals and vice versa. For instance, it is not uncommon that some model input is represented by the conjunction of multiple concrete hardware signals. The mapping between model variables and hardware signals is implemented manually in a layer that resides between the device drivers and the test driver, and requires significant expertise of the test engineers.

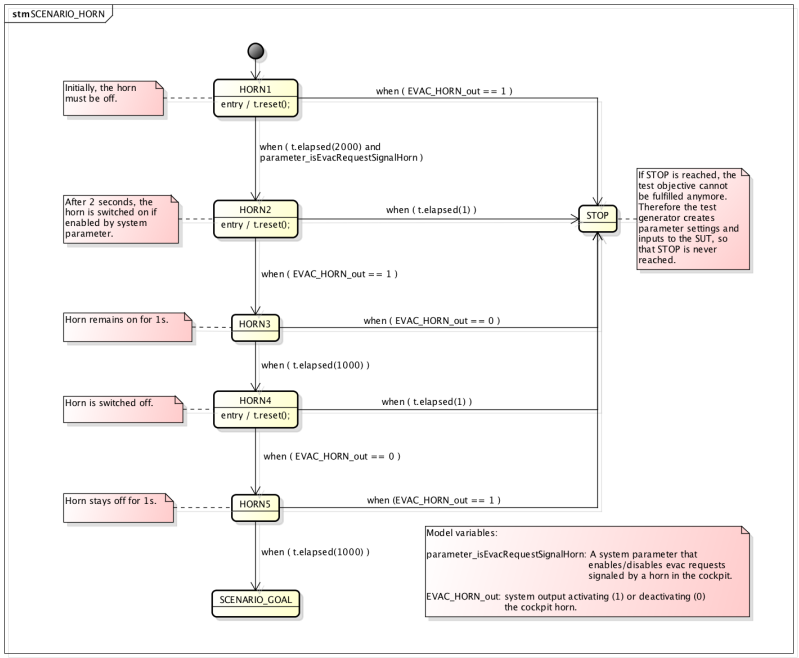

Fig. 1. A test scenario characterizes a family of stimulations

2.3 Test Scenarios

Model-based test generation often produces tests that are semantically correct, but not very realistic, which may be an issue if the tests are used for certification purposes. RTT-MBT provides a mechanism to restrict the test generator by defining constraints for the test environment. Defining a complete test environment specification suitable for all tests generated from a model, however, can require a lot of effort or even be practically infeasible. Simple test goals with RTT-MBT can be defined as states or transitions to be covered, but complex test goals must be specified in LTL, which is not intuitive to most users. Both problems are addressed using additional state machines in the test models that define so-called *test scenarios*. A test scenario defines a complex test goal trough a sequence of sub-goals and excludes undesired behavior. An example of such a test scenario is given in Fig. 1. This way, a test scenario combines the partial definition of test environment restrictions that are tailored to a complex test goal together with the intuitive step by step definition of the goal itself. Note that the test generator is still used to calculate the concrete test data, but is restricted through the constraints imposed by the additional scenario state machine.

3 Conclusion

We have described an approach to model-based testing of highly configurable avionics control systems. To name one example, the approach has been used for the verification of sub-systems of the Airbus A350 aircraft. Our strategy relies on test models that describe the system behavior depending on application parameters, which are integrated as model constraints. An interesting aspect with respect to economic viability of model-based testing is the efficiency of test model development. Significant reductions in efforts can be achieved with regression campaigns, but the initial investment for the transition should not be underestimated, which may be explained by the fact that model development requires a different skill set than traditional test development. There are, of course, open issues to be addressed in the future. For instance, the question of sufficient configuration coverage needs to be answered. To this end, we currently adapt an input equivalence class testing strategy with guaranteed fault detection properties.

References

1. Lapschies, F.: SONOLAR homepage (Jun 2014), http://www.informatik.uni-bremen.de/agbs/florian/sonolar/
2. Peleska, J., Vorobev, E., Lapschies, F.: Automated test case generation with SMT-solving and abstract interpretation. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) Nasa Formal Methods, Third International Symposium, NFM 2011. LNCS, vol. 6617, pp. 298–312. Springer, Pasadena, CA, USA (April 2011)
3. Verified Systems International GmbH: RTT-MBT: Model-Based Testing, https://www.verified.de/products/model-based-testing