Integrating Model-Based and Constraint-Based Testing Using SpecExplorer

Vivek Vishal, Mehmet Kovacioglu, and Rachid Kherazi
Philips Healthcare
Best, The Netherlands
Email: {vivek.vishal,rachid.kherazi, mehmet.kovacioglu}@philips.com

Mohammad Reza Mousavi
Eindhoven University of Technology
Eindhoven, The Netherlands
Email: m.r.mousavi@tue.nl

***Abstract*—We report on our experience with model-based testing using SpecExplorer within the Flat X-Ray Detection (FXD) Department of Philips Healthcare. Our initial experiments showed a practical obstacle in combining traditional functional testing techniques with model-based testing using SpecExplorer. We overcome this obstacle by specifying the constraints on our data domain in a spreadsheet and interfacing SpecExplorer with a constraint solver in order to generate concrete test data for the behavioral specifications. We report on some empirical results obtained from our experiments.**

***Keywords*-Model-Based Testing, Decision-Table-Based Testing, Constraint-Based Testing, SpecExplorer;**

I. Introduction

A. Problem domain

Flat X-Ray Detection (FXD) group of Philips Healthcare is responsible for the subsystem which generates, detects and translates X-Rays into medical images. The subsystem consist of a controller and a flat X-Ray detector (a digital photo-sensitive plate with layers that convert X-Ray to light). There are many different medical applications that use different configurations of this subsystem. The customers are system developers in the medical domain who integrate the subsystem to manufacture a complete X-Ray system. Stringent requirements enforced by regulatory authorities (such as the Food and Drug Administration, FDA, in the United States) in the medical domain include providing consistent tracking of requirements to implementation and tests. Hence, the present work is carried out in a strictly regulated domain, which requires a clear, traceable and effective test and quality assurance process.

The current test environment consists of an infrastructure that can simulate (parts of) the hardware used by the subsystem and a set of test-scripts based on a well known unit-test framework (NUnit) to perform semi-automatic tests. This environment is also used to generate clear and consistent test logging and tracing. Although the current setup of the test environment has provided a flexible solution for semi-automatic testing of various configurations and setups, there is still ample room for increasing the effectiveness and the efficiency of the tests (with respect to the effort spent on different phases of testing, as well as various notions of coverage versus testing time). Model-Based Testing (MBT) is considered as a serious candidate for this purpose, because it offers a promising approach in automating test generation. Using MBT, testers can update the model and rapidly regenerate a new test suite, avoiding tedious and error-prone editing of the suite of hand-crafted test. It is particularly effective in the current System Under Test (SUT) as its specification changes frequently and various products may have to be tested with slightly varying models or configurations.

B. Problem definition

We perform a controlled experiment in applying MBT to our domain and compare its efficiency and effectiveness with the traditional semi-automatic approach. To this end, we use the SpecExplorer tool from Microsoft Corporation and build the infrastructure to interface it directly with our SUT. A major complicating factor (as it turned out during the experiment) is the inter-dependencies between the behavioral model and the data model used for parameter values, which determine the amount of X-Ray generated. Since exposure to X-Ray is harmful to human beings, it has to be made sure that they are well within their prescribed range. Testing this aspect involves choosing data values for different parameters of the system. As the SUT has a large number of parameters, testing all combinations of parameters soon becomes infeasible. Also, the standard data selection techniques provided by SpecExplorer fail due to combinatorial explosion in our setting. Moreover, the parameters are not independent and selection of one value of a parameter determines the valid range of other parameters, while the built-in techniques from SpecExplorer are suitable for independent data parameters. Hence, in addition to applying MBT to this domain, we aim at firstly, developing a framework that can automatically combine the behavioral model with an appropriate data model, and secondly, applying it to our setting and measuring its efficiency and effectiveness.

C. Summary of the results

Using SpecExplorer, we have successfully applied MBT to our domain (see [10] for another experience report); we have interfaced our testing framework with a constraint solver in order to efficiently select test data from our data model. Our empirical results show that the final framework (henceforth called MBT+CBT, for Model-Based Testing + Constraint-Based Testing) saves, respectively, about 80% and about 20% of the total test effort compared to the semi-automatic and the plain MBT approach (without constraint solving), while the final framework shows significant improvements with respect to different coverage metrics compared with the other two techniques.

D. *Related work*

Constraint-based solving is a well-studied research area going back to seminal work in the early 90's such as those of [6], [7]. The general idea is to feed a data model to a constraint solver and use the constraint solver to generate data values automatically. For example, in [6], this ideas is developed in the context of mutation testing, where necessary data to kill a mutant is expressed as an algebraic expression, which is then solved by a constraint solver in order to generate test cases. In [7] the idea of constraint-based testing is developed for a formal testing framework based on state-based specifications (particularly, in the Vienna Development Methods, VDM).

Using behavioral models in MBT usually calls for abstract models (in terms of finite-state machines, or labelled transition systems) which focus on the input-output behavior of the SUT and hide the details about its state variables and their exact valuations (cf. [5], [12]). Although this abstraction level is useful in building compact representations of system behavior, in most practical applications data and data valuations have to be accommodated in such models. Theoretical extensions of behavioral models have been developed and used in the context of MBT in the past few years, see, for example, [2], [3], [9], [11]. Also the SpecExplorer framework provides some support for integrating data with behavioral modeling, but it only provides support for a few standard data selection mechanisms. In our experiments, none of the provided techniques were applicable due to the combinatorial explosion of the combination of data parameters. (This has been observed in earlier experiments with other tools, see, for example [2], which applies other techniques than what we explored in this paper for restricting the test data selection.) Hence, we were forced to design our own framework for combining data models with MBT using SpecExplorer. We could not find any similar framework in the context of SpecExplorer in the literature.

E. *Structure of the paper*

In Section II, some more information about the problem domain and the SUT is provided. In Section III, we report on our approach to applying MBT to our problem domain. Subsequently, in Section IV, we describe how we have applied CBT on our data model and integrated it with MBT. Empirical results regarding different test frameworks are presented in Section V. Concluding remarks as well as some avenues for future research are presented in Section VI.

II. Background

X-Rays systems are used for medical imaging in various cardiovascular-, and neurological procedures. The system under test is the Flat Detector Subsystem (FD) also known as the Image Detection Subsystem (IDS) responsible for generation, detection and translation of X-Rays to images.

The IDS consists of following components:

- A flat detector for detecting X-Rays,
- An anti-scatter grid for ensuring that X-Rays fall perpendicularly on the flat detector,
- A temperature control unit maintaining the temperature of the flat detector, and
- A flat detector controller for a number of control activities, including: dose measurement and control, image preprocessing and transformation, subsystem timing generation and temperature control.

We are only concerned with testing the flat detector controller, which will be called the SUT in the remainder of the paper. An example of the IDS and the SUT is depicted in Figure 1.

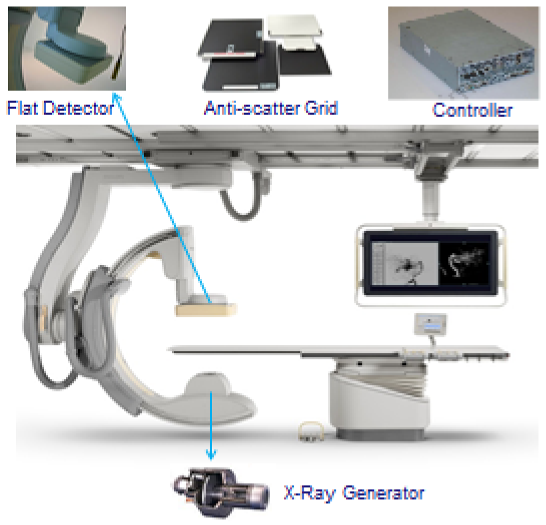

Figure 1. System and Flat Detector

In the traditional setting, the SUT is wrapped around by a test environment known as Bellephoron (Bello). Bello contains everything (Hardware/Software) that is needed to test the SUT. It is also responsible for creating an environment which enables the test script to communicate with the different interfaces of the SUT.

III. Applying MBT

Three main activities have to be carried out in order to apply MBT to our SUT:

1) constructing models,

2) building an adaptor, and
3) dealing with data parameters.

We briefly describe each of the activities in the remainder of this section. The last activity led to the observation that the current framework, as supported by SpecExplorer, is not sufficient for our purpose, which is the motivation for the extension reported in the next section.

A. *Constructing models*

A major problem in MBT is constructing models that are correct and correct with respect to (and traceable to) the requirement specification. This is a non-trivial task as the requirement specification is informal and naturally ambiguous. To develop the model of the subsystem, we follow an iterative and incremental approach, as depicted in Figure 2. We used rounds of reviews both for the graphical representations of our models as well as for the generated test cases and their outcomes. Requirement engineers provided useful insight, which led to corrections of the initial models.

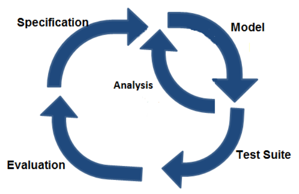

Figure 2. Iterative & incremental development model

We initially considered a basic set of requirements in the specification, design a model to capture those requirements, generate a test suite from the model and evaluate the results after executing the test suite. This cycle continues until all the requirements have been covered correctly, i.e., all requirements are reflected in the model. In addition to interaction with the requirement engineer, we also had access to a detailed Product Requirement Specification (PRS) document, which helped us in checking and validating our models.

B. *Implementing an adapter*

The MBT infrastructure is shown in Figure 3. The infrastructure for MBT requires the creation of the test adapter, which is a thin software layer that sits between the model and the SUT.

Finally test cases covering about 10000 transitions (resulting from a different sequence of actions) were generated from the model within seconds. The execution of those test cases took less than 10 minutes.

C. *Dealing with data*

Data is an important aspect of testing for our SUT. In the SUT, there are more than 20 parameters, which

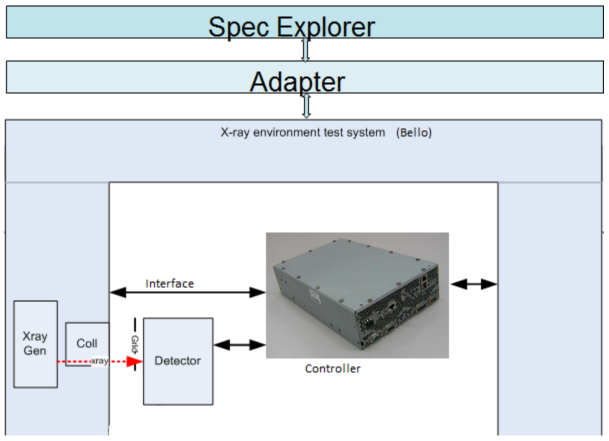

Figure 3. MBT Infrastructure

comprise a particular configuration of the subsystem. Most of the parameters are independent of each other and can be tested in isolation. However, there are parameters that are interdependent on each other, e.g., through an expression of the following form:

$$MinValue < A * (B/100) * C \leq MaxValue,$$

where *MinValue* and *MaxValue* are determined based on the values of other parameters. There are about 200 such *MinValue* and *MaxValue* pairs, based on different valuations of other parameters.

SpecExplorer has various built-in techniques for data combination, such as pairwise, n-wise, interaction, seeded, and isolated. However, it does not provide support for user-defined combination strategies. The built-in combination strategies of *SpecExplorer* do not provide any means for reflecting dependencies among parameters. Nevertheless, we tried the pairwise data selection technique from *SpecExplorer* to experiment with our developed framework. The results of our experiments (reported in Section V) did show improvement over the existing infrastructure in terms of test effort but left some of the coverage measures unchanged. An analysis of this results led to the conclusions that an extension of the current infrastructure with genuine support for a data model (reflecting dependencies among parameters) is worthwhile. We report on this extension in the next section.

IV. Integrating with CBT

As explained in the previous section, our attempt to apply MBT to our application domain revealed essential dependencies between the behavioral model and its data parameters. We used a traditional data modeling technique, i.e., *decision tables* [8], to specify our data model. The choice is motivated by their ease of use and similarity to the spreadsheets used for requirement specification in our domain. A decision table is a table consisting of conditions,

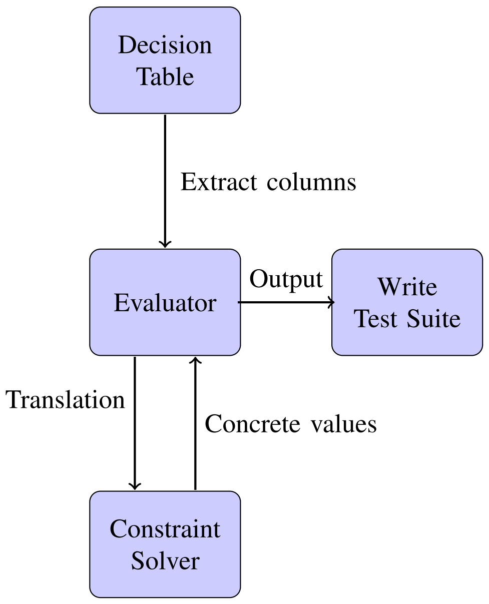

Figure 4. Architecture

rules and expected outcomes. Part of a decision table for the following constraint (discussed in Section III-C) is given in Table I.

$$MinValue < A * (B/100) * C \leq MaxValue,$$

Note that $MinValue$ and $MaxValue$ in our actual model depend on a number of other parameters and hence are represented by $MinValue(D, E)$ and $MaxValue(D, E)$, respectively, to represent these dependencies.

c1: MinValue(D,E) < A * (B / 100) * C	T	T	F	F	...
c2: A * (B / 100) * C ≤ MaxValue(D,E)	T	F	T	F	...
a1:Value out of range		×	×	×	
a2:Valid dose		×			

Table I
DECISION TABLE FOR A TYPICAL CONSTRAINT

In order to interface the decision tables with our MBT framework we need to generate concrete data values and feed them into our behavioral models. We built an automatic translator from the spreadsheet format to the input language of a constraint solver. For our experiments, we used a simple constraint solver in C# (called ZogSolver: http://zogsolver.sourceforge.net), but any other constraint- or SMT-solver can be used for this purpose.

V. RESULTS

In this section, we discuss the results obtained during our experiment with Model-Based Testing of the controller component in the Image Detection Subsystem. We compare the result of the MBT experiment with the current practice

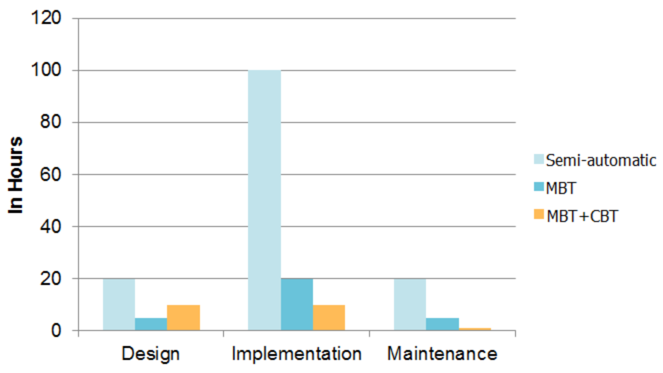

Figure 5. Comparison: Effort

of semi-automated testing. We gathered empirical metrics in two different categories:

- Effort involved
- Coverage achieved

A. *Effort*

We mainly measured the effort involved in designing (models required to generate) the test-suite and implementing them, as well as its execution and adaptation (maintenance). We did not take the effort involved in implementing the test infrastructure, as the traditional infrastructure has evolved throughout years of testing practice and is difficult to compare with the relatively lightweight infrastructure we built for the MBT and its combination with CBT. Figure 5 shows a comparison of the effort involved in the three approaches.

Test design effort reported in Figure 5 refers to the logical design of test suits, which involves extracting information from the PRS and phrasing it in terms of scenarios and data models. Implementation concerns extracting concrete test-scripts (in case of the semi-automatic method), and behavioral models (in case of MBT and MBT+CBT). Note that in the case of MBT+CBT some effort is spent in designing a data model at the design phase and hence concrete data values need not be implemented in the implementation model, that is why MBT+CBT shows a significant improvement in the implementation effort over the MBT approach. Maintenance activity refers to amount of effort required to reuse the implementation when a new configuration of the subsystem is available. In the current semi-automated testing the data values for the scripts have to be manually modified. In MBT, maintenance requires a few lines of change in the model code and some (considerable) changes in the data values. Maintenance in MBT+CBT just requires a few lines of change in the model code and possibly some modification of the symbolic constraints.

1) Coverage: In order to measure the effectiveness of our approach, we measure the coverage of our test-suites based

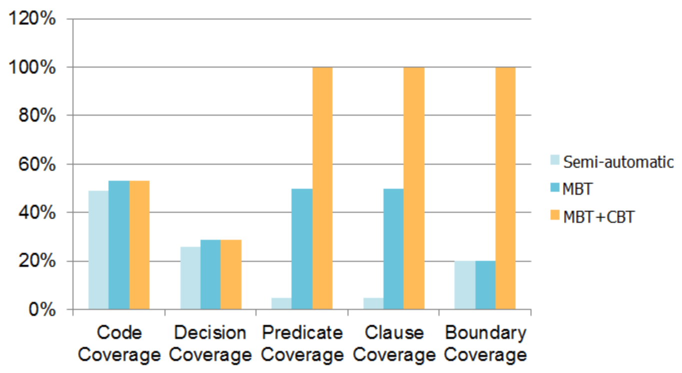

Figure 6. Comparison: Coverage

on both implementation (code) and model coverage metrics. To this end, we used and measured the following metrics:

- Code (statement) coverage is the metric that denotes the percentage of statements exercised by the test suite.
- Decision coverage is a metric that denotes the branches that are exercised by the test suite.
- Predicate coverage is a model coverage metric which measures the percentage of the boolean expressions (in the data model) evaluated both to *true* and *false*.
- Clause coverage is a model coverage which measure the percentage of atomic boolean expressions (not containing any logical operators) that are evaluated both to *true* and *false*.
- Boundary coverage is a model coverage metric, which measures the percentage of the boundaries that have been tested.

In order to compute the code coverage and the decision coverage, we instrumented our SUT with a code coverage analyzer tool Bullseye (http://www.bullseye.com/).

For the semi-automatic approach we had to calculate the model coverage metrics manually by inspecting the data values used in the test scripts. In the current automated tests or MBT, boundaries for only independent parameters are tested. No test is conducted in order to verify the correct implementation of boundaries at *MinValue* and *MaxValue*. When MBT is used along with the data combination tool, all boundaries for each *MinValue* and *MaxValue* pair is tested. The comparision in form of chart is shown in Figure 6

2) Other Metrics: A number of other metrics for comparison are also used measured. The metrics along with their comparison are given in Table II.

- **Testing technique used:** Automated tests combine the parameters on the basis of Boundary Value Analysis (BVA) and Elementary Comparison Test (ECT). However, interdependent boundaries are not tested and ECT is applied only for few *MinValue* and *MaxValue* pair as it has to be done manually. While conducting MBT we have used the inbuilt combination strategies (pairwise combination) of *SpecExplorer*. MBT + CBT uses BVA along with Decision Table (DT) and random testing. The boundaries of the dependencies are tested.
- **Number of test cases generated:** Automated tests generate less number of test cases than MBT or MBT+CBT. MBT generates about 13000 test cases as it combines all the parameter value provided to it. However, most of the combinations result in an invalid combination. MBT + tool generates about 1000 test cases, half of which results in valid combinations (tests are expected to pass).
- **Test execution time:** Test execution time is measured by the difference between the starting time of a test and the time at which it completes the execution. Both start time and end time of a test are retrieved from the logging information available in the test infrastructure.
- **Perceived effectiveness:** The effectiveness of the semi-automated test is considered low as it generates less number of test cases with high effort. MBT generates a lot of test cases, however most of them are invalid. The effort involved is also medium as one has to give values for the parameters manually from the requirement specification. The effectiveness of MBT + CBT is considered high as a reasonable number of "smart" test cases are generated with very low effort.
- **Additional bugs found:** Using MBT or MBT along with CBT, two additional bugs were detected that were overlooked by the current semi-automated tests.

Metrics	Automated test	MBT	MBT + CBT
Testing technique	BVA + ECT	Pairwise	DT+CBT
Test cases generated	∼100	∼13000	∼1000
Test execution time	1 minute	43 minutes	7 minutes
Perceived effectiveness	Low	Medium	High
Additional bugs found	-	2	2

Table II
COMPARISON

VI. CONCLUSIONS

We applied Model-Based Testing using Spec Exlorer to the Image Detection Subsystem responsible for generation, detection and translation of X-rays to images. We used an existing test infrastructure to communicate with the SUT and developed an adapter, that acts as a wrapper around the whole infrastructure. We designed a model of the subsystem using its requirement specification in the input language of SpecExplorer. A number of test cases were generated from the model automatically. The execution of these test cases were done in few minutes.

Model-Based Testing has obvious advantages over current semi-automatic testing techniques with respect to the spent effort and the coverage achieved. One of the advantages of MBT is its ability to retrace long walks through of the SUT using the test cases automatically derived from the model. The main cost involved is learning MBT and building models, which are investments requiring a certain abstract view of the specification and the system under test.

One of the critical issues we encountered while applying MBT concerned data parameters and their interdependencies. SpecExplorer provides some built-in data selection mechanisms which turned out to be insufficient for our purpose. Hence, we developed our own tool that interfaces SpecExplorer with a constraint solver in order to generate concrete test data for the behavioral model.

Our framework is currently being used in the practice of testing at the FXD Department at Philips Healthcare. We still need to address a number of open issues in order to improve the effectiveness of our MBT practice. Proper visualization of the generated test data and developing a method for establishing correctness of our models are also important issue, which needs to be addressed.

Acknowledgments. The authors would like to thank the members and the management of the FXD department for their support. The authors would also like to thank the members of the SpecExplorer Forum for their prompt feedback.

References

[1] P. Ammann, and J. Offutt. Introduction to Software Testing. Cambridge University Press, 2008.

[2] T. Bauer, R. Eschbach, M. Großl, T. Hussain, D. Streitferdt, and F. Kantz. Combining Combinatorial and Model-based Test Approaches for Highly Configurable Safety-critical Systems. CTIT Workshop Series WP09-08, pages 9–22, CTIT, 2009.

[3] A. Beer and S. Mohacsi. Efficient Test Data Generation for Variables with Complex Dependencies. In ICST'08, 1st Int. Conf. on Software Testing, Verification and Validation, pages 3-11, IEEE, 2008.

[4] A. Bertolino, G. De Angelis, L. Frantzen, and A. Polini. Model-Based Generation of Testbeds for Web Services. In TESTCOM/FATES'08, Int. Conf., volume 5047 of LNCS, pages 266-282. Springer, 2008.

[5] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner. (Eds.) Model-Based Testing of Reactive Systems. volume 3472 of LNCS, Springer, 2005.

[6] R.A. DeMillo and A.J. Offutt. Constraint-Based Automatic Test Data Generation. IEEE Transactions on Software Engineering 17(9):900-910, 1991.

[7] J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In FME93, Int. Conf. on Industrial Strength Formal Methods, volume 670 of LNCS, pages 268 - 284, Springer, 1993.

[8] J. B. Goodenough, and S. L. Gerhart. Toward a theory of test data selection. ACM SIGPLAN Notices, 10(6):493–510, 1975.

[9] A. Gotlieb. Euclide: A constraint-based testing platform for critical C programs. In ICST'09, 2nd Int. Conf. on Software Testing, Validation and Verification (ICST'09), pages 151-160, IEEE, 2009.

[10] W. Grieskamp, N. Kicillof, K. Stobie, V. A. Braberman. Model-based quality assurance of protocol documentation: tools and methodology. Softw. Test., Verif. Reliab. 21(1):55-71, 2011.

[11] S. Mohacsi and J. Wallner. A Hybrid Approach for Model-Based Random Testing, In VALID'2010, 2nd Int. Conf. Advances in System Testing and Validation Lifecycle, pages 10-15, IEEE, 2010.

[12] J. Tretmans. Model Based Testing with Labelled Transition Systems, In Formal Methods and Testing 2008, volume 4949 of LNCS, pages 1–38, Springer, 2008.