THESIS ON INFORMATICS AND SYSTEM ENGINEERING C49

Model-Based Testing of Reactive Systems

ANDRES KULL

TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Control

Dissertation was accepted for the defence of the degree of Doctor of Philosophy in Engineering on October 15, 2009.

Supervisors: Professor Leo Mõtus
Department of Computer Control
Tallinn University of Technology

Professor Jüri Vain
Department of Computer Sciences
Tallinn University of Technology

Opponents: Researcher Dr. Margus Veanes
Microsoft Research, USA

Professor Johan Lilius
Embedded Systems Laboratory
Åbo Akademi University, Turku, Finland

Defence of the thesis: December 15, 2009

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for any academic degree.

/Andres Kull/

ISSN 1406-4731
ISBN 978-9985-59-943-3

INFORMAATIKA JA SÜSTEEMITEHNIKA C49

Reaktiivsete süsteemide mudelipõhine testimine

ANDRES KULL

Contents

Abstract 8
Lühikokkuvõte 10
Acknowledgements 12
Abbreviations 13
PART I: OVERVIEW 15
1 INTRODUCTION 17
1.1 Organisation of thesis 17
1.2 Field of research 17
1.3 Motivation 19
1.4 Objectives of thesis 20
2 SCOPE AND RELATED WORK 21
2.1 Characteristics of reactive systems 21
2.2 Taxonomy of model-based testing 22
2.3 Scope summary and related work 27
3 GENERATING TESTS FROM DETERMINISTIC MODELS 31
3.1 Generating tests from EFSM using guided model-checking 31
3.2 Generating tests from EFSM incrementally using model checking 33
4 GENERATING TESTS FROM NONDETERMINISTIC MODELS 36
4.1 Synthesis of test purpose-directed reactive planning tester 36
4.2 Case study-based performance evaluation of reactive planning tester 38
5 REQUIREMENT-DRIVEN TESTING 40
PART II: PRACTICAL RESULTS AND CASE STUDIES 43
6 MOTES – TEST GENERATOR 45
6.1 Overview 45
6.2 Description of light switch example 46
6.3 Creating the IUT model 49
6.4 Preparing test data 51
6.5 Importing the model and test data into MOTES 52
6.6 Defining test purpose (coverage/goal) 53
6.7 Choosing the test generation engine 55
6.8 Executing the test generation engine 57

7 CASE STUDIES 58
7.1 Introduction 58
7.2 Testing of Sofia-SIP stack 59
7.2.1 Overview of IUT 59
7.2.2 Objectives of case study 60
7.2.3 System adapter 61
7.2.4 Modelling the IUT 61
7.2.5 Test generation and execution 64
7.2.6 Test coverage analysis 64
7.2.7 Objectives evaluation 66
7.3 Testing controllers of street lighting system 67
7.3.1 Overview of IUT 67
7.3.2 Objectives of case study 68
7.3.3 System adapter 70
7.3.4 Modelling the IUT 70
7.3.5 Test generation and execution 73
7.3.6 Comparing test generation performance against manual test scripting 74
7.3.7 Objectives evaluation 75
7.5 Summary of case studies 77
8 CONCLUSIONS 80
REFERENCES 82
PART III: RESEARCH PAPERS 87
PAPER 1: Ernits, J. P., Kull, A., Raiend, K., Vain, J., Generating tests from EFSM models using guided model checking and iterated search refinement. *In: Formal Approaches to Software Testing and Runtime Verification: First Combined International Workshops FATES 2006 and RV 2006, Seattle, WA, USA, August 15-16, 2006*, Revised Selected Papers: Havelund, K., et al. Berlin: Springer, 2006, (Lecture Notes in Computer Science; 4262), 85-99. . 89
PAPER 2: Ernits, J. P., Kull, A., Raiend, K., Vain, J., Generating TTCN-3 test cases from EFSM models of reactive software using model checking. *In: Informatik 2006 - Informatik für Menschen: Proceedings: Beiträge der 36.Jahrestagung der Gesellschaft für Informatik e.V.(GI), 2.bis 6.Oktober 2006 in Dresden. (Ed.) Hochberger, Ch.; Liskowsky, R.*. Bonn: Köllen, 2006, (Lecture Notes in Informatics; P-94), 241 - 248. 107

PAPER 3: Vain, J., Raiend, K., Kull, A., Ernits, J., Synthesis of test purpose directed reactive planning tester for nondeterministic systems. *In: ASE'07 : 2007 ACM/IEEE International Conference on Automated Software Engineering, Atlanta, Georgia, November 5-9, 2007, proceedings: 22nd IEEE/ACM International Conference on Automated Software Engineering.* ACM Press, 2007, 363 - 372. 117

PAPER 4: Kull, A., Raiend, K., Vain, J., Kääramees, M., Case Study Based Performance Evaluation of Reactive Planning Tester. *In: CTIT Workshop Proceedings: 2nd Workshop on Model-based Testing in Practice (MoTiP 2009), June 23, 2009, Enschede, The Netherlands,* 2009, 87 – 96. 129

PAPER 5: Ernits, J. P., Kääramees, M., Raiend, K., Kull, A., Requirements-driven model-based testing of the IP Multimedia Subsystem. *In: BEC 2008: 2008 International Biennial Baltic Electronics Conference: Proceedings: 11th Biennial Baltic Electronics Conference, Tallinn University of Technology, October 6-8, 2008, Tallinn, Estonia.* Tallinn: Tallinn University of Technology, 2008, 203 - 206. 141

LIST OF PUBLICATIONS 147

Appendix A: FORMAL Transition LANGUAGE FOR MOTES (MTL) 149

Appendix B: Elulookirjeldus 152

Appendix C: Curriculum Vitae 154

Abstract

This thesis focuses on the development of model-based testing (MBT) technology that can be used for testing industrial-scale reactive systems and that is a reasonably simple for test engineers to use after a short period of study.

MBT is an automation approach for deriving tests automatically from the model of the implementation under test (IUT). In the domain of reactive systems the MBT is an automation approach for deriving tests automatically for functional black box tests of the IUT.

Researchers have been working on reactive systems MBT for the last couple of decades. Several commercial tools and many academic tools are available. The benefits of MBT are obvious to many, at least in the research community, but it has yet to be accepted by the industry. The main reasons for this are the poor usability (modelling complexity) and weak scalability of the test generation methods for industrial-scale testing.

The driving force behind my doctoral studies was a desire to create MBT technology for reactive systems that is easy for ordinary test engineers to use and that is equal to industrial-scale testing tasks. In order to be accepted by test engineers and to be easy to learn, the modelling paradigm of the technology must be well-known and accepted by the software engineering community. My aim was to develop an MBT technology that allows the IUT to be modelled using a modelling language as close to UML as possible, since UML is a widely accepted modelling standard in various branches of the industry.

In carrying out this mission, the thesis proposes several novel methods and techniques that form its theoretical foundation:

- iterative model checking-based test generation from deterministic models
- reactive planning tester synthesis for test generation from nondeterministic models
- requirement-driven testing through model composition

The iterative test generation method is based on explicit state model checking. It is well known that model checking suffers from state-space explosions if the complexity of the model increases and/or the goal of model checking broadens. The iterative model checking based test generation method builds the tests iteratively by splitting the model checking goal into simpler sub-goals. In each iteration only a simple sub-goal is solved, and the resulting test sequence is appended to the sequence generated so far according to the previous sub-goals. The method is therefore invulnerable to state-space explosions caused by the complexity of the test goal and allows tests to be generated from significantly larger and more complex models. Test sequences generated in such a way are suboptimal only. Splitting the goal into sub-goals can be viewed as another form of optimisation done either manually by the test engineer or automatically by the program.

The reactive planning tester-based method of deriving tests for nondeterministic systems has significant advantages over other known methods

of online nondeterministic systems testing due to its longer and parametrically adjustable planning horizon. The reactive planning tester is synthesised offline using reachability analysis of the model. During the online phase the reactive planning tester finds the suboptimal path to the next test coverage item in the model. Due to the longer planning horizon the tester meets the defined test goals significantly sooner than random walk or *anti-ant*-like algorithms, for example.

Systems testing processes in the industry are usually requirement-driven. This means that testing should verify whether a particular system requirement defined in requirements specification or other system specifications (interface specifications, functional specifications, etc.) is implemented correctly. The requirement-driven approach in MBT should be supported as early as the modelling phase using appropriate modelling formalisms. The requirements should be able to be modelled in a way that later allows parts of the model to be easily selected as test coverage items belonging to a particular requirement and a test case for this particular requirement to be generated. The thesis shows how the requirement-driven approach can be applied to the building of the model of the IUT using either composition of Uppaal [UPPA] automata or NModel [JVC+08] model program *features*.

The principles underlying the development of these novel methods and techniques have been presented at international conferences and published in referred journals. A selection of representative papers has been compiled and attached to this thesis.

The thesis describes the MBT tool that has been developed, MOTES, which implements novel test generation methods. MOTES is a test generator that generates test cases (from deterministic models) or reactive planning tester (from nondeterministic models) in the TTCN-3 language from extended finite state models (EFSM). The suitability of MOTES technology for industrial-scale testing tasks has been demonstrated in two case-studies in different branches of the industry – telecommunications software and telematics controllers.

Lühikokkuvõte

Käesolev doktoritöö esitleb reaktiivsete süsteemide mudelipõhiseks testimiseks loodud tehnoloogiat, mis on kasutatav tööstuslike testimisülesannete lahendamiseks igapäevases inseneripraktikas ning tavaliste testiinseneride jaoks mõistliku pikkusega koolituse käigus lihtsalt omandatav.

Mudelipõhine testimine on testide automatiseerimise meetod, mille puhul testid tuletatakse automaatselt testitava süsteemi mudelist. Reaktiivsete süsteemide puhul tähendab mudelipõhine testimine automaatset testide tuletamist süsteemi funktsionaalseks testimiseks "musta-kasti" meetodil.

Reaktiivsete süsteemide mudelipõhise testimise alaseid teadusuuringuid on tehtud juba mitme viimase kümnendi jooksul. Välja on töötatud mõned kaubanduslikud ja hulgaliselt akadeemilisi mudelipõhise testimise vahendeid. Mudelipõhise testimise plussid võrreldes traditsioonilise testimisega on hästi teada vähemalt akadeemilistes ringkondades. Vaatamata ilmsetele eelistele ei ole mudelipõhine testimine leidnud veel aktsepteerimist tööstusringkondades. Peamisteks põhjusteks on mudelipõhiste tehnoloogiate keeruline kasutatavus ja nende tehnoloogiate piiratud skaleeruvus suurte tööstuslike testimisülesannete lahendamiseks.

Minu doktoritöö põhiliseks liikumapanevaks jõuks oli luua reaktiivsete süsteemide testimiseks niisugune mudelipõhise testimise tehnoloogia, mida tavalistel testiinseneridel oleks lihtne kasutada ja mis samas skaleeruks hästi ka tööstuslike testimisülesannete vajadustele. Selleks, et mudelipõhine testimine lööks kasutajate hulgas läbi, peab ta põhinema üldtunnustatud ja laialt levinud modelleerimise formalismil. Minu eesmärgiks oli võimaldada kasutajal modelleerida formalismis, mis oleks võimalikult lähedane UML-le, kuna UML on levinud ja aktsepteeritud modelleerimiskeel mitmetes tööstusvaldkondades.

Võetud missiooni täitmiseks oli vaja lahendada rida teoreetilisi probleeme. Doktoritöö esitab mitmeid autori poolt väljatöötatud uudseid meetodeid ja tehnikaid, mis moodustavad doktoritöö teoreetilise aluse:

- iteratiivsel mudelikontrollil põhinev testide genereerimine deterministlikest süsteemi mudelitest,
- reaktiivse planeeriva testri sünteesimine mittedeterministlike süsteemide mudelipõhiseks testimiseks,
- süsteemi nõuetest juhitud mudelipõhine testimine ja nõuete esitamine nõuete mudelite kompositsioonina.

Doktoritöös esitatud iteratiivne testide genereerimise meetod kasutab *explicit state* mudelikontrolli meetodit. On üldtuntud fakt, et mudelikontrolli põhiprobleemiks on potentsiaalne olekuruumi plahvatus mudeli või saavutatavuse eesmärgi keerukuse kasvades. Iteratiivsel mudelikontrollil põhinev testide genereerimise meetod ehitab testi iteratiivselt, lahutades saavutatavuse eesmärgi lihtsamateks alameesmärkideks. Igal iteratsiooni sammul lahendatakse ainult üks lihtsam alameesmärk ja leitud testijada liidetakse varemleitud testijadale. Selle meetodi puhul testieesmärgi keerukuse

tõus testide genereerimise keerukust ei mõjuta ja olekuruumi plahvatuse tõenäosust ei suurenda.

Reaktiivse planeeriva testri abil testide genereerimine mittedeterministliku süsteemi testimiseks on märgatavalt efektiivsem muudest teadaolevatest mittedeterministlike süsteemide *on-the-fly* testide genereerimise meetoditest, seda just tänu meetodi pikemale ja parameetritega juhitavale planeerimishorisondile. Reaktiivselt planeeriv tester sünteesitakse enne testide täitmist kasutades saavutatavuse analüüsi süsteemi mudelil. Süsteemi testimisel leiab reaktiivselt planeeriv tester suboptimaalse tee järgmise testikatvuse elemendini automaatselt. Tänu pikemale planeerimishorisondile saavutab reaktiivselt planeeriv tester testieesmärgi oluliselt kiiremini kui juhusliku otsingu või "*anti-sipelga*" algoritmid.

Süsteemide testimise protsessid on tööstuses enamasti nõuetepõhised. See tähendab, et testimise ülesandeks on kindlaks teha süsteemi vastavus spetsifitseeritud nõuetele. Selleks, et kasutada nõuetepõhist testimist mudelipõhisel testimisel, peaks nõuete spetsiifika olema arvesse võetud juba modelleerimisnotatsiooni valikul. Nõudeid peab olema võimalik modelleerida nii, et neid oleks võimalik kasutada saadud mudelil testikatvuskriteeriumitena. Doktoritöös on esitatud nõuetepõhise testimise tarbeks kasutatav modelleerimise ja mudelite kompositsiooni põhimõte, mille järgi süsteem modelleeritakse kui Uppaal [UPPA] automaatide või NModel [JVC+08]mudelprogrammide kompositsioon.

Loetletud uudsed meetodid ja tehnikad on kantud ette rahvusvahelistel konverentsidel ning publitseeritud vastavate konverentside väljaannetes. Valik konverentsiartiklitest on lisatud käesolevasse doktoritöösse.

Doktoritöö kirjeldab MOTES mudelipõhise testimise vahendit, mis realiseerib töö teoreetilises osas esitatud uudsed meetodid ja tehnikad. MOTES on testigeneraator, mis genereerib testitava süsteemi käitumist kirjeldavast deterministlikust laiendatud lõplikust olekumasinast testijadasid või mittedeterministlikust mudelist reaktiivselt planeeriva testri. Mõlemal juhul on väljundiks TTCN-3 keeles kodeeritud testiprogramm. MOTES tehnoloogia rakendatavust tööstuslike testimisülesannete lahendamisel on demonstreeritud kahel tööstuslikul näiteülesandel, mis esindavad erinevaid tehnoloogia valdkondi – telekommunikatsiooni tarkvara ja telemaatika kontroller.

Acknowledgements

Writing this doctoral thesis would not have been possible without the help and support of the kind people around me. Above all, I would like to thank my wife Sirlis and the rest of my family for tolerating my being busy with the thesis and for spending too little time with them. I would like to thank my parents, to whom I am grateful for a wonderful childhood and the values they instilled in me. My father has been my role model: he earned his PhD at the age of 48, and I was determined to beat him on that score. At least I've equalled it! Thanks, dad, for the competition!

I would like to thank my research fellows – Kullo Raiend (PhD) from Elvior; my supervisor, Prof. Jüri Vain; Juhan Ernits (PhD); and Marko Kääramees, all from Tallinn University of Technology – for our weekly meetings over the past four years, our joint publications and for their cooperation on a number of research projects. Without their support, ideas and brainstorming this thesis would never have been possible. I would like to thank my supervisor Prof. Leo Mõtus for his support and for thinking outside of the box. His authority and the promises I gave him in terms of deadlines were vitally important factors in getting the thesis finished. I would like to acknowledge the financial, academic and technical support of the Tallinn University of Technology. I would also like to acknowledge my colleagues in Elvior who have implemented MOTES and worked on the case studies of the thesis. I would like to thank the ITEA2 D-MINT project members and project leader Colin Willcock for their valuable cooperation in applying model-based testing technology in the European industry and understanding the model-based testing requirements of the industry. I would like to thank Estonian Enterprises for their financial support of the Estonian consortium of the D-MINT project.

Last but not least, I would like to thank myself for finally having completed the thesis! I've spent too many years on it, and in its unfinished state it proved a constant source of anxiety and pressure. I'm happy that at last I have time for my next personal 'projects'.

Abbreviations

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
ATA	Analogue Telephone Adapter
C	A programming language
CASE	Computer Aided Software Engineering
C#	A programming language
C++	A programming language
EFSM	Extended Finite State Machine
ETSI	European Telecommunication Standards Institute
FBCU	Feeder Box Control Unit
FSM	Finite State Machine
GSM	Global System for Mobile communications
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IP	Internet Protocol
IRA	Iterated Random Abstraction
IUT	Implementation Under Test
LOC	Lines of Code
MBT	Model-Based Testing
MTL	Formal Transition Language for MOTES
RPT	Reactive Planning Tester
SA	System Adapter or SUT Adapter
SIP	Session Initiation Protocol
SUT	System (or Software) Under Test
TTCN-3	Testing and Test Control Notation Version 3
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
UML	Unified Modelling Language
USSD	Unstructured Supplementary Service Data
UTP	UML Testing Profile
VoIP	Voice over IP
XMI	XML Metadata Interchange
XML	Extensible Markup Language

PART I: OVERVIEW

1 INTRODUCTION

1.1 Organisation of thesis

The most important theoretical contributions of the research are described in the five research papers attached in PART III of the thesis. Several novel methods and techniques form the theoretical foundation of the thesis:

- iterative model checking-based test generation from deterministic models
- reactive planning tester synthesis for test generation from nondeterministic models
- requirement-driven testing through model composition

The papers are presented in logical and chronological order. This order represents the history of the research, which started with issues of generating tests from deterministic models (PAPER 1, PAPER 2). The research continued with more complex issues of generating tests from nondeterministic models (PAPER 3, PAPER 4). PAPER 5 proposes feature-by-feature modelling and the idea of model composition to adapt model-based testing for widely used requirement-driven testing processes.

PART I includes an introduction to and scope of the thesis and related work. An overview of the research papers is presented here as well as an unpublished improvement on the test generation method originally published in PAPER 1.

PART II of the thesis demonstrates the model-based test tool MOTES, which implements the research methods described in PART I. Two case studies are described here to demonstrate the feasibility of MOTES technology and its application in solving industrial-scale model-based testing tasks.

1.2 Field of research

The thesis focuses on model-based functional testing of reactive systems using the black box testing [BEI95] principle. The task of black box testing (also known as ‘behavioural testing’ or ‘functional testing’) is to verify that the implementation under test (IUT) conforms to the specifications. The IUT is viewed as a black box which transforms input into output according to its specifications. In automated black box testing the IUT is executed against the test tool, which acts as the environment for the IUT. The test tool provides input for the IUT and examines the actual output against that expected. Black box tests can be automated by the scripts that are executed by the test tool. Scripts can be implemented in general-purpose programming languages like Java [JAVA], in scripting languages like Perl [PERL], in test-dedicated languages like TTCN-3 [TTCN] and more.

Although script-based automation increases black box testing productivity and tests repeatability in regression tests, practitioners have experienced serious problems in traditional script-based black box testing:

- Test engineers are rarely able to create a sufficient amount of test scripts manually to achieve adequate test coverage. Manual scripting is a time-consuming activity and involves an effort comparable to implementing the IUT itself.

- In industrial projects the amount of test scripts may increase enormously. In the systems maintenance phase, the regression tests are used to verify that the old features of the IUT are still working after the implementation is changed. The IUT changes which affect its external interfaces are the reason why test scripts should be modified accordingly. In the case of a considerable number of legacy test scripts, maintenance costs may prove unacceptably high.

- The high maintenance costs of test scripts is the reason why test coverage of legacy test suites decreases over time, since only subsets of existing test scripts tend to be updated.

Model-based testing (MBT) refers to the automation of software black box testing where the test cases (test scripts) are derived, in whole or in part, from a model that describes the expected behaviour of the IUT [UL06]. The MBT workflow consists of the following primary steps:

- The external behaviour of the IUT is modelled according to the relevant specifications of the IUT. The model presents the correct expected behaviour of the IUT.

- The test purposes (test goals) are defined by the test engineer. The tests are always generated for particular test purposes that define the scope and coverage of the generated tests.

- The tests are generated automatically from the IUT model by a test generation tool using the test purposes.

- The generated tests are executed against the IUT to verify its conformity to the model.

MBT has the following benefits compared to traditional script-based black box testing:

- Modelling of the IUT for testing purposes has a similar effect on the quality of the IUT specification as the modelling of the IUT does on design and implementation purposes – it helps detect possible inconsistencies and ambiguities in the specification before actual testing.

- Automatic test generation is time-effective and cost-effective [PPW+05]. After creating the IUT model and defining the test goals, the remaining test generation can be fully automatic.

- Automatic test generation gives better test coverage than manually created tests [UTT05]. Tests can be created from an IUT model for many different test goals simply by defining the new test goal and generating new tests just by pressing a button. In this way, tests for different use cases can easily be generated from the same model. This is not feasible with manually created tests: implementing all of them manually is normally too costly.

- The maintainability of legacy tests improves. After updates in the behaviour or interfaces of the IUT there is no need to go through a huge number of legacy tests to update them accordingly. Updating the model alone is enough, and all tests can then be regenerated automatically from the model.

Considerable research has been done into MBT and many relatively mature prototype academic tools and a number of commercial tools are already available.

The transfer of the results of research into MBT to practical application has met with various problems. Many research tools use modelling formalisms which are different from those adopted in the industry for model-based design and development. This is partly because the models in design and development can be semi-formal, whereas an automatic MBT tool can only work on completely formal models. The difficulties of using formal languages for modelling IUTs have delayed their use in the industry. Another restriction in the practical use of MBT tools is the size and complexity of the models in the industry and the poor scalability of MBT tools in supporting them.

1.3 Motivation

With the increasing penetration of software-intensive systems into our everyday lives, the requirements of system functionalities and features have increased. At the same time, the requirements of system quality and reliability have also increased. With a growing number of requirements, the complexity of software-intensive systems is likewise growing. This is combined with increasing error-proneness, which is related to shortened development times.

The need to develop high-quality applications in a shorter time and at a lower cost requires more structured and automated analysis, design, and testing techniques. Automated testing techniques are more important and add more value to products and services delivered to the market as rapidly as possible, with minimal risks. Test automation becomes more vital in maintaining a technological edge and controlling costs.

According to the Journal of Information Technology Education [SZF+04], companies have allocated more than 40% of their product development time to testing. A NIST study in the USA in 2002 [NIST] showed that more than 59.5 billion dollars is lost every year as a result of quality problems in software applications, such as capabilities that do not reflect business needs, recurrent stability problems, errors, and crashes that sometimes lead to heavy business losses.

This explains why research on increasing testing effectiveness can lead to significant cost-saving effects.

1.4 Objectives of thesis

The research objectives of the thesis are as follows:

- To develop a novel, well-scalable method for generating tests from deterministic EFSM (Extended Finite State Machine) models of the IUT; to develop the test generator implementing the method; and to evaluate the feasibility of the method and the test generator in industrial-scale case studies.

- To develop a novel, well-scalable method for generating tests from nondeterministic EFSM models of the IUT which, in online testing, achieves test goals faster than any known online testing method; and to develop the test generator implementing the method; to evaluate the feasibility of the method and test generator in industrial-scale case studies.

The research method used in the thesis is predominantly experimental and constructive, and the focus of the thesis is therefore practical rather than theoretical. The solutions to the problems listed above are constructed using incremental development cycles followed by empirical evaluations.

2 SCOPE AND RELATED WORK

2.1 Characteristics of reactive systems

Each model-based testing approach is highly dependent on the characteristics of the IUT. These characteristics define the suitable modelling paradigms for each kind of IUT. In this thesis the MBT is applied for the testing of reactive systems. Reactive systems are systems which must continually respond to the stimuli from their environment. As such, the computations of the reactive system are driven by the stimuli received from the environment. In response to the stimuli, the IUT changes its internal state and produces responses which are sent back to the environment [HEL+05]. Examples of reactive systems include controllers in branches of industry like telecommunications, automotives, avionics and transport. Most real-time systems include embedded reactive software. Reactive systems are often critical to safety and must be thoroughly tested to ensure that they meet specific functional and non-functional requirements. Since the number of potential input sequences that reactive systems must handle is infinite, a great deal of testing is needed to be sure that the system behaves as expected.

Software embedded in reactive systems has important characteristics that should be taken into account in MBT methods:

- Continuous execution – typical reactive software should never stop functioning
- Concurrency inside the software to handle concurrent processes in the environment
- Asynchronous communication between software components internally and between the components of the environment
- History-based behaviour – reactions to stimuli do not depend on current stimuli alone but also on the history of past stimuli
- Time-dependent behaviour – behaviour depends on the order of stimuli, the time intervals between consequent stimuli, and absolute time
- Multiple acceptable outputs on the same stimuli (output nondeterminism) are possible due to the concurrency and race conditions inside the reactive system

2.2 Taxonomy of model-based testing

Model-based testing is a large domain. The term '*model-based testing*' is widely used, with slightly different meanings. Surveys on different model-based testing approaches are presented in [BJK+05], [UTT05], [UL06], [UPL06].

In this thesis only a small proportion of the MBT field is covered. To place the thesis in the context of the MBT landscape as a whole, the framework of model-based testing taxonomy introduced in [UPL06] is used.

The *taxonomy* of MBT includes three general *classes*: the model, test generation and test execution. Each *class* is divided into *categories*. The model-related categories are *subject*, *independence*, *characteristics* and *paradigm*. The test generation *class* is split into *test selection criteria* and *technology*, while the test execution is divided into execution options. See Figure 1.

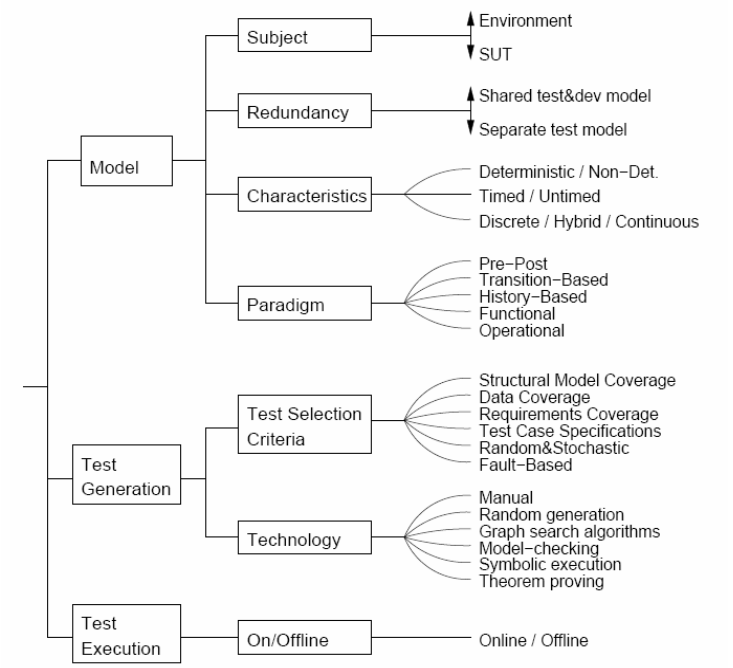

Figure 1: Overview of the Taxonomy for MBT in [UPL06]

The *subject* defines what is modelled. The *subject* can be the intended behaviour of the IUT or the possible behaviour of the environment of the IUT, or a combination of the two.

Scope of thesis:
The intended behaviour of the IUT is the *subject* in this research. The model presents the expected correct behaviour of the IUT, which acts as the 'criterion of truth'.

The model for IUT black box testing expresses the intended behaviour of the IUT in its interfaces as seen from the outside of the IUT (black box). This approach assumes that the IUT is controllable and observable via its interfaces. The model expresses the behaviour of the IUT as an expected reaction observed in the observable interfaces to the external stimuli received by the controllable interfaces. This explains why the design model of system implementation cannot be used as such for black box testing purposes. Although the sources of the models for design and test purposes are the same, they model different aspects of the systems for different purposes. The first concentrates on the inner architecture and behaviour of implementation. The second concentrates solely on external behaviour, which is outside the controllable and observable.

The *independence* aspect reflects the source of the model. If the model is designed directly from informal requirements specifically for testing purposes by a team that is independent of IUT developers, there will be a high degree of independence between the models created for testing and development purposes, and the testing is more likely to discover significant errors. Independence between models for testing and development purposes increases the chance that during modelling for testing purposes ambiguities will be able to be detected in the IUT specifications as a side-effect of the modelling. Reusing too many existing development models in creating models for testing purposes can weaken the independence of the test suite and reduce its capabilities to detect implementation errors.

Scope of thesis:
The *independence* aspect of the model is beyond the scope of this thesis.
In the context of this research it is important that the model of the IUT for black box testing purposes does exist, but it is unimportant whether the model was created independently of the development model by an independent team.

The *characteristics* aspect describes the properties of the model that are dependent on the characteristics of the IUT. These characteristics relate to nondeterminism, to the incorporation of timing issues and to the continuous or event-discrete nature of the model and the IUT.

- Deterministic models are those where the model's output is deterministically defined by the model input and the current state. Nondeterministic models are those where the input of the model can generate many alternative outputs in the current state of the model. Nondeterminism stems partially from the internal parallel processes of the IUT, timing and hardware-related asynchronous processes. Other sources of nondeterminism are the higher abstraction level of the model

compared to IUT implementation and the ambiguities in the specifications of the IUT.

> **Scope of thesis:**
> Both deterministic and nondeterministic models fall within the scope of this research.

- Timed models are used to express the real-time constraints on the model. In general, the timing correctness of reactive systems is an important issue to test.

> **Scope of thesis:**
> Systems with hard real-time constraints are beyond the scope of this thesis. A timed model with time constraints is not used. Only timers and timeouts are used in the model to detect missing or delayed events in the IUT.

- The model can be discrete, continuous or a mixture of both.

> **Scope of thesis:**
> The assumption is that the model is discrete. The model receives its input events at discrete time moments and also produces output events at discrete time moments.

The *paradigm* aspect reflects the style of the model and the notation of creating it. Various modelling paradigms are available. In the modelling of reactive systems the most commonly used modelling paradigms are transition-based ones. These are most natural for presenting the state-based behaviour of the IUT and the reactive relationships between IUT inputs and outputs. Typically different state machine notations are used to model reactive systems. Examples of transition-based modelling notations are:

- Finite state machine (FSM) is a notation where the nodes represent the states of the system and the transitions represent the actions or operations of the system. Textual information is used to express the input-output relationship on the transitions.

- Extended finite state machine (EFSM) is FSM with variables, guard conditions and variable assignment operations attached to transitions. EFSM is a more compact state machine presentation than FSM, with improved expression power.

- Statecharts (e.g. UML State Machines [UML], and STATEMATE Statecharts [HN96]) are EFSMs with features that increase the expressive power at the next level. Parallel and hierarchical states are represented on Statecharts.

Scope of thesis: EFSMs are used to model the IUT.

The *test selection criteria* aspect describes the different test selection criteria used in test generation.

- *Structural model coverage* is a test selection criterion that specifies the intended test coverage in terms of model structural elements such as states and transitions.
    - Control-flow oriented coverage criteria for models are derived from similar code coverage criteria like *statements*, *decisions*, *loops* and *path coverage* [MM63].
    - Data-flow oriented coverage criteria attempt to cover the control flow graph elements that define or use data variables. Examples of data-flow oriented coverage criteria are *all definition−use-pairs*, *all definitions* and *all uses* [BEI95].
    - Transition-based coverage criteria define the transitions of the model that should be visited by generated tests. Examples of transition-based coverage are *all paths*, *all transitions* and *selected transitions* [UL06].
- *Data coverage* is a test selection criterion that specifies the intended test input data coverage. The three most often used data coverage criteria are:
    - Boundary value criteria select test input values at the boundaries of the input domain [KPL+04].
    - Statistical data coverage criteria select test input values that follow a certain statistical distribution [HOW97].
    - Pair-wise testing criteria select test input values so that all pairs of input values are tested [BG02].
- *Requirements coverage:* this aims to generate tests so that all of the requirements of the IUT are tested [UL06].

- *Explicit test case specification:* this is test selection by the test engineer, who explicitly defines the test objectives of the model. The notation used to express the test objectives may be the same as the notation used for the model [UPL06]. Notations commonly used for test objectives include FSM, UML Testing Profile (UTP) [UTP], regular expressions, temporal logic formulas, constraints and Markov chains [JUS08].

- *Random and stochastic criteria:* a typical approach is to use a Markov chain [MT09] to specify the expected IUT usage profile. Another example is to use a statistical usage model in addition to the behavioural model of the IUT [CLP08].

- *Fault-based criteria:* these rely on knowledge of typically occurring faults, often designed in the form of a fault model.

> **Scope of thesis:**
> Transition-based structural model coverage criteria are used for test selection. Requirements-coverage test selection is implemented by defining the requirements coverage using structural coverage over the model elements involved in modelling the requirement.

The *test generation technology* paradigm describes the different underlying technologies used in MBT methods:

- *Manual/Automatic:* tests can be generated automatically by test generation tools or developed manually.

> **Scope of thesis:**
> In the thesis the control flow of the tests is generated automatically, while the test data used by the tests is prepared manually.

- *Random generation:* random generation of tests is done by sampling the input space of a system (monkey tests).

- *Graph search algorithms:* dedicated graph search algorithms include node or arc coverage algorithms such as the Chinese Postman algorithm [EJ73], which covers each arc at least once.

> **Scope of thesis:**
> Graph search algorithms and reachability analysis are used in a novel method to synthesise a reactive planning tester for testing nondeterministic systems.

- *Model checking:* model checking is a technology used to verify the properties of a system using reachability analysis [CGP99]. The general idea of test case generation with a model checker is to first specify the test case in terms of reachability properties e.g. "eventually, a certain state is reached or a certain transition fires". A model checker then yields traces which reach the given state or which eventually make the transition fire. Different model checking techniques exist e.g. explicit state model checking and symbolic model checking.

  Scope of thesis:
  The explicit state model checking is applied to test generation for deterministic systems.

- *Symbolic execution:* the idea behind symbolic execution is to run an executable model with sets of input constraints instead of single input values to generate traces [MA00]. To derive test cases these traces are instantiated with concrete input values

- *Theorem proving:* this is used to check the satisfiability of formulas in the models [Duf91]. Theorem provers can be used in test generation in a similar way to model checkers, with a theorem prover replacing the model checker.

- *Online/offline test generation* defines whether the tests are generated in advance of test execution (offline test generation) or simultaneously with it (online test generation). The term '*on-the-fly*' is often used for online test generation. Offline test generation is mostly used to generate tests for deterministic IUT, while on-the-fly generation is mostly used with nondeterministic IUT.

  Scope of thesis:
  Offline test generation is used to generate tests for deterministic systems.
  A mixture of offline and online test generation is used to generate tests for nondeterministic systems.

2.3 Scope summary and related work

The scope of this thesis was extracted from the MBT landscape as a whole in the previous section using the framework of model-based testing taxonomy. This section summarises the scope and provides an overview of the work related to this scope.

Tests are generated from the model of the IUT. The model is derived for testing purposes from the specifications of the IUT (requirements specifications, interface specifications, et al.). Only the expected (correct) behaviour of the IUT

is modelled. It is assumed in the thesis that the model of the IUT can be either deterministic or nondeterministic.

From the deterministic model of the IUT, test cases with the test oracle are generated offline and executed online against the IUT. A test oracle is a mechanism for determining whether a test has been successful or not. It is used by comparing the output(s) of the system under testing, (for a given test case input), to the outputs that the oracle determines the product should have. Such offline-generated test cases implement a test sequence that covers the IUT according to the test coverage defined by the user. Many MBT solutions are restricted to generating the test from deterministic models only. The following model-based testing tools fall into this category: Conformiq Qtronic [Hui07], Leirios Test Generator [BBC+06] (marketed as Smartesting), Agatha [GGR+06], SpecExplorer [VCG+08], NModel [JVC+08], ATG [GMS+07], MiLEST [JUS08], and Reactis Tester [REAC].

Nondeterminism is an essential characteristic of reactive systems, and working solely with deterministic models leads to excessive narrowing of the testing field. This thesis proposes an approach for nondeterministic systems where the reactive planning tester is generated offline from the nondeterministic model of the IUT. The reactive planning tester is executed online against the IUT. It is the task of the reactive planning tester to generate tests on the fly, taking into account the nondeterministic responses from the IUT and the intended test coverage defined by the user. Other tools providing on-the-fly test generation for the testing of nondeterministic systems are SpecExplorer [VCG+08], NModel [JVC+08], and Uppaal TRON [LMN+05].

This thesis focuses on IUT with soft real-time properties only. Hard real-time constraints and performance testing are beyond the scope of the thesis. In the context of the thesis only the correct externally visible behaviour of the IUT and the order of events are important. Timers and timeouts are represented in the IUT model to denote missing or delayed events from the IUT and to present delays in the test sequences.

It is assumed in the thesis that the model of the IUT is discrete. The model receives its input events at discrete time moments and also produces output events at discrete time moments.

EFSM as a subset of a UML state machine is used for modelling the IUT in the thesis. The following modelling notations were under consideration in the early stages of research: textual modelling language, FSM, EFSM and Statecharts.

Textual modelling languages such as Spec# (a modelling language for SpecExplorer [VCG+08]) and C# (a modelling language for NModel [JVC+08]), are high-level programming languages. They have the expression power that enables the modelling of almost any aspects of the system at a very detailed level. Textual modelling languages are hard for test engineers with limited programming skills to learn. They lack graphical relationships between model components.

FSMs have received extensive theoretical study by the model-based testing community. Researchers have been working on FSM test generation algorithms since the early 1960s. Several complete test generation methods have been invented for generating tests that guarantee the IUT implementation identity with the corresponding FSM model. Those algorithms are not typically used in industrial practice because they are too restrictive in that they make strong assumptions about the model and the IUT. FSMs lack the expression power required to model industrial-scale systems. Serious models built in FSM grow to be huge and lack readability. NModel [JVC+08], for example, supports FSM models in addition to C# model programs.

Statecharts [HN96] is a state machine with hierarchical and parallel states. It is a powerful state machine notation which is widely accepted in the industry. Unfortunately, test generation from Statecharts presents many obstacles which are hard to overcome due to the existence of parallel states. Algorithms are available for flattening the Statecharts [Was04], [KBC05], but due to this flattening the traceability from generated tests back to the original model becomes difficult. Tracing generated tests back to the model is urgently needed during test execution to identify the aspects of the model that implementation violates. Statecharts modelling notation is used with some limitations in Qtronic [Hui07] and ATG [GMS+07].

EFSM has weaker expression power compared to the Statecharts. EFSM is a compact presentation of FSM, making it a better choice than FSM. The benefit of EFSM as a graphical presentation over textual modelling notations is that it is more readable and understandable, but there are still people who prefer only textual presentations. The choice of the modelling language can be considered a matter of taste. EFSM, like FSM, is a theoretically well studied notation. There are a large number of tools and techniques for manipulating EFSMs. The availability of model checking tools for experimenting with test generation from EFSM was one of the reasons for its selection for the modelling notation in this research. EFSMs have gained wide acceptance in software modelling [Ho02] and are used as semantic models for specification languages such as Statecharts and UML state machines. The timed automata notation used by the Uppaal TRON [LMN+05] is an extension of EFSM also.

Model structural coverage criteria are used for test selection, which is based on finding transition sequences that cover user-defined model elements (coverage items). Tests are generated according to the following test coverage criteria: *selected states*, *selected transitions*, *all transitions*, *all states* and *all transition pairs*. The requirements coverage criterion is implemented by combining the structural coverage items. Test selection based on model structural coverage criteria is used also by Qtronic [Hui07], Leirios [BBC+06], Agatha [GGR+06], and Uppaal TRON [LMN+05].

Test sequences (test control flow) are generated from the model automatically, but the test data instances that satisfy the test sequences are prepared manually in the approach proposed in this thesis. Automatic generation of test data instances is implemented in Qtronic [Hui07], Agatha

[GGR+06], SpecExplorer [VCG+08], NModel [JVC+08], MiLEST [JUS08], and Reactis Tester [REAC] in addition to the generation of test sequences.

Explicit state model checking is used for offline test generation from deterministic models. In this research, the Uppaal Cora [UpCo] model checker is used as a test generation engine. Using explicit state model checking for test generation is not a new idea [HMR04] – the most commonly used model checkers in the context of testing are:

- explicit state model checker SPIN (Simple Promela Interpreter) [Hol97];
- Symbolic Analysis Laboratory SAL [MOR+04], which supports both symbolic and bounded model checking;
- symbolic model checker SMV [McM92] and its derivative NuSMV [CCG+99], which support symbolic and bounded model checking.

Applying explicit state model checking iteratively over the set of coverage items is a novel approach in this thesis and outperforms 'standard' explicit state model checking-based test generation in scalability.

A novel graph search and reachability algorithm is used to synthesise a reactive planning tester for test generation from nondeterministic models. The method outperforms online test generation methods which rely on random choice, such as TorX [BFV+99], Uppal TRON [LMN+05] and SpecExplorer [VCG+08], and *anti-ant* heuristic-based state-space exploration methods introduced in [LL05] and used in [VRC06] because the reactive planning tester results in shorter tests due to the longer planning horizon [PAPER 4]. A mixture of offline and online test generation is used. In the offline phase the reactive planning tester is synthesised from the model, while in the online phase the tester is executed. It autonomously finds a sub-optimal (in terms of test sequence length) path that traverses the test coverage items.

The test assessment algorithm is derived from the model of the IUT that expresses the expected (correct) behaviour of the system. Test assessment is carried out automatically by the generated TTCN-3 test cases in the online phase.

3 GENERATING TESTS FROM DETERMINISTIC MODELS

3.1 Generating tests from EFSM using guided model-checking

The theoretical contribution of the thesis to generate tests from deterministic EFSM is presented in PAPER 1 and PAPER 2. The papers address test generation from the deterministic model of the IUT using model checking. PAPER 1 describes the test generation method and PAPER 2 describes the implementation of the method.

The first paper is entitled **"Generating tests from EFSM models using guided model checking and iterated search refinement" [PAPER 1]**. The paper was written by Juhan Ernits, Kullo Raiend, Jüri Vain and the author of this thesis. It was presented at the First Combined International Workshops FATES 2006 and RV 2006 in Seattle, USA, in August 2006.

The paper describes a method for generating test sequences from the models of the IUT using the guided model checker Uppaal Cora [UpCo]. The IUT is modelled using EFSM modelling notation. The motivation to work with EFSMs was justified because the specifications provided in terms of (for example) suitably restricted UML Statecharts can be converted into the equivalent EFSMs, and EFSMs provide a semantically well-defined model representation that can be applied to test generation. The algorithms for searching test sequences are relatively simple if the software is modelled using finite state machines (FSM), because FSM does not have variables and guard conditions, unlike EFSM. The existence of variables and guard conditions makes the search of test sequences much more complex. This complexity is caused by the large number of value combinations the variables can have and by the need to satisfy guard conditions on transitions. A well-known option for generating tests from EFSMs is to use the search machinery provided out of the box by model checkers [CGP99]. With model checkers the coverage of a test case is specified using reachability properties on the model. The model checker solves the reachability task and generates a witness trace that can be transformed into an abstract test sequence satisfying the specified test coverage criteria. The abstract test sequence can be further encoded as executable test code.

The test generation method presented in the paper allows for the specification of various structural test coverage criteria in EFSM – for example, *selected states/transitions*, *all transitions* and *all transition pairs*. The paper proposes a method that combines the techniques of model construction with a novel method of reachability search in model checking which we have named *iterated random abstraction* (IRA). The problem of generating test sequences is formulated as a bounded reachability problem and solved through model checking.

Model checking [BJK+05] is a state-space exploration-based method. The critical factor in space exploration-based methods is scalability i.e. the ability to

handle the exponential growth of the search state-space. One example of an issue where scalability quickly becomes acute is generating tests according to structural test coverage criteria that result in long witness traces – for example, *all transitions* test coverage or *all possible subsequences of transitions of length* $k > 1$.

The paper describes a method of constructing Uppaal [UPPA] models to achieve test sequences that satisfy the test coverage criteria and experiments with the search options of Uppaal to achieve test sequences that are suboptimal in terms of length. The paper shows how guiding the search with cost variables influences the lengths and required amount of memory of test generation. The authors apply the novel bitstate hashing state-space reduction-based iterated random abstraction method to shorten the length of the test sequences with respect to the length gained using a depth-first search. In fact, the method merges search-guiding with the iterated random abstractions to reduce the lengths of the generated test sequences. As a result, the scalability of applying explicit state model checking for test generation increases. Uppaal and its guided counterpart Uppaal Cora [UpCo] were used because they enabled the influence of guiding and iterated random abstraction in the context of test generation to be demonstrated.

The test generation method and different search strategies were compared by applying them to a stopwatch and INRES protocol [Hog91] case studies. The authors carried out a comparison of different search strategies on a stopwatch model. The comparison confirmed what has previously been stated: that explicit state model checking does not scale well for test sequence generation purposes, as breadth-first searches, which would yield a short sequence, run out of memory with simple models and depth-first searches produce very long sequences while consuming large amounts of memory as the model becomes more complex. A bitstate hashing-based iterated random abstraction method for checking reachability proved more scalable for test generation than the traditional search strategies used in model checking. Additionally, extending the EFSM model with guiding cost expressions yielded better results, and some heuristic tuning of the cost expressions drastically improved the results.

The most important contribution of the paper is the proof it presents of the concept of applying guided model checking in conjunction with the iterated random abstraction method to generate suboptimal test sequences. Experiments have shown that the lengths of test sequences generated using explicit state model checking can be improved by combining guiding and iterated random abstractions.

The second paper, entitled **“Generating TTCN-3 test cases from EFSM models of reactive software using model checking” [PAPER 2],** is a practical continuation of the studies reported in PAPER1. It describes a test generation tool that implements the method. The paper was written by Juhan Ernits, Kullo Raiend, Jüri Vain and the author of this thesis. The paper was presented at the

MOTES 2006 conference as part of *Beiträge der 36.Jahrestagung der Gesellschaft für Informatik* in Dresden, Germany, in October 2006.

The paper describes the architecture and workflow of the model-based testing tool for the generation of executable TTCN-3 [GHR+03] test cases from a deterministic, strongly connected system model using the Uppaal Cora [UpCo] model checker. The test cases are generated for black box testing of the reactive IUT. In order to specify the observable behaviour of the IUT, the formal transition language for EFSM [Appendix A] is defined. This approach allows the user to specify various structural test coverage criteria of EFSMs – for example, *selected states/transitions*, *all transitions* and *all transition pairs*. The Uppaal model is constructed from the model of the IUT and a coverage criterion. Uppaal Cora is used to find an abstract test sequence that is suboptimal in terms of length. The problem of generating test sequences is formulated as a bounded reachability problem and solved by the Uppaal Cora model checker. When a model checker solves a reachability task it generates a witness trace that corresponds to an abstract test sequence. The abstract test sequence is further encoded to test code in the TTCN-3 language. TTCN-3 was selected as the output because it is a dedicated language for testing purposes, it is standardised, and it is widely accepted in the software testing industry (especially so in telecommunications). The rules for transforming the abstract test sequences to TTCN-3 are presented. In the approach the test cases are generated offline i.e. test cases are generated from the model of the IUT before the tests are run.

The most important contribution of the paper lies in the value of proposing the complete procedure of transforming a formal IUT model into executable test code that satisfies user defined structural coverage criteria.

In the paper the test generator tool does not yet have a name. It was later dubbed ‘MOTES’ after the name of the conference at which the implementation principles were published.

3.2 Generating tests from EFSM incrementally using model checking

The test generation method which was presented in PAPER 1 and which was implemented according to the test tool architecture and workflow presented in PAPER 2 suffers from two issues that prevent it from being applied without improvements for industrial-scale applications testing.

First, the method that relies on explicit state model checking suffers from the state-space explosion. This is a common problem in model checking. The method works well in experiments with small models and simple test coverage criteria, as reported in PAPER 1.

The appearance of the state-space explosion depends on the size of the model and the complexity of the reachability problem to be solved. Having implemented the initial tool it was possible to experiment with industrial-scale models, and the results were not encouraging.

Second, defining the reachability problem as a conjunction of the test coverage items and solving the reachability problem by model checking

provides the shortest test sequence through all of the coverage items that the specific model-checker can find. With this algorithm the order and amount of coverage items in the test sequence is out of the control of the test engineer. The model checker finds a test sequence that covers all of the coverage items in an order that depends on the search strategy of the model checker. In many cases this reduces the flexibility of defining test goals. Often test engineers want to generate tests according to more complex scenarios – for example, first covering item `A` and then `B`, then covering items `C` and `D` in any order and finally covering `A` and then `B` again.

To overcome these issues, the method introduced in PAPER 1 was improved by using model checking incrementally in test generation. The core idea of incremental model checking lies in splitting the model checking problem into sub-problems and solving the sub-problems one by one, incrementally. Instead of immediately solving a reachability problem formulated as a conjunction of test coverage items, a set of coverage items is serialised and the reachability problem is solved individually for each coverage item in the set, one by one.

The main advantage of incremental model checking over the original method is its better scalability. The original method uses a conjunction of test coverage items as a reachability problem. Each additional item in the conjunction makes solving the reachability problem harder because the model-checker has to find a trace that satisfies all of the items in the conjunction. An improved method with incremental model checking solves the reachability problem of only one coverage item in each model checking iteration. This means that in contrast to the original method, the complexity of the reachability task no longer increases exponentially with the number of coverage items. Instead, the complexity of the method with incremental model checking is only linear in the number of coverage items. Contemporary model checkers can solve the reachability problem of only one coverage item on quite large models. More extensive measurements are needed to make any quantitative conclusions about the limits of the scalability of the method. Industrial-scale case studies performed with the MOTES test generator, which implements incremental model checking-based test generation, have yet to witness any issues caused by the state-space explosion.

The second advantage of incremental model checking over the original method is better control over test coverage. With the original method the model checker decides the order in which it visits the test coverage items. It is not possible to control how many times and in what order the test coverage items are visited. The incremental method defines the test coverage as an ordered set of coverage items where the order of the coverage items is defined using a regular expression with interleaving parallelism throughout the coverage items. For instance, the regular expression `<A; B; (C || D); A; B>` defines the following order of traversing coverage items `A`, `B`, `C`, and `D`. The test generation starts with the solving of the reachability problem of coverage item A from the initial state of the model. Next, the test generation increment starts from the state reached after reaching coverage item `A` and ends when test

coverage item B is reached. Next, the reachability task for C and D is solved by model checking, finding the test sequence starting from the state after reaching B and ending the state after covering both C and D. Next, the test sequence from the current state to reach coverage item A is found and, finally, the test sequence from the current state to reach coverage item B is found. As a result of incremental model checking, the user can generate complex test scenarios throughout the model. This feature of the method is an essential prerequisite for using the method in the requirement-driven test generation process in defining complex test coverage criteria for complex system requirements using structural test coverage criteria. It makes it possible to use the method in generating tests for industrial-scale applications, as demonstrated in the case studies in Chapter 7.

4 GENERATING TESTS FROM NONDETERMINISTIC MODELS

The theoretical contribution of this thesis in generating tests from nondeterministic EFSM is presented in PAPER 3 and PAPER 4.

4.1 Synthesis of test purpose-directed reactive planning tester

The third paper is entitled **"Synthesis of test purpose directed reactive planning tester for nondeterministic systems" [PAPER 3]**. The authors of the paper were Jüri Vain, Kullo Raiend, Juhan Ernits and the author of this thesis. The paper was presented at the ACM/IEEE International Conference on Automated Software Engineering (ASE'07) in Atlanta, USA in November 2007.

The paper describes the model-based construction of an on-the-fly tester for the black box testing of the IUT. The external behaviour of the IUT is modelled as an output-observable nondeterministic EFSM with the assumption that all transition paths are feasible.

On-the-fly test generation is considered to be the most appropriate technique for nondeterministic IUT [VCG+08]. The term '*on-the-fly*' denotes a test generation and execution algorithm that computes successive stimuli incrementally at runtime, directed by the test purpose and the observed outputs of the IUT. A test purpose (or test goal) is a specific objective or property of the IUT that the tester sets out to test. The state-space explosion problem experienced with many offline test generation methods is eliminated by on-the-fly techniques because only a limited part of the state-space needs to be explored at any point in time. On the other hand, exhaustive planning is difficult on the fly due to the limitations of the available computational resources to meet the required response time of the tester. The tester cannot make calculations that are too long after the observed IUT outputs because the IUT may have time limits for responses. The simplest approach to the selection of test stimuli is to apply the 'random walk strategy', where no test sequence has an advantage over others. This is inefficient, because it is based on the random exploration of the state-space and leads to test cases that are unreasonably long and may not achieve the specified test goal. To overcome this deficiency, additional heuristics are applied to guide the exploration of the state-space [VRC06]. The other extreme of guiding is exhaustive planning by solving constraint systems at each stage of the test. For instance, the witness trace generated by model checking provides possibly the optimal selection of next test stimuli. The critical issue in the case of explicit state model checking algorithms is the size and complexity of the model leading to the explosion of the state-space, especially in cases such as 'combination lock' or deep loops in the model [HMR04].

PAPER 3 proposes a balance between the tradeoffs of using simple heuristics and exhaustive planning methods for on-the-fly testing. The principles of reactive planning are applied to the problem of test planning under

uncertainty. Reactive planning operates in a timely fashion and hence can cope with highly dynamic and unpredictable environments [WN97]. Just one subsequent input is computed at every step, based on the current context. Instead of producing a complete test plan with branches (a test tree), a set of decision rules is produced. The rules are constructed by applying offline analysis based on the given IUT model and the test purpose. A reactive planning tester is synthesised from the IUT model. This tester is able to generate test inputs on the fly depending on the observed reactions of the IUT and the test purpose without having a preset test tree generated in advance. The proposed approach leads to a tester that directs the IUT efficiently towards the user-defined test purpose during online test execution.

We focus on test purposes where the coverage items can be defined as a set of traps associated with the transitions of the IUT model [HMR04]. Traps are Boolean variables that are associated with the transitions of the IUT model and are used to measure the progress of the test run. A situation where all traps have been reached means that the test purpose has been achieved. The goal of the tester is to generate a test sequence so that all traps are visited at least once during the test.

PAPER 3 presents a way of constructing a tester which, at runtime, selects a suboptimal test path from trap to trap by finding the shortest path to the next unvisited trap. The principles of reactive planning are implemented in the form of decision rules for selecting the shortest paths at runtime. The tester is synthesised as an EFSM where the rules for online planning are derived during tester synthesis and encoded in the transition guards of the EFSM. The decision rules are constructed in advance of test execution from the IUT model and test purpose.

At each step of test execution only the rules associated with transitions from the current state of the tester EFSM are evaluated to select the next transition with the highest gain. Thus, the number of rules that need to be evaluated at each step is relatively small. The decision rules are constructed taking into account the reachability of all trap-equipped transitions from a given state and the length of the paths to them. The current value (visited or not) of each trap is also taken into account. The decision rules are derived by performing reachability analysis from the current state to all trap-equipped transitions by constructing the shortest path trees. The gain functions which form the terms of the decision rules are derived from the shortest path trees by simple rewrite rules. The resulting tester drives the IUT from one state to the next by generating inputs and by observing its outputs. When generating the next input for the IUT the tester takes into account which traps have already been visited. The execution of decision rules at the time of test execution is significantly faster than finding an efficient test path by state-space exploration algorithms, but nevertheless produces a test sequence which, lengthwise, is close to optimal.

The main contribution of PAPER 3 is an algorithm for constructing a tester which selects a suboptimal test path from trap to trap on the fly by finding the shortest path to the next unvisited trap iteratively. No costly model exploration

and path finding operations are needed online. In the context of the experiments presented in the paper, the reactive planning tester is more efficient at runtime than random choice and anti-ant algorithms. The planning feature of the reactive planner results in significantly shorter test sequence lengths on average. The reactive planner outperforms *anti-ant* algorithms by more than the order of magnitude in cases where a more directed search is presumed i.e. the test purpose covers the model partially.

4.2 Case study-based performance evaluation of reactive planning tester

Contemporary on-the-fly model-based test generators which primarily focus on planning strategies which are computationally cheap but far from optimal cover just a fraction of the spectrum of test control strategies. Typical examples of these are simple random choice and anti-ant. Exhaustive planning during online testing is not feasible because of the lack of available computational resources to meet the required response time of the tester and because of the low scalability of the methods in regard to the size of the model. The reactive planning tester presented in PAPER 3 is targeted to fill the gap between these two extremes. The key idea of RPT lies in offline learning of the IUT model to prepare the data for efficient online reactive planning.

In PAPER 3, experiments on a small model are conducted to compare RPT efficiency against the random choice and *anti-ant* test selection methods. Experiments with larger models were not possible at the time because of the lack of tool support. The MOTES test generator for RPT synthesis (see Chapter 6) was available by the time of writing PAPER 4, making reporting on experiments with bigger models possible. The paper **"Case study based performance evaluation of reactive planning tester" [PAPER 4]** was written by Kullo Raiend, Jüri Vain, Marko Kääramees and the author of this thesis. It was presented at the 2nd Workshop on Model-Based Testing in Practice on 23 June 2009 in Enschede, the Netherlands. The paper confirms that in industrial-scale case studies the RPT significantly outperforms the random choice and *anti-ant* test selection methods. Based on a case study of a feeder box controller of a city lighting system, we demonstrated that tuning the planning horizon of RPT allows you to reach close to optimal tester behaviour (in terms of test sequence length) with computationally feasible expenses. The model in the case study is a strongly connected state machine with 31 states and 78 transitions.

The experiments were performed using two different coverage criteria: *all transitions* and a *single selected transition*. Different RPT planning horizons (of between 0 and 20 steps) were used in the experiments, which showed that increasing the planning horizon decreases the average length of the test sequences exponentially while the planning time of the next step increases by not more than the length of the planning horizon to the power of 2. In the experiments, the planning time was within the range of 13-22 msec. RPT significantly outperforms the *anti-ant* and random choice methods. On average,

the RPT with the maximum planning horizon resulted in test sequences which were more than 100 times shorter than the *anti-ant* and random choice methods.

The main contribution of PAPER 4 lies in the fact that it proves the feasibility of RPT in industrial-scale case study, demonstrating its advantages over the random choice and *anti-ant* methods, and showing the test sequence length and planning time dependencies on the RPT planning horizon. Generalisation of the results of performance analyses will require additional experiments with different case studies in future.

5 REQUIREMENT-DRIVEN TESTING

The fifth paper is entitled **"Requirement-driven model-based testing of the IP Multimedia Subsystem" [PAPER 5]**. The authors of the paper were Juhan Ernits, Marko Kääramees, Kullo Raiend, and the author of this thesis. It was presented at the 2008 International Biennial Baltic Electronics Conference (BEC2008) in Tallinn, Estonia in October 2008.

Software testing processes in industry are usually requirement-driven. This means that the testing should verify if a particular software requirement defined in requirement specifications and other reference specifications is implemented correctly. As the word '*requirement*' has variety of meanings, the paper uses the definition given in [UL06]: A *requirement* is a testable statement of some functionality that the product must have. In requirement-driven testing the targeted test coverage is defined by covering the requirements. In the previous papers, structural test coverage criteria were applied to test generation. There is a significant issue with structural test coverage criteria for industrial users. Test criteria – like *all transition*s, *all k-switches* and so on – may sound like excellent coverage criteria for researchers, but industrial users ask which of their requirements such generated tests cover. They would like to track the generated tests and test logs back to the requirement level. They would like to know if all of the requirements are tested and which generated test cases cover particular requirements. The requirement-driven approach in testing is widely employed throughout the industry, using manual and script-based testing. Industry users ask the same of MBT.

The requirement-driven approach in MBT should be supported as early as the modelling phase by using suitable modelling formalisms. The requirements should be able to be modelled in a way that easily allows parts of the model to be selected as test coverage items belonging to the particular requirement and a test case for this particular requirement to be generated.

The IP Multimedia Subsystem (IMS) [IMS] is an architectural framework for the delivery of Internet protocol multimedia services to mobile users. The functionality of the system is specified to allow interoperability between equipment from different manufacturers. IMS was originally designed by the wireless standards body of the 3rd Generation Partnership Project (3GPP) [3GPP] and forms part of the vision for the evolution of mobile networks beyond GSM. The European Telecommunication Standards Institute (ETSI) [ETSI] has developed standardised IMS interoperability tests to verify the interoperability of IMS networks from different manufacturers. IMS as the IUT and IMS interoperability tests as the test specification were selected for a case study for the paper because IMS represents a typical telecommunications system that is specified as a collection of requirements and the IMS interoperability test specification specifies the subset of IMS requirements that should be tested to ensure interoperability. IMS interoperability tests represent typical requirement-driven testing used in the industry.

In the paper the authors examine how to build the IMS models in a way that provides an easy means of attaching requirements i.e. certain paragraphs in the specification to a specific part of the model. The IMS interoperability test specifications feature long lists of detailed requirements that must be satisfied by different components that comprise the IMS. The goal of the paper was to provide techniques for modelling the IUT and applying the MBT with the objective of achieving full requirement coverage.

The paper presents the idea of modelling a fragment of the IMS protocol from the point of view of the requirements. The paper provides solutions using two different techniques for modelling the IMS. Both techniques serve the same purpose and are driven by the same basic idea. A network of the Uppaal automata [UPPA] and NModel [JVC+08] model program are used to model the IMS according to the specifications. The goal was to associate the model components as well as possible with the requirements in the specification. This allows the components of the model to be traced back to the paragraphs in the requirement specifications. The modelling was carried out in two steps: modelling the infrastructure of the IMS, which is involved in all requirements (the general part of the model); and modelling the specific requirements or features as separate components of the model, which are connected to the general model using model composition.

Uppaal was used to build the EFSM models of the IUT because it supports the composition of automata and has a model checking backend for the analysis of the model and for the generation of witness traces (corresponding to test sequences). The composition of the automata was a valuable feature to utilise because it allows different features of the system to be modelled as small feature automata and for them to be combined into a system model. NModel was used because it supports the composition of small model programmes that model certain features of the system and construct the system model feature by feature iteratively. An important aspect of both tools and techniques is their models composition support. These tools and techniques were not compared in the paper, but it does demonstrate how the requirement-driven approach can be applied to the building of the model of the IUT using either composition of automata or model program features.

The paper also demonstrates that it is possible to model a IUT consisting of separate requirements presented as separate automata (Uppaal) or features of model programs (NModel) by using model composition. Requirement automata or features can be composed using a general infrastructure model as the basis of the model. In the IMS case the infrastructure model was derived from the IMS infrastructure requirements and specific requirements were modelled as separate model components in the underlying infrastructure. Using such an approach the different features of the model are decoupled from one another. The test engineer can easily define the coverage of the tests by selecting the component models representing the different requirements and generate corresponding test cases from the composition of the infrastructure model and the appropriate requirement models to cover the selected requirements. In this way it is easy to

generate tests that only cover selected requirements of the model and the test engineer can trace the requirements from the requirement specification through the model to the generated test cases and test logs, and vice versa.

PART II: PRACTICAL RESULTS AND CASE STUDIES

6 MOTES – TEST GENERATOR

6.1 Overview

The results of the theoretical contribution of the thesis presented in Chapters 3 and 4 were implemented in the model-based test tool MOTES.

MOTES is a tool which generates TTCN-3 tests from the EFSM model of the IUT. It accepts EFSM models of the IUT prepared using third party UML CASE tools. Currently it supports models exported from Poseidon for UML [PUML] and Artisan Studio [ARTI] CASE tools. Context variables and the interfaces of the IUT should be defined in TTCN-3. Test data instances and templates are also defined in TTCN-3. In the future it will also be possible to define the data with CASE tools, using class and object diagrams. Tests produced by MOTES in TTCN-3 language can be executed by any TTCN-3 test tool against the IUT (see Figure 2).

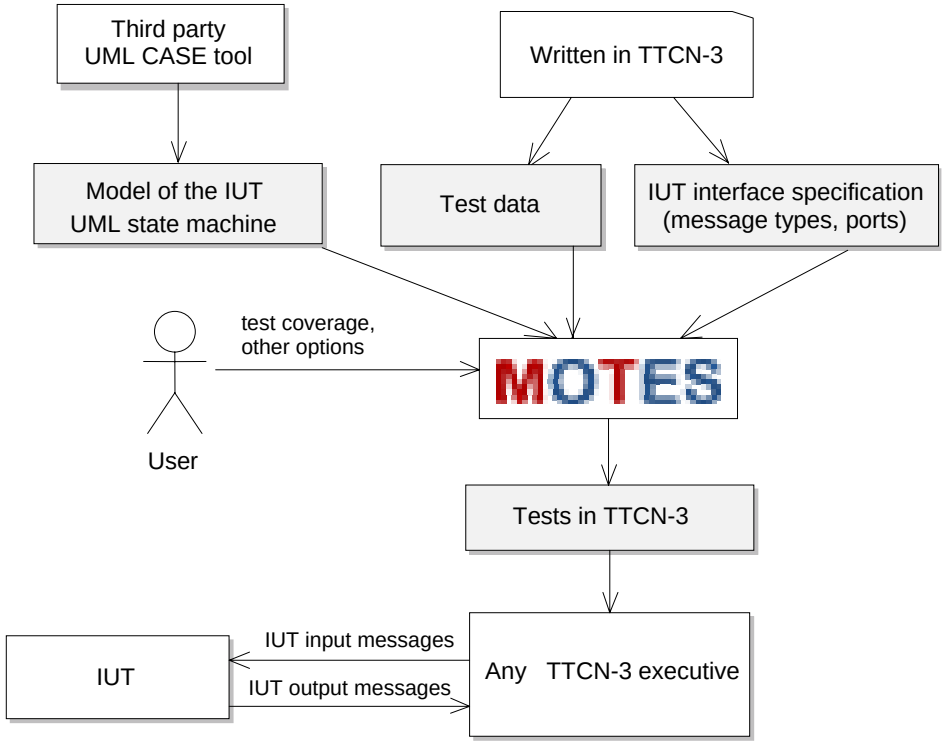

Figure 2: MOTES context

In order to generate tests from deterministic EFSM, the iterative model checking based test generator is used in MOTES. The generator is based on the method described in Section 3.2. It produces an abstract test sequence (a

sequence of transitions) driven by the user-defined test coverage. The abstract test sequence is converted to TTCN-3 test cases as described in PAPER 2. The resulting TTCN-3 test cases can be executed against the IUT.

To generate tests from the nondeterministic model, the abstract RPT is generated using the method described in Section 4.1. The abstract RPT is converted into TTCN-3 in order to execute it against the IUT. RPT implementation in TTCN-3 is an on-the-fly test generator that autonomously finds the test sequence that satisfies the user-defined test coverage of the nondeterministic IUT.

The test generation workflow and the most important features of MOTES are explained in the sections below using a simple light switch example.

6.2 Description of light switch example

The implementation under test in the example is a `Light Switch` which turns a light on or off at the user's request (Figure 3).

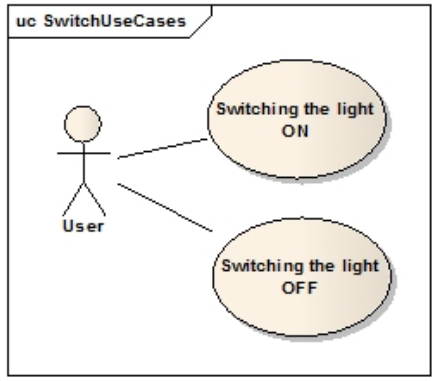

Figure 3: `Light Switch` use cases

The `Light Switch` is typically driven by a human (the environment) and must implement the following functional requirements:

`Requirement 1`: The light is switched on at the request of the environment.

`Requirement 2`: The light is switched off at the request of the environment.

`Requirement 3`: If the light is already on or off, requesting the same operation (turning the light on or off, respectively) does not change the system state.

Let us suppose that the `Light Switch` receives messages from the `Environment`, which is represented by an external technical system or human

user. All activities are initiated by the environment. After receiving input from the environment, the `Light Switch` reacts by turning the light on or off. Thus the state of the IUT (light on/off) responds to the `Environment` after the request is fulfilled.

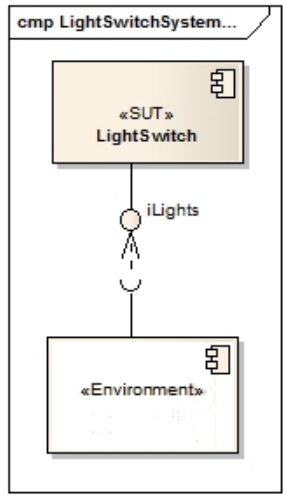

Figure 4: `Light Switch` context

The `Light Switch` is controllable by the environment through the `iLights` interface (Figure 4). The interface defines the following messages between the `Light Switch` and the `Environment`:

`TurnOn`: request from the `Environment` to the `Light Switch` to turn the light `on`

`TurnOff`: request from the `Environment` to the `Light Switch` to turn the light `off`

`LightIsOn`: response from the `Light Switch` to the `Environment` indicating that the light is `on`

`LightIsOff`: response from the `Light Switch` to the `Environment` indicating that the light is `off`

The messages `TurnOn` and `TurnOff` are sent over the `iLights` interface by the `Environment`. After handling the messages, the `Light Switch` responds in regard to its state by using the messages `LightIsOn` or `LightIsOff` (see interaction diagrams in Figure 5).

Let us assume that the tests should be generated from the model of the `Light Switch` according to Table 1.

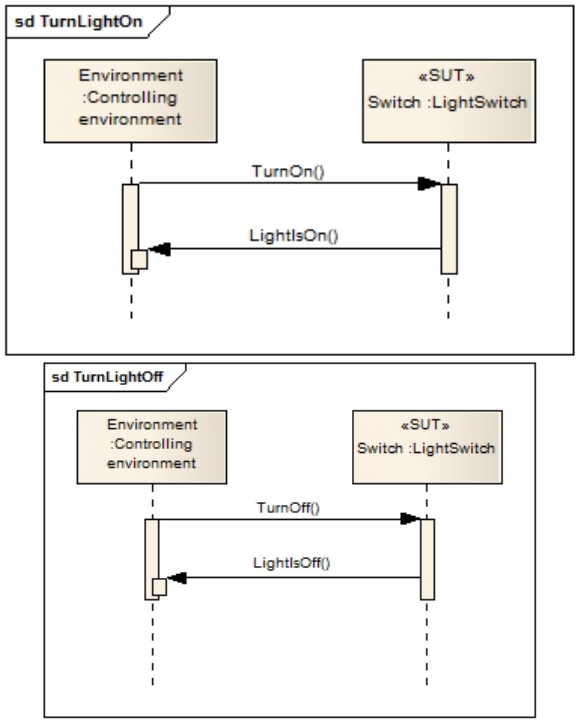

Figure 5: Interaction diagrams

Table 1: Test purposes

Test purpose 1	Test that the `Light Switch` turns the light from `off` to `on`
Reference	`Requirement 1`
Preconditions	The light is `off`
Input	The request message `TurnOn` is sent by the `Environment` to the `Light Switch`
Expected results	The response message `LightIsOn` is sent by the IUT to the `Environment`

Test purpose 2	Test that the `Light Switch` turns the light from `on` to `off`
Reference	`Requirement 2`
Preconditions	The light is `on`
Input	The request message `TurnOff` is sent by the `Environment` to the `Light Switch`
Expected results	The response message `LightIsOff` is sent by the IUT to the `Environment`

Test purpose 3	Test that the light remains `off` when the `Light Switch` is commanded to switch the light from `off` to `off`
Reference	`Requirement 3`
Preconditions	The light is `off`
Input	The request message `TurnOff` is sent by the `Environment` to the `Light Switch`
Expected results	The response message `LightIsOff` is sent by the IUT to the `Environment`

Test purpose 4	Test that the light remains `on` when the `Light Switch` is commanded to switch the light from `on` to `on`
Reference	`Requirement 3`
Preconditions	The light is `on`
Input	The request message `TurnOn` is sent by the `Environment` to the `Light Switch`
Expected results	The response message `LightIsOn` is sent by the IUT to the `Environment`

6.3 Creating the IUT model

The model of the IUT for MOTES consists of the EFSM, descriptions of the context variables and interface specifications.

EFSM is used to model the behaviour of the IUT. MOTES does not have its own editor for EFSMs: third party UML CASE tools are used to create them. At the time of writing, MOTES is able to import state machines from Poseidon for UML [PUML] and Artisan Studio [ARTI] CASE tools. In the future it is expected to be able to import state machines from any UML CASE tool that supports exports in XMI 2.1 format. EFSMs for MOTES are drawn as flat UML state models without parallel and hierarchical states. UML state machines do not have a formally specified language for the presentation of guard conditions, input events and actions on transitions. Because it is only possible to generate tests from the formal system model, a formal transition language (MTL) was developed for MOTES which is used in transitions of the EFSM. MTL is presented in Appendix A.

Example: Figure 6 below presents the EFSM model of the IUT in the `Light Switch` example.

- `LightSwitch_Off` and `LightSwitch_On` model light states.
- It is assumed that the initial state of the `Light Switch` is `off`.
- Transitions `T1, T2, T3, T4, T7` and `T8` model transitions between states as described in Table 2.

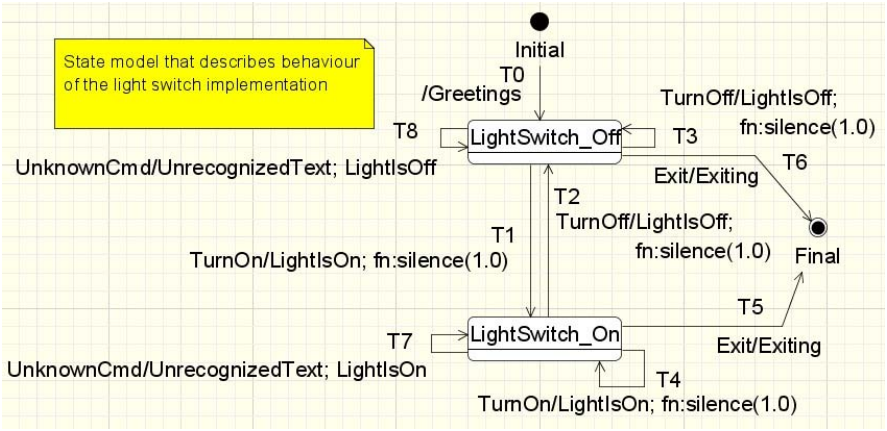

Figure 6: `Light Switch` model drawn in Poseidon

Table 2: Table of transitions

Transition	Starting state	Trigger (command)	Effect (output)	Next state
T1	LightSwitch_Off	TurnOn	LightIsOn, Silence	LightSwitch_On
T2	LightSwitch_On	TurnOff	LightIsOff, silence	LightSwitch_Off
T3	LightSwitch_Off	TurnOff	LightIsOff, silence	LightSwitch_Off
T4	LightSwitch_On	TurnOn	LightIsOn, Silence	LightSwitch_On
T7	LightSwitch_On	UnknownCmd	LightIsOn, Silence	LightSwitch_On
T8	LightSwitch_Off	UnknownCmd	LightIsOff, Silence	LightSwitch_Off

- All of the use cases described above are covered by these transitions.
- The function `silence(float Duration_sec)` helps visualise the test execution process by adding a delay between tests.
- The model has no transition guards or context variables.

Context variables are the variables that are used in EFSM to store state information in addition to the control states of the EFSM.

Interface specifications define the input/output ports used in the model. The ports define the IUT interface towards its environment. Port definitions define the data type of the messages that the ports accept.

It would be possible to define context variables and ports in UML using classes, and they would be instantiated using objects derived from these classes. In MOTES a different approach is taken: the aim of MOTES is to generate TTCN-3 test cases, and to streamline the test generation process the context variables and ports are defined directly in TTCN-3 files in the current version.

Example: The TTCN-3 `TestConfiguration` module in Figure 7 describes the TTCN-3 test component `Tester`, its interface for communicating with the IUT `iLights` and the message types (`Command` and `Output`) accepted by the port. In addition, it defines the function `silence(float duration_sec)`.

```
module TestConfiguration {
	// Test configuration definitions

	// Import message type definitions
	import from TestData all;
	// Define tester ports
	type port iLightsPortType message {
		out Command;
		in Output
		};

	// Define tester with a message port iLights
	type component Tester {
		port iLightsPortType iLights;
		};

	//Define functions
	//The function silence is introduced in the state model
	for
	//visualizing of lamp switching at test running

	// Silence (no output)
	function silence (in float duration) runs on Tester {
		timer timerSilence;
		timerSilence.start(duration);
		alt {
			[] timerSilence.timeout {
				//OK
				}
			[] any port.receive {
				log("A message was received during
			required
				    silence time!");
				setverdict (inconc);
				stop;
				}
			}
		};
	}
```

Figure 7: `TestConfiguration` module in TTCN-3

6.4 Preparing test data

Test control-flow is generated by MOTES automatically from the IUT model. The test data used by the tests should be prepared manually. Like context

variables and interface definitions, the test data are also defined and instantiated directly in TTCN-3.

Example: Figure 8 below includes the test data definitions for the `Light Switch` example. The TTCN-3 module `TestData` defines the possible messages (`Command`) sent to the IUT, such as `TurnOn` and `TurnOff`, and possible response messages (`Output`) from SUT, such as `LightIsOn` and `LightIsOff`.

```
module TestData {

        // Message type definitions
        type charstring Command; // Light switch input command type
        type charstring Output; // Light switch output type

        // Message instance definitions

        // Light switch commands
        template Command TurnOn := "turnOn"; // Command for turning the
                                             // lights on
        template Command TurnOff := "turnOff"; // Command for turning the
                                               // lights off
        template Command Exit := "exit"; // Command for exiting the
                                         // implementation
        template Command UnknownCmd := "xyz"; // An example of unknown
                                           // command for the implementation

        //Light switch outputs
        //Expected output lightIsOn
        template Output LightIsOn := "lightIsOn";
        //Expected output lightIsOff
        template Output LightIsOff := "lightIsOff";
        //When the implementation starts the expected output contains
        // string 'ready'
        template Output Greetings := pattern "*ready*";
        //Response to the unknown command contains string 'Unrecognized'
        template Output UnrecognizedText := pattern "*Unrecognized*";
        //Response to the exiting command contains string 'Exiting'
        template Output Exiting := pattern "*Exiting*";
        }
```

Figure 8: `TestData` module in TTCN-3

6.5 Importing the model and test data into MOTES

The test project browser view in MOTES includes a pre-defined structure of artefact folders for the maintaining of a clear structure for the test generation project (Figure 9).

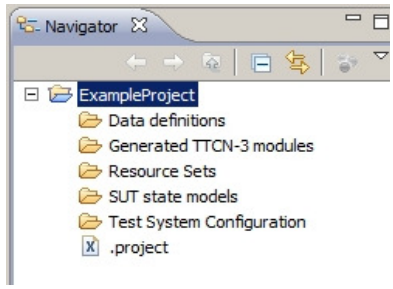

Figure 9: Test project browser view in MOTES

Those predefined folders are used to organise the input and output artefacts of the test project:

- *Data definitions* – for storing data definition TTCN-3 files
- *Generated TTCN-3 modules* – for storing the TTCN-3 files generated by MOTES
- *Resource Sets* – for storing resource files that link together the relevant input artefacts for a particular test generation task or purpose (IUT model, data definitions, configurations and context variable definitions). Different models and different test data instances are needed for different test generation purposes. The term '*resource set*' is used in MOTES for such collections of data. The concept of the *resource set* corresponds to a *project* in software development environments like Borland Delphi and Microsoft Visual Studio.
- *IUT state models* – for storing the state models of the IUT (in XMI format)
- *Test System Configuration* – for storing test system configuration definition TTCN-3 files

The user must import EFSM model files exported from the UML tool, TTCN-3 files with context variables, interface definitions and test data to the relevant test project folders. Next, the user must create a new *resource set* to link together the relevant input resources using a resource set editor.

Example: Figure 10 illustrates the resource set editor using the resources of the `Light Switch` example displayed.

6.6 Defining test purpose (coverage/goal)

MOTES uses model structural test coverage elements to define test coverage. The following coverage criteria are available:

- *selected elements (states/transitions*)
- *all transitions*
- *all n-transition sequences where n>=2*

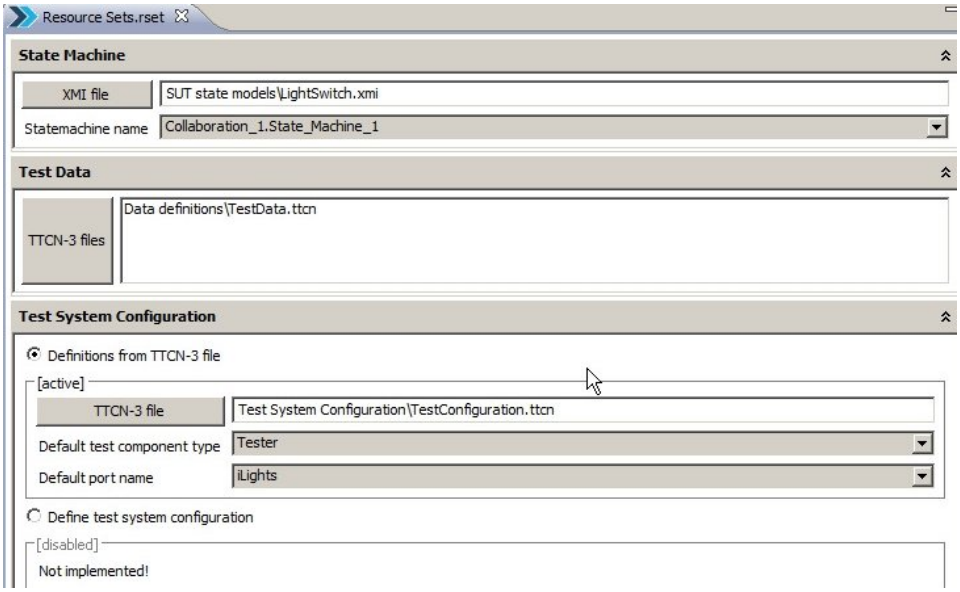

Figure 10: Resource set editor in MOTES

Selected elements (*selected states* and *transitions*) used to define test coverage can be selected from the list of EFSM transitions and states. It is possible to create ordered sets of coverage items and to define how many times each coverage item or a subset of the item should be covered. Complex test scenarios can be created using ordered sets of coverage items. For example, it is possible to define that the test generator should find a test that first covers transition `T1`, then covers transitions `T2` and `T3` three times and finally covers transition `T1` once again.

All transitions test coverage defines that the test generator should find a test that covers all of the transitions of the EFSM at least once.

All n-transition sequences is a test coverage criterion that allows long and exhaustive tests that cover all subsequent transition sequences of *n* transitions to be created. In MOTES *n* can be 2 or 3.

Example: Let us use selected elements coverage in the `Light Switch` example to define a complex coverage scenario. After selecting the *Selected Elements* option, the dialogue box in Figure 11 opens. The left-hand pane includes a list of all states and transitions in the EFSM. The right-hand pane is a tree-view editor that allows the selected elements coverage to be constructed for the model. The available elements can be dragged from the left-hand pane to the suitable place in the right-hand pane.

The tree-view *selected elements* coverage editor has a root node that can be either *sets* or *lists*. *Sets* and *lists* contain *elements*, and *elements* can also be *sets* or *lists*. All *elements* in a *set* and *list* must be covered by the generated tests. In

the case of *sets*, the order of *elements* is not important to the user and the test generator decides the order itself. In the case of *lists,* the order of covering the *elements* is defined by the order of the *elements* in the *list*. The number in front of an element defines how many times the particular element must be covered by the generated test case.

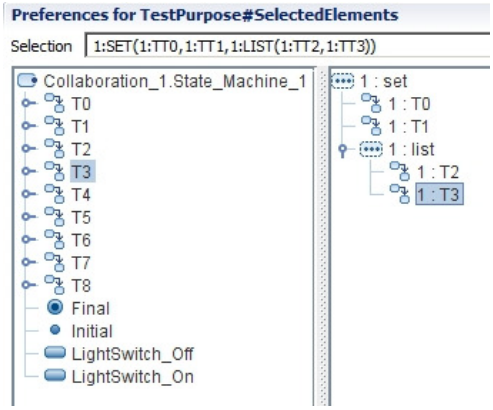

Figure 11: Selected elements coverage editor in MOTES

6.7 Choosing the test generation engine

MOTES includes two test generation engines:

- model checking engine
- reactive planning tester engine

The test generation preferences view in MOTES is presented in Figure 12. The model checking engine can be selected using the radio button `Generate test sequence` and the reactive planning tester engine can be selected using the radio button `Generate reactive planning tester`.

The model checking engine implements the test generation method presented in PAPER 1, PAPER 2, and Section 3.2. The engine is used to generate test cases for the deterministic IUT. The engine produces test cases that deterministically control the execution of the deterministic IUT through the sequence of transitions on the IUT model. The test cases produced cannot test nondeterministic IUT because they always expect deterministic IUT behaviour in response to the generated stimuli. The engine utilises the Uppaal Cora model checker [UpCo] in finding test sequences. The most important configuration parameter for the model checking engine is `use iterative mode`. If the parameter is checked, the model checker works in iterative mode according to the algorithm described in Section 3.2. With each iteration the shortest path to the nearest unvisited trap is found. This is a greedy algorithm which returns a

suboptimal test sequence. Iterative model checking makes it possible to find test sequences from state models which are not strongly connected in the case of limited computer memory. If `use iterative mode` is not checked, the model checker attempts to find the whole test sequence with the minimum length required to reach the test purpose.

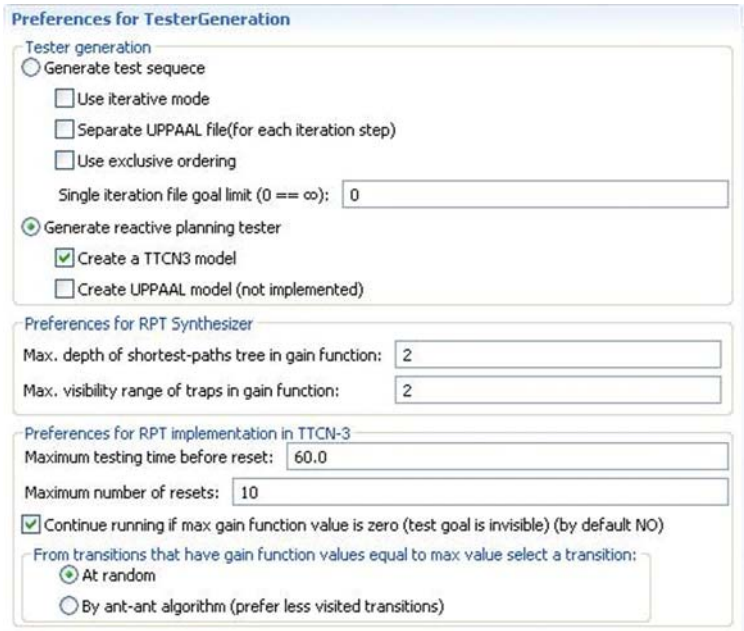

Figure 12: MOTES test generation preferences

The reactive planning tester engine implements the test generation method presented in PAPER 3. It does not generate predefined test cases, but rather a reactive planning tester which, in the online phase, generates test stimuli on the fly. The reactive planning tester can be used to generate tests for deterministic or nondeterministic IUT. The reactive planning tester engine uses the reachability analysis in synthesising the reactive planning tester, which is generated in TTCN-3 and can be executed by any TTCN-3 executive in the same way as TTCN-3 test cases produced by the model checking engine. The intelligence of the reactive planning tester is encoded in gain functions that characterise the potential choices of test stimuli. In the online testing phase the reactive planning tester must decide which move from those possible to make. For each possible move it calculates the gain function and makes the move which promises maximum gain. The gain function gives a higher value if the next move will lead to a larger amount of unvisited coverage items faster. The reactive planning tester engine can be configured using the following configuration parameters (see also Figure 12):

- `max depth of shortest-paths tree in gain function`
  The parameter defines the look-ahead planning horizon. It also defines the maximum path length of the shortest path starting from the root of the shortest-path tree. In other words, it defines the length of the planning horizon in transitions where the reactive planning tester looks ahead. The gain functions are calculated taking into account the trap amount and location in the tree. Planning within the horizon is precise.
- `max visibility range of traps in gain function`
  The visibility parameter defines the visibility horizon of the traps. Within the visibility range only the reachability of the traps is important, not the exact location, as in the previous parameter. The visibility parameter can also be bigger than the planning horizon. In the event that the trap is outside of the planning horizon but within the visibility horizon, the planning algorithm knows that the trap is reachable from the given shortest-path tree but does not know exactly how far away the trap is.
- `max testing time before reset`
  This parameter defines the maximum testing time for a situation where some of the traps have yet to be visited. In nondeterministic models, restarting the IUT may increase the chance of visiting unvisited traps after reset.
- `max number of resets`
  This parameter defines the maximum number of IUT resets that are performed in trying to cover traps that have yet to be covered.
- `continue running if max gain function value is zero :from transitions that have gain function values equal to max value`
  In the case of equal gain values, this parameter allows you to select the next transition from the alternatives either randomly or with an *anti-ant* algorithm.

6.8 Executing the test generation engine

Once the steps above are completed, the user presses the “G” button and the test generator does the rest. The TTCN-3 test cases or reactive planning tester TTCN-3 code are generated under the current *resource set*. The generated TTCN-3 files can be imported to any TTCN-3 test tool and run against the IUT.

7 CASE STUDIES

7.1 Introduction

This chapter demonstrates the applicability of the MOTES test generator and the feasibility of the underlying methods and techniques in industrial-scale case studies. It presents case studies from different branches of the industry: telecommunications and telematics. Whilst evaluating the overall applicability of the technology, the case studies also have their own slightly different objectives to evaluate.

The following two case studies are demonstrated:

- Sofia-SIP stack testing
- feeder box control unit testing

The case study sections include the following sub-sections:

- overview of IUT
- objectives of case study
- system adapter
- modelling the IUT
- test generation and execution
- objectives evaluation

Finally, the evaluation results of the case studies are summarised and issues for further study are outlined.

Both case studies were performed using the same workflow and set of tools (Figure 13):

- Poseidon for UML CASE tool by Gentleware [PUML] was used for the state machine modelling.
- MOTES was used to generate TTCN-3 tests from the model of the IUT.
- MessageMagic by Elvior [ELMM] was used to execute the generated TTCN-3 tests against the IUT.
- A case study-specific system adapter designed to connect IUT and MessageMagic was developed for each case study.

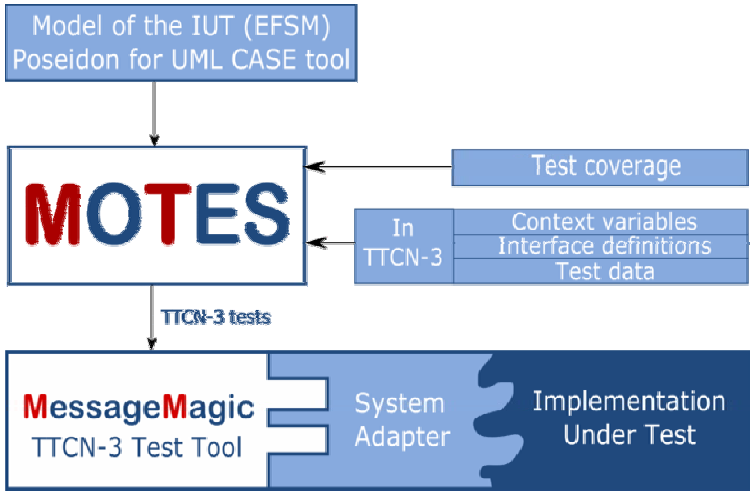

Figure 13: Common architecture of test environments in case studies

7.2 Testing of Sofia-SIP stack

7.2.1 Overview of IUT

The **Session Initiation Protocol** (**SIP**) is a signalling protocol widely used for setting up and closing down multimedia communication sessions such as voice and video calls over the Internet. Other feasible application examples include video conferencing, streaming multimedia distribution, instant messaging, presence information and online games. The protocol can be used for creating, modifying and terminating two-party (unicast) or multiparty (multicast) sessions consisting of one or more media streams. The modification can involve changing addresses or ports, inviting more participants, adding or deleting media streams and more.

SIP was originally designed by Henning Schulzrinne and Mark Handley in 1996. The latest version of the specification is RFC 3261[SIP] from the Internet Engineering Task Force (IETF) SIP Working Group. In November 2000, SIP was accepted as a 3GPP signalling protocol and permanent element of the IMS architecture for IP-based streaming multimedia services in cellular systems.

SIP User Agents (UAs) are the end-user devices used for creating and managing SIP sessions. A SIP UA has two main components. The User Agent Client (UAC) sends messages and answers with SIP responses. The User Agent Server (UAS) responds to SIP requests sent by the peer. SIP UAs may work in point-to-point mode. Typical implementations of a UA are SIP soft-phones, SIP hard-phones, and SIP-enabled analogue telephone adapters.

Sofia-SIP [SOFI] is an open-source SIP User-Agent library that complies with the IETF RFC3261 specification [SIP]. Sofia-SIP can be used as a building block for SIP client software for uses such as VoIP, instant messaging and other real-time and person-to-person communication services. The primary target

platform for Sofia-SIP is GNU/Linux, although it also targets the embedded world. Sofia-SIP is based on a SIP stack originally developed at the Nokia Research Centre. The Sofia-SIP library is written in C. The Sofia-SIP stack has an API interface towards applications that are using the SIP stack and a network interface between two SIP stacks.

7.2.2 Objectives of case study

The conformance of the Sofia-SIP protocol stack to the SIP specification [SIP] was tested in the case study. The target was to generate tests according to the test purposes specified in the ETSI conformance test specification for SIP applications [ConfS]. SIP functionality as a whole was not tested in the case study, but only the SIP UAC part using the UDP transport.

The following aspects were evaluated in the case study:

- **Overall feasibility of MOTES technology for conformance testing of industrial-scale telecommunications applications.** Sofia-SIP is a typical telecommunications protocol application that is far from being too trivial. The specification describing IUT functionality in the scope of the case study is around 80 pages in length [SIP]. Test specifications for the same functionality include around 15 pages [ConfS]. Implementation of the Sofia-SIP functionality in the scope of the case study included around 500 lines of C++ code [SOFC].

- **Adaptability of MOTES technology for the requirement-driven testing process.** Test generation in MOTES is based on model structural coverage criteria. This is a very different approach compared to the requirement-driven testing process used widely in the industry. In the state model of the IUT the requirements are usually spread throughout the model and can partly overlap. Telecommunications systems testing specified by ETSI processes is strictly requirement-based. Test purposes in the test specifications define which features of the IUT should be tested and how they should be tested [ETPR].
    - The possibility of mapping the requirement-based approach to the test generation method using structural coverage of the model was evaluated.
    - The testing power of the generated test cases was evaluated compared to the test purposes specified in the test specification [ConfS].

- **Feasibility of the EFSM for modelling industrial-scale telecommunications applications.** The modelling power, size, complexity and readability of the resulting EFSM were evaluated.

- **Feasibility of the iterative model checking based test generation method.** The applicability of the method for generating test cases from the deterministic model was evaluated. Whether the case study application could be modelled using a deterministic model was also evaluated, as was whether the use of a deterministic model could cause any issues.

- **Bug detection ability of generated tests.** The Sofia-SIP stack library has been available to developers for some time. The quality of Sofia-SIP implementation is otherwise unknown. Therefore it was interesting to observe whether the case study would reveal any real bugs in its implementation.

7.2.3 System adapter

The test environment of Sofia-SIP contains Sofia-SIP implementation (IUT), a system adapter and MessageMagic for executing TTCN-3 test cases. The system adapter connects the IUT to the MessageMagic test tool.

Sofia-SIP has two interfaces: an API providing services for the application layer (the functions `nua_create`, `nua_destroy`, `nua_invite`, `nua_bye` and a `callback_function`) and an interface towards the network communicating with other SIP peers on the Internet (SIP messages over the transport layer). In the test environment, TTCN-3 test cases executed by MessageMagic simulate both – an application layer over API and other SIP peers over the network interface. The system adapter is connected to MessageMagic through a TTCN-3 standardised TRI interface (Figure 14).

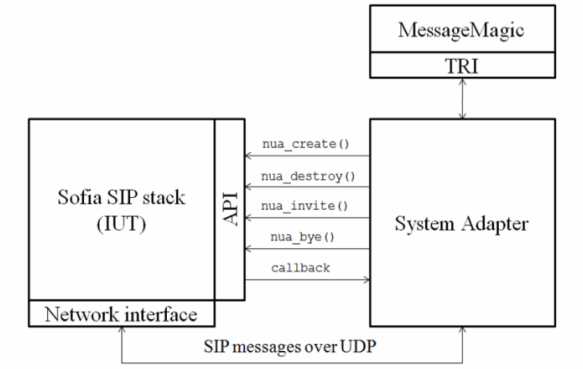

Figure 14: Sofia-SIP test environment

7.2.4 Modelling the IUT

The model was built using information from two documents – the SIP specification [SIP] and the ETSI conformance test specification for SIP

[ConfS]. The target was to create tests for the SIP UAC with UDP transport satisfying the relevant test purposes in the conformance test specification.

Discussion

The IUT model is always built to test particular functionalities and for particular testing needs. It is very important to understand in MBT that the model of the IUT should be created on the right abstraction level. Only those features that need to be tested should be present in the model, and they must be modelled on the same abstraction level that the tests need to verify them. In this way the model created for test purposes remains readable and maintainable and the tests generated from the model test the features on the right abstraction level.

It is often claimed that the more detail in which you model your system the better the tests with better test coverage you will generate. In principle this is true, but the trade-off is model complexity. Models which are too complex are often poorly readable, hard to maintain and may cause performance problems for test generation tools and methods. A model by its very definition is an abstraction of reality designed to keep things comprehensible. A graphical model that presents the IUT details on the level of very high granularity is a graphical programming language and therefore looses many benefits that modelling should give to users.

Test purposes in ETSI conformance test specification for SIP

The scope of the case study includes 88 test purposes in the SIP conformance specification [ConfS].

An example of a typical SIP test purpose is as follows:

`TPId: SIP_CC_OE_CR_V_008:`
Ensure that the IUT, once a dialog has been established, having sent a BYE request, on receipt of a Success (200 OK) response considers the session and the dialog terminated.

This test purpose, as with all other test purposes in [ConfS], is highly abstract. Test purposes miss many important details for building tests that meet test purposes. The test purposes must be analysed and interpreted by a test engineer who knows the SIP specification in order to put them into a formal language for modelling or test case implementation. In the case study, such analysis was completed before the SIP UAC model was constructed.

Modelling approach in case study

The model was built according to test purposes. Only SIP UAC behaviour that was required to be tested by the test purposes was modelled. During the

modelling the SIP specification [SIP] was consulted to translate the abstract test purposes into formal modelling language.

The resulting deterministic EFSM of the SIP UAC includes the behaviour that was required to be tested by the given test purposes. The model has 19 states and 52 transitions. A fragment of the model is presented in Figure 15. The model has 24 context variables. It is hard to measure the time spent building the model. In the case study the MBT process was iterative, consisting of the study of the SIP specifications, updating the model with certain aspects of SIP functionality, updating the system adapter, generating test cases and running them against the IUT. Usually after each update the test execution failed, and often the reasons were incorrect model or incorrect test data. The total time spent on the case study was 50 man days. This includes everything from analysis of the SIP specification to the situation where the test cases for all planned test purposes were generated and executed.

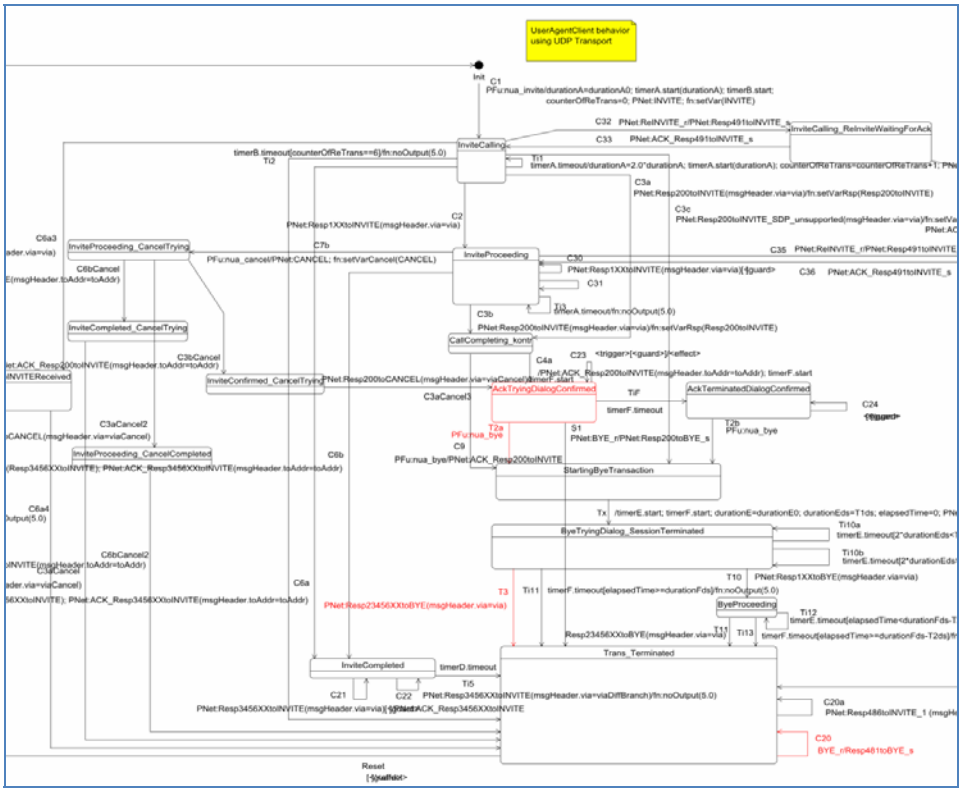

Figure 15: Fragment of Sofia-SIP UAC model

7.2.5 Test generation and execution

Although the model of SIP UAC includes the behaviour for testing all of the given test purposes, the model does not have information as to which parts of the model implement particular test purposes. It is not possible to tell the MOTES test generator to *take the SIP UAC model and generate a test case that covers the test purpose SIP_CC_OE_CR_V_008*, for example. MOTES uses model structural coverage defined as set of coverage items (selected states and transitions). Therefore, the test coverage specified by the test purposes should be transformed to the form of the model structural coverage items.

It was possible to define test coverage using model structural coverage items for 78 test purposes out of 88 (87%) defined in the SIP conformance specification [ConfS] and to generate test cases from them. For the remaining 10 test purposes, the equivalent test coverage was not possible to define using the structural coverage items and tests were not generated. For example, test purposes that require modelling of parallel transactions were not possible because MOTES lacks the feature of modelling parallel transactions and generating tests for them. Therefore, it was not possible to generate test cases for parallel INVITE transactions.

For generating tests according to 78 test purposes, 35 different test goals were created. Each of the test goals was built to generate tests for one or several test purposes. The test cases generated for the 78 test purposes amounted to 17,500 lines of TTCN-3 code.

In addition to the test purposes of the SIP conformance specification, the tests were generated from the same model using the *all transitions* test purpose. All transitions of the model are able to be visited with a test sequence of 120 steps. The *all transitions* test purpose resulted in 2900 TTCN-3 lines of code (LOC). In addition to the generated TTCN-3 code, around 1690 TTCN-3 lines of code were written manually:

- data types – 780 LOC
- templates – 800 LOC
- test system configuration and TTCN-3 custom functions – 110 LOC

The generated test cases were executed using the MessageMagic TTCN-3 test tool [ELMM] against the Sofia-SIP stack. The test environment is presented in Figure 14. The execution of the test cases discovered 5 bugs in the Sofia-SIP stack library.

7.2.6 Test coverage analysis

Test coverage analysis was carried out in order to determine the quality (coverage) of the generated test cases compared to the intended test coverage specified by the test purposes.

Tests generated from the model may result in test cases that have:

- the same coverage as defined by the test purposes;
- stronger coverage than defined by the test purposes; or
- weaker coverage than defined by the test purposes.

A generated test with the same coverage as specified by the test purpose tests precisely (and only) the aspects specified by the test purpose. Successful completion of the test case means that the test purpose has been met, while failure means that it has not been met.

A generated test with stronger coverage than specified by the test purpose tests more aspects than specified by the test purpose. For example, the message sent by the IUT might have several fields and the expected values of all of these are specified by the model. At the same time, the test purpose may only require the testing of the value of one of these against the expected value. Successful completion of such a test case proves that the test purpose has been met, but failure does not prove that the test purpose has not been met. For example, the expected value of a field specified by the test purpose may match the actual value in the received message (i.e. test case execution is successful against the test purpose) but the mismatch of the actual value against the expected value of another field in the same message may make the test case fail.

The test purposes define the precondition states. They assume that implementation is in the precondition state before a specified test begins. How the precondition state should be achieved is never defined by the test purposes: it is given that the precondition should be met before the test can start. The execution of the test case generated from the state model starts from the initial state of the model and covers the model items according to the test purpose. This means that the generated test covers not only the functionality required by the test purposes, but also the functionality to reach the precondition state. A generated test of this kind covers the model more broadly than specified by the test purpose and has stronger coverage than the test purpose.

A generated test with weaker coverage than specified by the test purpose tests fewer aspects than defined by the test purpose. For example, the test purpose *"Ensure that the IUT, to establish a call, sends an* `INVITE` *request including a* `From` *header with a* `TAG` *parameter"* cannot be covered precisely because, with a static template, the presence of a `TAG` cannot be checked in the current MOTES implementation. The generated test case tests all other aspects specified by the test purpose except the presence of the `TAG` parameter. Successful completion of such a test case does not prove that the test purpose has been met, but failure proves that the test purpose has not been met.

Test coverage analysis of 78 generated test cases against the corresponding test purposes gave the following results:

- 3 test cases matched the test coverage required by the corresponding test purposes
- 69 test cases had stronger test coverage than required by the corresponding test purposes

- 6 test cases had weaker test coverage than required by the corresponding test purposes

7.2.7 Objectives evaluation

The evaluation results of the case study objectives are as follows:

Overall feasibility of MOTES technology for conformance testing of industrial-scale telecommunications applications

The Sofia-SIP stack is a typical telecommunications protocol application. The case study demonstrated that MOTES technology is feasible for conformance testing of such applications.

Adaptability of MOTES technology for the requirement-driven testing process

As expected, adapting MOTES technology for requirement-driven testing causes some issues. The elements of the model that are involved by modelling particular requirements of the IUT are spread throughout the model. This makes it hard to understand in which part of the model and how a particular requirement is modelled. Tracking the requirements from specification to model is highly complicated.

In the case study the test purposes were mapped to the model structural coverage items in order to generate test cases according to the specified test purposes. It was demonstrated that for most test purposes (87%) the mapping was possible.

The test coverage of the generated test cases against the test purposes was compared. It was demonstrated that in most cases (69 test purposes out of 78) the generated test case resulted in stronger test coverage than specified by the corresponding test purpose. Only in some cases (3 test purposes out of 78) did the generated test case result in weaker test coverage than specified by the corresponding test purpose.

Feasibility of EFSM for modelling industrial-scale telecommunications applications

The case study demonstrated that EFSM has enough modelling power to model industrial-scale telecommunications applications. The complexity of the resulting EFSM is on the edge of readability and maintainability. The lack of hierarchical states in the modelling notation was experienced as a drawback.

Feasibility of the iterative model checking-based test generation method

The use of the deterministic model was justified in the case study. The SIP stack UAC component itself exhibits well-specified deterministic behaviour and the

implementation does not have internal sources of nondeterminism. The iterative model checking-based test generator engine was used for test generation in the case study. The iterative method did the work for all test purposes that were able to be defined using the sets of model structural coverage items. In addition, the tests were created for the *all transitions* test purpose, which resulted in a test sequence of 120 transitions. Note that attempting this on the same model and same test goal with normal model checking (not iterative) did not produce results, because the model checker froze after consuming all of the available 1GB memory resources allocated to it.

Bug detection ability of generated tests

The Sofia-SIP stack library has been available to developers for some years. The quality of Sofia-SIP implementation is otherwise unknown. During the case study 5 bugs were detected in Sofia-SIP implementation.

7.3 Testing controllers of street lighting system

7.3.1 Overview of IUT

The IUT of this case study is a Feeder Box Control Unit (FBCU), a subsystem of the street lighting control system operating in Tartu, the second biggest city in Estonia.

The street lighting system monitors and controls several hundred street lighting feeder boxes. Each feeder box provides a power supply to several groups of connected street lights. The street lighting control system controls the feeder boxes over a commercial GSM network. The FBCU is the control device of the feeder box (see Figure 16). It remotely controls the street lights, switching them on and off based on the light conditions, time of day or direct operator request. It is possible to continuously alter the time the lights come on as the seasons change and to take into account the weather conditions, the location of the lights and the nature of the environment they are lighting at particular time. The network enables feeder boxes to send monitoring and alarm information back to the lighting operator. The control system collects data about input and output feeder behaviour, identifies lighting failures and produces comprehensive reports on the status of the lights, faults and energy consumption. The general architecture of the street lighting control system is shown in Figure 16.

The communication between the FBCU and lighting control system is implemented using GSM USSD communication. A typical scenario of communication between the lighting operator and FBCU is as follows:

1. The lighting operator sends a message for a feeder box to a mobile operator USSD gateway through the Internet.
2. The USSD gateway receives the message and forwards it to the mobile network.
3. The mobile network dispatches the message to the FBCU of the feeder box.

4. The FBCU executes the command and responds to the mobile network.
5. The message is sent from the mobile network to the USSD gateway.
6. The USSD gateway forwards the message to the lighting operator.

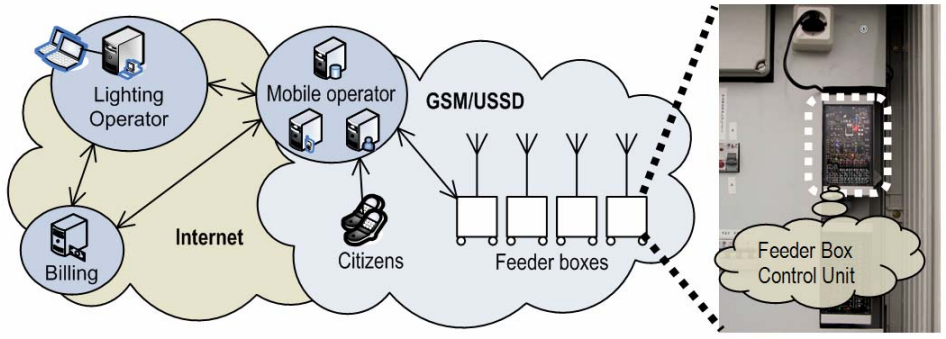

Figure 16: Architecture of street lighting control system

The FBCU consists of a microcontroller, an interface to the integrated GSM modem, a SIM card socket, impulse power supply units and protection circuits between the micro controller pins and FBCU external connectors.

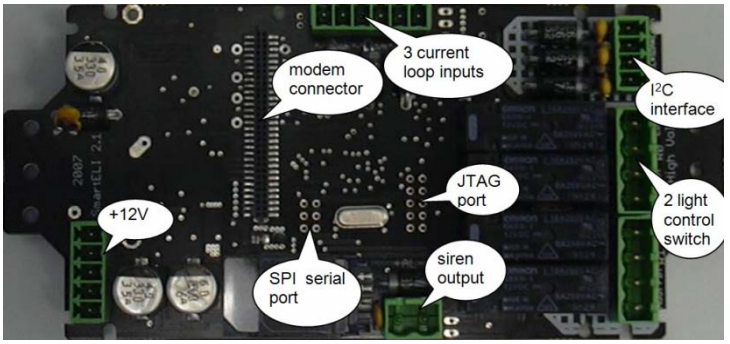

Figure 17: FBCU

7.3.2 Objectives of case study

The case study was performed as part of the ITEA2 D-MINT project [DMIN] by an Estonian consortium to evaluate model-based testing and related technology in the field of telematics. The testing scope in the case study included the start-up and power control features of the FBCU.

The following aspects were evaluated in the case study:

- **Overall feasibility of MOTES technology for testing hybrid industrial-scale controllers.** FBCU is a complex hybrid embedded system consisting of hardware and software components. It communicates with the environment using analogue and discrete input/output signals. In addition, it communicates over the GSM modem through messages. The specification describing the functionality of the IUT in the scope of the case study includes 18 use cases described over 45 pages. The embedded software of FBCU includes about 4000 lines of C code.
- **Combining MOTES technology with the complex hybrid system adapter.** The test tool, which has message-based communication with the IUT, had to be connected to the electrical input/output signals of the IUT. This required a complex system adapter containing software and hardware components to control and observe the IUT in the tests. In addition the system adapter had to have a component which would allow visual monitoring of the electrical signal characteristics in order to trace test case influence at the IUT pin level.
- **Applicability of MOTES technology for exploratory testing**. The term '*exploratory testing*' is normally used as the opposite of automatic script-based testing [Bach03]. It refers to manual testing where a skilled test engineer manually explores the state-space of the application trying to break the system. According to advocates of exploratory testing, script-based testing can never lead to equally good results compared to exploratory testing because the scripting is not flexible and it takes too much time to code the intelligence of the test engineer into the test script. The applicability of MOTES technology for exploratory testing was evaluated in the case study.
- **Feasibility of EFSM for modelling hybrid industrial-scale controllers.** The modelling power, size, complexity, and readability of the resulting EFSM were evaluated in the case study.
- **Feasibility of the iterative model checking-based test generation method.** The applicability of the method for generating test cases from the deterministic model was evaluated. Although the FBCU is nondeterministic in nature, as discovered in the case study, the nondeterminism was abstracted away using `FunctionCall` and `interleave` constructs of MTL [Appendix A]. Using custom TTCN-3 functions the interleaving and nondeterministic behaviour of the FBCU was hidden in the system model. The feasibility of the new construct was evaluated in the case study.
- **Feasibility of the reactive planning tester-based test generation method.** Reactive planning tester-based test generation was evaluated on the nondeterministic model of the FBCU. The feasibility of the RPT

method, its scalability and performance against random choice and *anti-ant* on-the-fly test generation methods were evaluated.

- **Ability of generated tests to detect bugs.** Although FBCU was already part of the field tests when the case study began, it was still under development. A couple of new hardware versions and several new software versions were introduced during the case study. Therefore, it was essential to assume that real bugs would be detected. The nature of the detected bugs was evaluated.
- **Suitability of MOTES technology for incremental system development.** Due to the incremental development of the FBCU during the case study it was possible to evaluate MOTES technology in the system development maintenance phase. The maintenance phase is characterised by changing system requirements with redesign, reimplementation, and retesting phases. The performance of automatic test generation against the estimated performance of manual test creation in the maintenance phase was able to be compared.

7.3.3 System adapter

Because of the complex electrical interface of the FBCU the most time-consuming activities in building the test environment were those related to the system adapter. The system adapter is a complex system of hardware and software components. The hardware consists of commercial measurement and control devices connected to the pins of the FBCU. These devices provide current to the FBCU, read its output pins and write data to its input pins. The devices are controlled by the virtual instruments of the LabVIEW [LabV] environment. The task of the LabVIEW virtual instruments is to receive the output messages produced by the TTCN-3 test code running on the MessageMagic TTCN-3 test tool [ELMM] in order to convert them into the control messages of the control devices that will provide the corresponding electrical signals to the FBCU. The LabVIEW virtual instruments read FBCU electrical output signals via the measurement devices and convert them to the input messages for the TTCN-3 test code running in MessageMagic (Figure 18).

7.3.4 Modelling the IUT

The FBCU is specified in the requirements specification, which includes use cases of FBCU behaviour. The requirements specification used in the case study was the best that can be expected. It was created by a skilled analyst who understood the test automation issues and was able to create specification that already included use cases in terms of logical signals, not in terms of electrical signals and their levels.

The case study included testing of the following functionalities of the FBCU:

- power supply management
- siren control
- door switch control

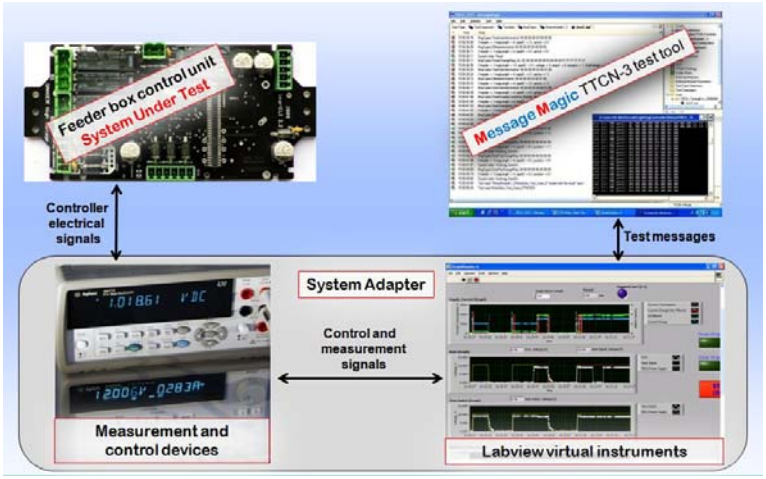

Figure 18: Test environment of FBCU

Modelling approach

Two modelling phases can be distinguished in the case study: the initial phase and the maintenance phase.

Initial phase

The initial phase included an analysis of the IUT based on the requirement document and interviews with the developers of the FBCU. In this phase several increments of the models for different testing purposes were created. The initial phase appeared to be quite a long process because it was performed in parallel with the development of the system adapter. There was a lot of experimenting and trial-and-error involved in the process due to the complex nature of the system adapter. Often there were two options – either to define something differently within the model or to re-implement something differently in the system adapter. Studying the problem area of the physical controller and understanding the relationship between different electrical signals was also difficult for the software test engineers. As most of the time was spent clarifying and experimenting with the model and system adapter, it is difficult to view the modelling phase as a task in itself. It is almost impossible to estimate how much of the work was spent modelling the IUT and how much developing the system adapter and integrating the IUT into the test environment. In general, the initial phase lasted for approximately 42 man weeks. It was estimated that the performance of the test case creation in the first phase using automatic model generation would still exceed the performance of the TTCN-3 coding for the same amount of tests by around 2.5 times.

A deterministic model with 16 states, 51 transitions and 6 context variables was created during the initial phase. A fragment of the model is presented in Figure 19.

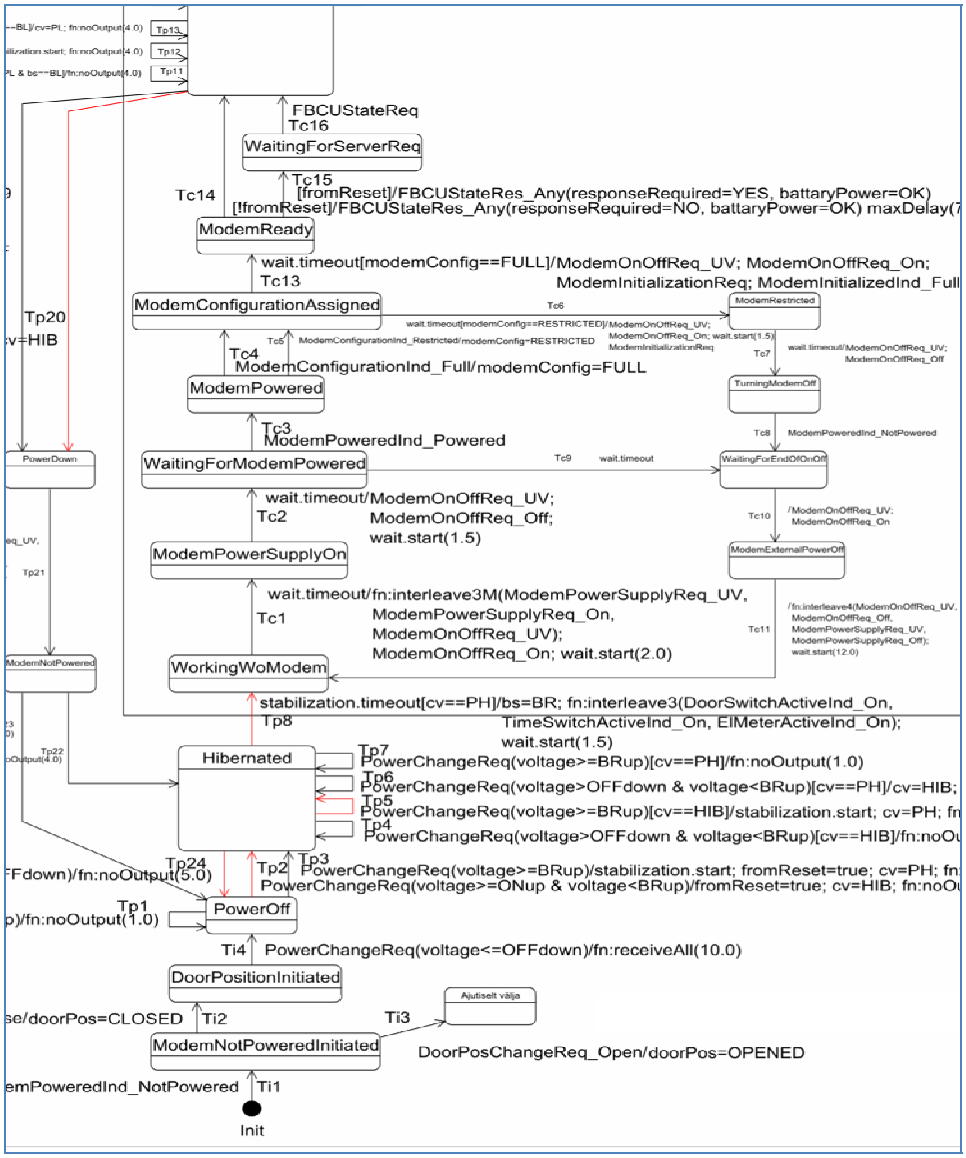

Figure 19: Fragment of FBCU deterministic model

Maintenance phase

The maintenance phase began after a new important requirement for the FBCU was set by the customers: "*It is very important to save the battery life, because it is costly to change the batteries on site"*. This requirement radically changed the start-up and power control functionalities of the FBCU. It was defined that starting from a particular battery current level, all of the functions of the FBCU should be shut down. This was very different from the previous requirements, where the FBCU attempted to be fully operational as long as the battery allowed

it to be. Due to the radical change in the behaviour of the IUT the existing models had to be modified to such an extent that it made more sense to create them again from scratch. As the problem area was already well-known and the system adapter was ready, the building of new models only took 10 man days.

A deterministic model with 20 states, 44 transitions and 6 context variables was created. The resulting model more or less covered the same scope of functionality as the model created during the initial phase.

In addition, a nondeterministic model was created to evaluate the feasibility and performance of the RPT method. The strongly connected state model of the FBCU includes 31 states and 78 transitions. Pairs of nondeterministic transitions depart from seven states of the model and a group of three nondeterministic transitions depart from one state of the model. The nondeterministic model was derived from the corresponding deterministic model by flattening the deterministic EFSM and adding a couple of nondeterministic transitions. A fragment of the nondeterministic model is presented in Figure 20.

7.3.5 Test generation and execution

Initial phase

The test case generated according to the *all transitions* test purpose includes a test sequence with 112 steps and 3000 LOC.

The test case generated according to the *all transition pairs* test purpose includes a test sequence with 286 steps and 4230 LOC.

In addition, tests were generated for 8 selected test purposes using test coverage defined by an ordered set of states and transitions. The generated test cases for these test purposes include 20,000 LOC in total.

Maintenance phase

The test case generated from the deterministic model according to the *all transitions* test purpose includes a test sequence with 75 steps and 2950 LOC.

The test case generated from the same model according to the *all transition pairs* test purpose includes a test sequence with 491 steps and 5080 LOC.

In addition, test cases were generated for 4 selected test purposes using test coverage defined by an ordered set of states and transitions. The generated tests for these test purposes include 10,000 LOC in total.

Reactive planning tester implementation in TTCN-3 generated from the nondeterministic model includes 5180 LOC.

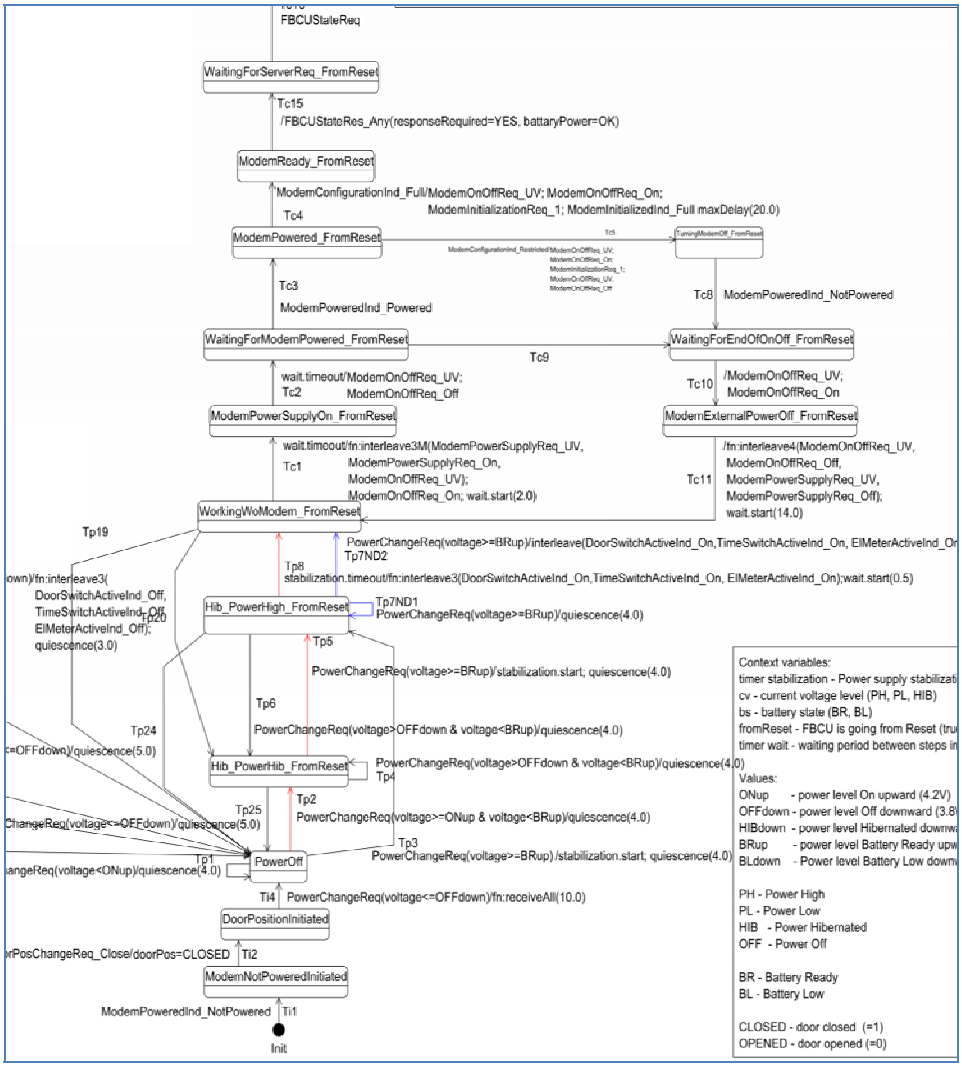

Figure 20: Fragment of FBCU nondeterministic model

7.3.6 Comparing test generation performance against manual test scripting

The benefits of automatic test generation compared to manual test scripting were estimated in the case study. The amount of generated TTCN-3 code was taken as the basis of evaluation. The amount of work required to manually create the same amount of TTCN-3 test code was calculated on the basis of the past TTCN-3 coding performance statistics available in Elvior.

MBT in general and MOTES technology in particular proved the benefits over manual test scripting. The productivity of system modelling and automatic test generation exceeds the manual process by many times. Building a model and a system adapter for the first time for an unknown field can take more time,

but in the maintenance phase, when there is an existing system adapter and model, subsequent changes are very fast. Additional tests can be created from the same model within moments and the generated tests are sure to be clean of bugs.

In the initial phase a considerable amount of time was spent on tasks such as analysing the IUT, building the system adapter, integrating the IUT into the test environment and creating models for first time. In this phase the performance was not much better compared to manual test scripting. The performance measurements of the case study exhibit around 60% of time savings over manual test building.

In the maintenance phase the time savings can be significant. With the know-how from the initial phase, it only took 10 days to build the new models from scratch and to generate new TTCN-3 test cases with the same coverage as in the initial phase. The code size of the generated TTCN-3 test cases was around 18 kLOC. Using the manual process, the writing of the same amount of test cases (to achieve the same test coverage) and to debug them would have required around 360 days (with test case coding performance of 50 LOC/day). Therefore the advantage of automatic test generation over the manual process appears to be around 36 times greater in the maintenance phase.

7.3.7 Objectives evaluation

The evaluation results of the case study objectives are as follows:

Overall feasibility of MOTES technology for testing hybrid industrial-scale controllers

The case study demonstrated that MOTES model-based testing technology in general is feasible for testing hybrid industrial-scale controllers like the FBCU. The case study demonstrated that most of the difficulties were in the building of the system adapter, not in modelling and test generation.

Combining MOTES technology with the complex hybrid system adapter

System adapter development proved to be the most time-consuming activity in test environment building for the model-based testing of hybrid systems. The case study demonstrated that such development of a complex system adapter only pays off if the test environment is reused over a longer period. Without regression testing needs in the maintenance phase, investing in such a system adapter is a waste of time and money. As demonstrated in the case study, only one significantly new increment in the maintenance phase justifies the investment made in building the system adapter.

Disclaimer: although discovered in the case study, the issues with the system adapter do not speak against MBT and MOTES technology. A similar system adapter is also needed in the event that the test cases are written manually without any MBT technology.

Applicability of MOTES technology for exploratory testing

The case study demonstrated that MOTES technology in particular and model-based testing in general is applicable for exploratory testing. In the case study, a general start-up and power control model was built which allows different current levels to be set up in start-up. Long test sequences with different powering schemes were generated from the same model by playing with different test coverage criteria. Several bugs in the controller were detected using such tests. Changing the coverage criteria and generating new test cases from the existing model is very simple and takes just a few seconds. Many experiments can be done in this way, as in exploratory testing, but in addition the IUT can be tested in a more systematic way with better coverage and repeatable results. These are significant benefits of model-based exploratory testing over manual exploratory testing.

Feasibility of EFSM for modelling hybrid industrial-scale controllers

The case study demonstrated that EFSM has enough modelling power to model hybrid industrial controllers. The EFSMs built in the case study contain, on average, 18 states, 47 transitions and 6 context variables. The complexity of such EFSMs is on the edge of readability and maintainability. The lack of hierarchical and parallel states in the modelling notation used was experienced as a drawback.

Feasibility of the iterative model checking-based test generation method

Although the FBCU is nondeterministic in nature, the nondeterminism was abstracted away using `FunctionCall` and `interleave` constructs of MTL. Because the models of the IUT did not include kinds of nondeterminism other than the order of expected receiving messages, the deterministic model with abstraction worked well in the case study.

Generating tests according to *all transitions* and *all transition pair* test purposes (which resulted in test sequences of 112 and 286 steps respectively) demonstrated the feasibility of the iterative model checking-based test generation method for generating serious test cases.

Feasibility of the reactive planning tester-based test generation method

Reactive planning tester-based test generation was evaluated on the nondeterministic model of the FBCU. The feasibility of the RPT method, its scalability and performance against random choice and *anti-ant* on-the-fly test generation methods were evaluated in the experiments of the case study. The method of the experiments and the results are presented in PAPER 4. In short, reactive planning tester-based test generation gives significantly shorter test sequences than other on-the-fly test generation methods like random choice and

anti-ant. This is achieved by the longer planning horizon of the RPT compared to other methods. The experiments demonstrated that increasing the planning horizon decreases the length of the test sequences exponentially. The online planning time only increases by a power of 2 in the length of the planning horizon.

Ability of generated tests to detect bugs

Although FBCU was already in field tests when the case study began, it was still under development. A couple of new hardware versions and several new software versions were introduced during the case study. 3 software bugs were detected during the case study.

Suitability of MOTES technology for incremental system development

Due to the incremental development of the FBCU during the case study, it was possible to evaluate MOTES technology in the system development maintenance phase. The maintenance phase is characterised by changing system requirements with redesign, reimplementation and retesting phases. Model-based testing in general and MOTES technology in particular are efficient for incremental system development as demonstrated by the test creation performance measurements in the maintenance phase of the case study. Any automated test environment pays off when reused several times during the product life cycle. The same is true for MBT. The case study demonstrated that the benefits of MBT in the maintenance phase can be enormous.

7.5 Summary of case studies

Two different case studies from different branches of the industry were conducted. Table 3 summarises their qualitative and quantitative aspects and results in order to place the case studies in a broader perspective and to compare them against each other.

The most important lessons learned from the case studies were as follows:

1. The system models in the case studies model a large amount of requirements of the system. The ability of MOTES technology to cope with such models is evidence that the technology is applicable to model-based testing of industrial-scale applications.
2. MOTES technology is generally applicable for functional black box testing as shown in two different case studies from different fields.
3. The iterative model checking-based test generation method implemented in MOTES is feasible and scalable for industrial-scale deterministic application testing.
4. The RPT-based test generation method implemented in MOTES is feasible and scalable for industrial-scale nondeterministic application testing. RPT generates significantly shorter test sequences than other online test generation methods like *anti-ant* and random choice. Efficiency is provided

with affordable online calculation time which remains in the range of tens of milliseconds.

5. EFSM is feasible for modelling the IUTs in the demonstrated case studies but it was found that flat EFSM models of such a size are on the edge of readability and maintainability. There is an urgent need for Statecharts modelling notation with hierarchical and parallel states.

Table 3: Case studies in numbers

Aspect	**SIP case study**	**FBCU case study**	
		initial phase	maintenance phase
Field of industry	telecommunications software	telematics/controllers	
Specification size (pages)	app. 90	app. 50	
IUT implementation size (LOC)	about 500 (C++)	about 400 (GNU C)	
Time spent on case study (weeks)	10	42	2
Deterministic/nondeterministic model	deterministic	deterministic	both
Number of models	1	1	2
Average number of states/transitions/context variables in model	19/52/24	16/51/6	25/61/3
Average length of test sequence for *all transitions* coverage	120	112	75
Total number of generated test cases	36	10	6
Average LOC in generated test cases	app. 570	app. 2700	app. 3000
Average test code generation productivity	410 LOC/day	128 LOC/day	1 800 LOC/day
Test generation performance over manual coding	app. 8 x	app. 2.5 x	app. 36 x

6. MOTES technology can, to some extent, be adapted for the requirement-driven testing process by defining the test purposes using model structural coverage items. It was demonstrated in the SIP case study that this is possible, but it lacks the means of tracing the requirements through the model to the test cases and test logs. Further research is needed for modelling requirements in a way that allows parts of the model to be easily selected as test coverage items and a test case to be generated for a particular requirement. PAPER 5 proposes an idea of model composition. More advanced development of the idea is a topic for future research.
7. Currently, MOTES technology is not able to generate test data automatically. Only test sequences and evaluation of the actual outputs against those expected can be generated automatically. Input data instances are prepared manually for test sequences. This is a drawback that slows down model-based testing and prevents the use of data coverage criteria for

test generation. Further research is needed to generate input data instances automatically.

8. The case studies, and especially the FBCU case study, demonstrated model-based test generation productivity against traditional manual test case development. After the initial investments are made in the building of the system adapter and knowledge has been gained about the problem area during the initial phase, model-based testing becomes highly productive in the maintenance phase, as demonstrated in the FBCU case study (Table 1).
9. The FBCU case study demonstrated MBT feasibility for exploratory testing.

8 CONCLUSIONS

MBT is an automation approach for deriving tests automatically from the model of the implementation under test (IUT). In reactive systems the MBT is an automation approach for deriving tests automatically for IUT functional black box tests.

The main motivation behind my doctoral studies was to create an MBT tool for reactive systems that is relatively easy for test engineers to use and which meets the needs of industrial-scale model-based testing. My aim was to develop a technology that allows the IUT to be modelled using a modelling language as close to UML as possible, since UML is a widely accepted modelling standard that is used in different branches of industry.

Several novel methods and techniques were proposed that form the theoretical foundation of the thesis:

- iterative model checking-based test generation from deterministic models
- synthesis of reactive planning tester for test generation from nondeterministic models
- requirement-driven testing through model composition

The reported iterative test generation method is based on explicit state model-checking. It is a well-known issue that model checking suffers a state-space explosion when the complexity of the model and/or reachability goal increases. The iterative model checking-based test generation method builds the tests iteratively by splitting the reachability goal into less complex sub-goals. In each iteration only one sub-goal is solved and the resulting test sequence is appended to the test sequence generated for the previous sub-goals. The method is therefore not vulnerable to a state- space explosion caused by the complexity of the test goal and allows tests to be generated from significantly larger and more complex models. Test sequences generated in such a way are suboptimal only. Splitting the goal into sub-goals can be viewed as another optimisation task done either manually by the test engineer or by program automatically by the program.

The reactive planning tester-based method of deriving tests for nondeterministic systems has significant advantages over other known on-the-fly testing methods due to its longer planning horizon. The reactive planning tester is synthesised offline using reachability analysis of the model. During the online phase the reactive planning tester is able to find the suboptimal way to the next test coverage item automatically. Due to the longer planning horizon the reactive planning tester meets the defined test goals significantly faster than random walk or *anti-ant*-like on-the-fly testing algorithms, for example.

In industry, software testing processes are usually requirement-driven. This means that testing should verify whether a particular software requirement defined in requirements specification or other specifications is implemented correctly. The requirement-driven approach in MBT should be supported as

early as the modelling phase by suitable modelling formalisms. The requirements should be able to be modelled in a way that later allows parts of the model to be easily selected as test coverage items belonging to the particular requirement and a test case to be generated for this particular requirement. It is shown how the requirement-driven approach can be applied in building the model of the IUT using either a composition of Uppaal [UPPA] automata or NModel model program [JVC+08] *features*.

The reasoning behind the development of these theoretical methods and techniques has been presented at international conferences and published in refereed journals. A selection of representative papers has been collated and attached to this thesis.

The practical part of the thesis includes a description of the MOTES test tool and technology, which implements the novel test generation methods presented in the thesis. The MOTES tool is a test generator which generates test cases in TTCN-3 language from extended finite state models (EFSM). The applicability of MOTES technology has been demonstrated in two industrial scale case studies. The thesis presents the results and lessons learned from these case studies in different branches of industry – telecommunications software and telematics controllers.

The research presented in the thesis highlighted several topics that will require attention in future research. This thesis focuses on EFSM notation for modelling the IUT. EFSM is a state machine without hierarchical and parallel states. For users to accept the MBT it is important to provide them with the chance to also use hierarchical and parallel states. Generating tests from models that also have hierarchical and parallel states is an important topic of future research. The thesis pays close attention and reports on good results in generating test sequences from deterministic and nondeterministic models. The test data for the generated tests were still prepared manually. Automatic test data generation is the next important topic for future research. The state models of the system were the only modelling artefacts used in this research. In practice there are several modelling artefacts the user would like to use in parallel to model different aspects of the systems. Building the system model from integrated sub-models that use different modelling artefacts is another important topic of future research.

REFERENCES

[3GPP] 3GPP web pages, http://www.3gpp.org/.

[ARTI] Artisan Studio products web pages, http://www.artisansoftwaretools.com/products/artisan-studio.

[Bach03] Bach, J.: Exploratory Testing Explained, http://www.satisfice.com/articles/et-article.pdf.

[BBC+06] Bernard, E., Bouquet, F., Charbonnier, A., Legeard, B., Peureux, F., Utting, M., Torreborre, E: Model-based Testing from UML Models. In: *MBT'2006, Model-based Testing Workshop*, INFORMATIK'06, volume P-94 of LNI, Lecture Notes in Informatics, Dresden, Germany, pages 223 - 230, October 2006.

[BEI95] Beizer, B.: *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. ISBN-10: 0471120944. John Wiley & Sons, Inc, 1995.

[BG02] Blass, A., Gurevich, Y.: Pairwise Testing. In: *Bulletin of the European Association for Theoretical Computer Science*, Number 78, October 2002, 100-132, 2002.

[BFV+99] Belinfante, A., Feenstra, J., Vries, R. d., Tretmans, J., Goga, N., Feijs, L., Mauw, S., Heerink, L.: Formal test automation: A simple experiment. In: *Csopaki, G., Dibuz, S., Tarnay, K. (eds.) 12th Int. Workshop on Testing of Communicating Systems*, pp. 179 - 196. Kluwer Academic Publishers, 1999.

[BJK+05] *Model-Based Testing of Reactive Systems*, Editors: Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A., no. 3472. In *LNCS*, Springer-Verlag, 2005.

[CCG+99] Cimatti, A., Clarke, E. M., Giunchiglia, F., Roveri, M.: NUSMV: A New Symbolic Model Verifier. In: *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*, pages 495 - 499, London, UK, Springer-Verlag, 1999.

[CGP99] Clarke, E. M., Grumberg, O., Peled, D. A.: *Model checking*, MIT Press, Cambridge, MA, USA, 1999.

[CLP08] Carter, J. M., Lin, L., Poore, J. H.: Automated Functional Testing of Simulink Control Models. In *Proceedings of the 1st Workshop on Model-based Testing in Practice – MoTiP 2008*. Editors: Bauer, T., Eichler, H., Rennoch, A., ISBN: 978-3-8167-7624-6, Berlin, Germany. Fraunhofer IRB Verlag, 2008.

[ConfS] Methods for Testing and Specification (MTS): Conformance Test Specification for SIP (IETF RFC 3261). Part 2: Test Suite Structure and Test Purposes (TSS & TP), ETSI TS 102 027-2 V4.1.1 (2006-07), http://www.etsi.org, 2006.

[DMIN] ITEA2 D-MINT project web pages, http://www.d-mint.org.

[Duf91] Duffy, D. A.: *Principles of automated theorem proving*, ISBN:0-471-92784-8, John Wiley & Sons, Inc. New York, NY, USA, 1991.

[EJ73] Edmonds, J., Johnson, E. L.: Matching, Euler Tours, and the Chinese Postman, *Mathematical Programming,* 5:88-124, 1973.

[ELMM] MessageMagic TTCN-3 test tool, Elvior MessageMagic product web pages, `http://messagemagic.elvior.com`.

[ETPR] Methods for Testing and Specification (MTS): Internet Protocol Testing (IPT): Testing: Methodology and Framework, ETSI EG 202 568 (V1.1.3), `http://www.etsi.org`.

[ETSI] European Telecommunication Standards Institute (ETSI) web pages, `http://www.etsi.org`.

[GGR+06] Gaston, C., Gall, P., Rapin, N., Touil, A.: Symbolic Execution Techniques for Test Purpose Definition. In: Testing of Communicating Systems, Lecture Notes in Computer Science, Volume 3964, pages 1 – 18, Springer Berlin / Heidelberg, 2006.

[GHR+03] Grabowski, J., Hogrefe, D., Réthy, G., Schieferdecker, I., Wiles, A., Willcock, C.: An introduction to the testing and test control notation (TTCN-3): Computer Networks: The International Journal of Computer and Telecommunications Networking, Vol 42, Issue 3, June 2003; pp. 375-403.

[GMS+07] Gadkari, A. A., Mohalik, S., Shashidhar, K. C., Yeolekar, A., Suresh, J., Ramesh, S.: Automatic Generation of Test-Cases Using Model Checking for SL/SF Models. In *Proceedings of MoDeVVa'07 in conjunction with MoDELS2007.* Nashville, Tenesee. 2007.

[HEL+05] Helmerich, A., Koch, N. and Mandel, L., Braun, P., Dornbusch, P., Gruler, A., Keil, P., Leisibach, R., Romberg, J., Schätz, B., Wild, T. Wimmel, G.: *Study of Worldwide Trends and R&D Programmes in Embedded Systems in View of Maximising the Impact of a Technology Platform in the Area,* Final Report for the European Commission, Brussels Belgium, 2005.

[HMR04] Hamon, G., de Moura, L., Rushby, J.: Generating efficient test sets with a model checker. In: *2nd International Conference on Software Engineering and Formal Methods*, Beijing, China, IEEE Computer Society, pp. 261–270, 2004.

[HN96] Harel, D., Naamad, A.: The STATEMATE semantics of statecharts. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Volume 5, Issue 4 (October 1996), pages 293 – 333, ISSN:1049-331X, 1996.

[Ho02] Hong, H. S., Lee, I., Sokolsky, O., Ural, H.: A temporal logic based theory of test coverage and generation. In: TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, London, UK, Springer-Verlag, pp. 327–341, 2002.

[Hog91] Hogrefe, D.: OSI-formal specification case study: The INRES protocol and service, Technical Report 91-012, University of Bern, 1991.

[Hol97] Holzmann, G. J.: The Model Checker SPIN. In: *IEEE Transactions of Software Engineering*, 23(5), pp. 279 - 295, 1997.

[HOW97] Howden, W. E.: Systems testing and statistical test data coverage. In:
Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC), pages 500 – 504, IEEE Computer Society Washington, DC, USA, 1997.

[Hui07] Huima, A.: Implementing Conformiq Qtronic. In: *Testing of Software and Communicating Systems*, Lecture Notes in Computer Science, Volume 4581, pages 1 – 12, Springer Berlin / Heidelberg, 2007.

[IMS] IP Multimedia Subsystem – Wikipedia, the Free Encyclopaedia. `http://en.wikipedia.org/wiki/IP_Multimedia_Subsystem`

[JAVA] Java web page, `http://java.sun.com`.

[JUS08] Justyna, Z.-N.: Model-based Testing of Real-Time Embedded Systems in the Automotive Domain. *PhD Thesis, Faculty IV – Electrical Engineering and Computer Science, Technical University Berlin.* Berlin, Germany, 2008.

[JVC+08] Jacky, J., Veanes, M., Campbell, C., Schulte, W.: *Model-based Software Testing and Analysis with C#.* Cambridge University Press, 2008.

[KBC05] Alnins, A., Barzdins, J., Celms E.: Model transformation language MOLA. In: *Proceedings of MDAFA: Model-Driven Architecture - Foundations and Applications*. Workshop, Linköping , SUEDE (10/06/2004), Lecture notes in computer science, vol. 3599, pp. 62-76, 2005.

[KLP+04] Kosmatov, N., Legeard, B., Peureux, F., Utting, M.: Boundary coverage criteria for test generation from formal models. In *Proceedings of the 15th International Symposium on Software Reliability Engineering*. ISSN: 1071-9458, ISBN: 0-7695-2215-7, Pages: 139 – 150. IEEE Computer Society Washington, DC, 2004.

[LabV] National Instruments, LabVIEW product web pages, `http://www.ni.com/LabVIEW/`.

[LL05] Li, H., Lam, C. P.: Using anti-ant-like agents to generate test threads from the UML diagrams. In: *TestCom*, pages 69–80, 2005.

[LMN+05] Larsen, K. G., Mikucionis, M., Nielsen, B., Skou, A.: Testing Real-time Embedded Software using UPPAAL-TRON - An Industrial Case Study. In: Proceedings of the 5th ACM International Conference on Embedded Software. Jersey City, NJ, USA. September 18-22, pp. 299-306, 2005.

[MA00] Marre, B., Arnould, A.: Test sequences generation from LUSTRE descriptions: GATEL. In *Proceedings of ASE of the 15th IEEE International Conference on Automated Software Engineering*, pages 229 – 237, ISBN: 0-7695-0710-7, Grenoble, France. IEEE Computer Society Washington, DC, 2000.

[McM92] McMillan, K. L.: *The SMV system.* Technical Report CMU-CS-92-131, Carnegie-Mellon University, 1992.

[MM63] Miller, J. C., Maloney, C. J.: Systematic mistake analysis of digital computer programs, *Communications of the ACM*, vol. 6, pp. 58-63, 1963.

[MOR+04] de Moura, L., Owre, S., Rueÿ, H., Rushby, J., Shankar, N., Sorea, M., Tiwari, A.: SAL 2. In: *Rajeev Alur and Doron Peled, editors, Computer-Aided Verification, CAV 2004*, volume 3114 of Lecture Notes in Computer Science, pages 496 - 500, Boston, MA, July 2004. Springer-Verlag. 2004.

[MT09] Meyn, S., Tweedie, R. L.: *Markov Chains and Stochastic Stability*. ISBN 978-0-521-73182-9, Cambridge University Press, 2009.

[NIST] NIST, `http://www.nist.gov/director/prog-ofc/report02-3.pdf`.

[PERL] Perl web page, `http://www.perl.org`

[PPW+05] Pretschner, A., Prenninger, W., Wagner, S., Kühnel, C., Baumgartner, M., Sostawa, B., Zölch, R., Stauner, T.: One evaluation of model-based testing and its automation. In *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, MO, U.S.A., Pages: 392 – 401, ISBN: 1-59593-963-2. ACM New York, NY, USA, 2005.

[PUML] Poseidon for UML, Gentleware product pages, `http://www.gentleware.com`.

[REAC] Reactis, Reactive Systems, Inc., `http://www.reactive-systems.com`.

[SIP] SIP: Session Initiation Protocol. The Internet Engineering Task Force, RFC 3261, `http://www.ietf.org/rfc/rfc3261.txt`.

[SOFC] Sofia SIP Library. Release 1.12.10, `http://sofia-sip.sourceforge.net/relnotes/relnotes-sofia/sip-1.12.10.txt`.

[SOFI] Sofia SIP library tutorial, web pages, `http://wiki.opensource.nokia.com/projects/SofiaTutorial`.

[SZF+04] Scott, E., Zadirov, A., Feinberg, S., Jayakody, R. The Alignment of Software Testing Skills of IS Students with Industry Practices – A South African Perspective, Journal of Information Technology Education, Volume 3, 2004.

[TTCN] TTCN-3 web page, `http://www.ttcn-3.org`.

[UL06] Utting M., Legeard B. *Practical Model-Based Testing: A Tools Approach.* ISBN-13: 9780123725011. Elsevier Science & Technology Books, 2006.

[UML] OMG: UML 2.0 Superstructure Final Adopted Specification, `http://www.omg.org/cgi-bin/doc?ptc/2003-08-02` [05/09/08].

[UpCo] Uppaal Cora web pages, `http://www.cs.aau.dk/~behrmann/cora/`.

[UPL06] Utting M., Pretschner A., Legeard B. *A taxonomy of model-based testing*, ISSN: 1170-487X, 2006.

[UPPA] Uppaal web pages, `http://www.uppaal.com/`.

[UTP] OMG: UML 2.0 Testing Profile. Version 1.0 formal / 05-07-07. Object Management Group, 2005.

[UTT05] Utting M. Model-Based Testing. In *Proceedings of the Workshop on Verified Software: Theory, Tools, and Experiments VSTTE 2005*. 2005.

[VCG+08] Veanes, M., Campbell, C., Grieskamp, W., Schulte, W., Tillmann, N., Nachmanson, L.: Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer. In: Formal Methods and Testing, vol. 4949, pp. 39-76, Springer Verlag, 2008.

[VRC06] Veanes, M., Roy, P., Campbell, C.: Online testing with reinforcement learning. In: *Havelund, K., Núñez, M., Rosu, G., Wolff, B. (eds.), FATES/RV*, volume 4262 of LNCS, pp. 240 – 253, Springer Verlag, 2006.

[Was04] Wasowski, A.: Flattening statecharts without explosions. In: *ACM SIGPLAN Notices*, Volume 39 , Issue 7, pp. 257 – 266, ACM New York, NY, USA, 2004.

[WN97] Williams, B. C., Nayak, P. P.: A reactive planner for a model-based executive. In Proc. of 15th International Joint Conference on Artificial Intelligence, IJCAI, pages 1178–1185, 1997.

PART III: RESEARCH PAPERS

PAPER 1: Ernits, J. P., Kull, A., Raiend, K., Vain, J., Generating tests from EFSM models using guided model checking and iterated search refinement. *In: Formal Approaches to Software Testing and Runtime Verification: First Combined International Workshops FATES 2006 and RV 2006, Seattle, WA, USA, August 15-16, 2006*, Revised Selected Papers: Havelund, K., et al. Berlin: Springer, 2006, (Lecture Notes in Computer Science; 4262), 85-99.

PAPER 2: Ernits, J. P., Kull, A., Raiend, K., Vain, J., Generating TTCN-3 test cases from EFSM models of reactive software using model checking. *In: Informatik 2006 - Informatik für Menschen: Proceedings: Beiträge der 36.Jahrestagung der Gesellschaft für Informatik e.V.(GI), 2.bis 6.Oktober 2006 in Dresden. (Ed.) Hochberger, Ch.; Liskowsky, R.*. Bonn: Köllen, 2006, (Lecture Notes in Informatics; P-94), 241 - 248.

PAPER 3: Vain, J., Raiend, K., Kull, A., Ernits, J., Synthesis of test purpose directed reactive planning tester for nondeterministic systems. *In: ASE'07 : 2007 ACM/IEEE International Conference on Automated Software Engineering, Atlanta, Georgia, November 5-9, 2007, proceedings: 22nd IEEE/ACM International Conference on Automated Software Engineering.* ACM Press, 2007, 363 - 372.

PAPER 4: Kull, A., Raiend, K., Vain, J., Kääramees, M., Case Study Based Performance Evaluation of Reactive Planning Tester. *In: CTIT Workshop Proceedings: 2nd Workshop on Model-based Testing in Practice (MoTiP 2009), June 23, 2009, Enschede, The Netherlands,* 2009, 87 – 96.

PAPER 5: Ernits, J. P., Kääramees, M., Raiend, K., Kull, A., Requirements-driven model-based testing of the IP Multimedia Subsystem. *In: BEC 2008: 2008 International Biennial Baltic Electronics Conference: Proceedings: 11th Biennial Baltic Electronics Conference, Tallinn University of Technology, October 6-8, 2008, Tallinn, Estonia.* Tallinn: Tallinn University of Technology, 2008, 203 - 206.

LIST OF PUBLICATIONS

Kull, A., Raiend, K., Vain, J., Kääramees, M. (2009). Case Study Based Performance Evaluation of Reactive Planning Tester. *In: CTIT Workshop Proceedings: 2nd Workshop on Model-based Testing in Practice (MoTiP 2009), June 23, 2009, Enschede, The Netherlands,* 2009, 87 – 96.

Ernits, J. P., Kääramees, M., Raiend, K., Kull, A. (2008). Requirements-driven model-based testing of the IP Multimedia Subsystem. *In: BEC 2008: 2008 International Biennial Baltic Electronics Conference: Proceedings: 11th Biennial Baltic Electronics Conference, Tallinn University of Technology, October 6-8, 2008, Tallinn, Estonia.* Tallinn: Tallinn University of Technology, 2008, 203 - 206.

Vain, J., Raiend, K., Kull, A., Ernits, J. (2007). Synthesis of test purpose directed reactive planning tester for nondeterministic systems. *In: ASE'07: 2007 ACM/IEEE International Conference on Automated Software Engineering, Atlanta, Georgia, November 5-9, 2007, proceedings: 22nd IEEE/ACM International Conference on Automated Software Engineering.* ACM Press, 2007, 363 - 372.

Kull, A., Raiend, K., Vain, J. (2007). Executable black-box tester model synthesis from a non-deterministic EFSM of the system. *In: Info- ja kommunikatsioonitehnoloogia doktorikooli IKTDK teise aastakonverentsi artiklite kogumik : 11.-12. mai 2007, Viinistu Kunstimuuseum:* [Tallinn: Tallinna Tehnikaülikooli Kirjastus], 2007, 105 - 108.

Vain, J., Raiend, K., Kull, A., Ernits, J. (2007). Synthesis of test purpose directed reactive planning tester for nondeterministic systems (extended abstract). *In: NWTP'07/FLACOS'07 Workshop Proceedings: Nordic Workshop in Programming Theory 2007, Oslo, 10.-12. October, 2007. (Ed.) Johnsen, E. B.; Owe, O.; Schneider G.,* Oslo: University of Oslo, 2007, 55 - 57.

Ernits, J. P., Kull, A., Raiend, K., Vain, J. (2006). Generating TTCN-3 test cases from EFSM models of reactive software using model checking. *In: Informatik 2006 - Informatik für Menschen: Proceedings: Beiträge der 36.Jahrestagung der Gesellschaft für Informatik e.V.(GI), 2.bis 6.Oktober 2006 in Dresden. (Ed.) Hochberger, Ch.; Liskowsky, R.,* Bonn: Köllen, 2006, (Lecture Notes in Informatics; P-94), 241 - 248.

Ernits, J. P., Kull, A., Raiend, K., Vain, J. (2006). Generating tests from EFSM models using guided model checking and iterated search refinement. *In: Formal*

Approaches to Software Testing and Runtime Verification: First Combined International Workshops FATES 2006 and RV 2006, Seattle, WA, USA, August 15-16, 2006, Revised Selected Papers: (Ed.) Havelund, K., et al., Berlin: Springer, 2006, (Lecture Notes in Computer Science; 4262), 85 - 99.

Kull, A. (2004). Improving Embedded Software Testing. *In: Proceedings 8th World Multi-Conference on Systemics, Cybernetics and Informatics: 8th World Multi-Conference on Systemics, Cybernetics and Informatics, July 18-21, 2004, Orlando, Florida, USA.. ,* 2004, 270 - 274.

APPENDIX A: FORMAL TRANSITION LANGUAGE FOR MOTES (MTL)

```
                  Identifier ::= <IDENTIFIER>
                  FloatValue ::= <FLOATING_POINT_LITERAL>
                   CharValue ::= <CHARACTER_LITERAL>
                     Cstring ::= <STRING_LITERAL>
                      Number ::= (<NUMBER>|"0")
        TransitionExpression ::= ((GuardCondition)?(Trigger)?
                                  TriggerEffectSeparator(
                                  EffectEffect)?<EOF>)
     TriggerEffectSeparator ::= (("|"))
              TriggerSingle ::= ((TimeTrigger|MessageTrigger)
                                  <EOF>)
                    Trigger ::= ((TimeTrigger|MessageTrigger))
      GuardConditionSingle ::= ("["Expression"]"<EOF>)
            GuardCondition ::= ("["Expression"]")
                      Port ::= (Identifier Colon)
            MessageTrigger ::= ((Port)?Identifier
                                  ("("InputParams")")?)
               InputParams ::= (InputParam(","InputParam)*)
                InputParam ::= (SingleExpression(AssignmentChar
                                  (SingleExpression |
                                  Ttcn3TemplateMechanism))*)
         InputParamSelection ::= ((EqualOp|RelOp)
                                  (Ttcn3TemplateMechanism|
                                  Expression))
      InputParamAssignment ::= (AssignmentChar Expression)
               TimeTrigger ::= (Identifier ".timeout")
              EffectSingle ::= (Action (";" Action)*<EOF>)
                    Effect ::= (Action (";" Action)*)
                    Action ::= ((StartTimer|StopTimer|
                                  FunctionCall|
                                  ContextVariableExpression|
                                  SendMessage|Quiescence|
                                  Interleave))
  ContextVariableExpression ::= (Assignment)
               SendMessage ::= (Message(SendMaxDelay)?)
                   Message ::= ((Port)?Identifier
                                  ("("OutputParams")")?)
              OutputParams ::= (OutputParam(","OutputParam)*)
               OutputParam ::= (SingleExpression(AssignmentChar
                                  (SingleExpression|
                                  Ttcn3TemplateMechanism))*)
                StartTimer ::= (TimerRef ".start"
                                  ("("(TimerValue)")")?)
```

```
                TimerValue ::= ((Identifier|FloatValue))
                 StopTimer ::= (TimerRef ".stop")
                Quiescence ::= ("quiescence" "("TimerValue")")
                Interleave ::= ("interleave" "("Message",
                               "Message(","Message)*")"
                               (SendMaxDelay)?)
              FunctionCall ::= ("fn:"Identifier"("(Value
                               (","Value)*)?")")
                  TimerRef ::= (Identifier(ArrayOrBitRef)?)
              SendMaxDelay ::= ("maxDelay" "("FloatValue")")
  Ttcn3TemplateMechanism ::= (("?"|"*"|"-"))
                   Value ::= ((VariableRef|PredefinedValue))
          PredefinedValue ::= ((BooleanValue|Cstring|Number|
                               FloatValue|CharValue))
              VariableRef ::= ((Identifier
                               (ExtendedFieldReference)?))
               Expression ::= ((SingleExpression|
                               ArrayExpression))
          ArrayExpression ::= (("{"(ArrayElementExpressionList)
                               ?"}"))
ArrayElementExpressionList ::= ((Expression(","Expression)*))
        BooleanExpression ::= ((SingleExpression))
              Assignment ::= ((VariableRef AssignmentChar
                               Expression))
        SingleExpression ::= ((AndExpression
                               (OrOp AndExpression)*))
           AndExpression ::= ((NotExpression(AndOp
                               NotExpression)*))
           NotExpression ::= ((("!")?EqualExpression))
         EqualExpression ::= ((RelExpression(EqualOp
                               RelExpression)*))
           RelExpression ::= ((AddExpression(RelOp
                               AddExpression)?))
           AddExpression ::= ((MulExpression(AddOp
                               MulExpression)*))
           MulExpression ::= ((UnaryExpression(MultiplyOp
                               UnaryExpression)*))
         UnaryExpression ::= (((UnaryOp)?Primary))
                 Primary ::= ((Value|"("SingleExpression")"))
  ExtendedFieldReference ::= ((((Dot Identifier)|
                               ArrayOrBitRef)+))
           ArrayOrBitRef ::= (("["FieldOrBitNumber"]"))
        FieldOrBitNumber ::= ((SingleExpression))
                   AndOp ::= (("&"|"&&"))
                    OrOp ::= (("|"|"||"))
                   AddOp ::= (("+"|"-"))
              MultiplyOp ::= (("*"|"/"|"mod"|"rem"))
```

```
       UnaryOp ::= (("+"|"-"))
         RelOp ::= (("<"|">"|">="|"<="))
       EqualOp ::= (("=="|"!="))
           Dot ::= ((","))
          Dash ::= (("-"))
         Minus ::= ((Dash))
     SemiColon ::= ((";"))
         Colon ::= ((":"))
AssignmentChar ::= (("="))
  BooleanValue ::= (("true"|"false"))
```

APPENDIX B: ELULOOKIRJELDUS

1. Isikuandmed

Ees- ja perekonnanimi:	Andres Kull
Sünniaeg ja -koht:	17.09.1961, Tallinn, Eesti
Kodakondsus:	Eesti

2. Kontaktandmed

Aadress:	Merirahu 49, Tallinn, Eesti
Telefon:	+372 5021203
E-posti aadress:	andres.kull@elvior.ee

3. Hariduskäik

Õppeasutus (nimetus lõpetamise ajal)	**Lõpetamise aeg**	**Haridus (eriala/kraad)**
Tallinna 46. Keskkool	Juuni 1980	Keskharidus
Tallinna Polütehniline Instituut	Juuni 1985	Süsteemiinsener/Kõrgharidus

4. Keelteoskus

Keel	**Tase**
eesti	emakeel
inglise	väga hea
vene	väga hea
soome	väga hea
saksa	algaja

5. Teenistuskäik

Töötamise aeg	**Tööandja nimetus**	**Ametikoht**
1992 – praegu	Elvior OÜ, Tallinn, Eesti	juht
1990 – 1992	VTT Electronics, Oulu, Soome	teadur
1985 – 1990	Protsessijuhtimise labor, Tallinna Polütehniline Instituut, Tallinn, Eesti	nooremteadur
1983 – 1985	Protsessijuhtimise labor, Tallinna Polütehniline Instituut, Tallinn, Eesti	assistent

6. Teadustegevus

1983 – 1990:	Bensoehappe tootmisprotsessi automatiseerimine.
1985 – 1990:	Tehnoloogiliste protsesside automaatjuhtimissüsteemide analüüs ja projektreerimine, teadmuste esitamine reaalaja juhtimissüsteemides, protsessijuhtimissüsteemide modelleerimine. Protsessijuhtimissüsteemi ja tema keskkonna formaalse operatsioonilise mudeli (Discrete Event Operational Specification Model) väljatöötamine. CASE tööriista prototüübi arendamine selle mudeli evalueerimiseks.
1990 – 1992:	Sardsüsteemide modelleerimine, koodi genereerimine. Sardsüsteemide testimine arvutis simuleeritud keskkonnas.
2003 – 2009:	Reaktiivsete süsteemide mudelipõhine testimine.

7. Kaitstud lõputööd

"Distributed process control for benzoic acid plant", Tallinna Polütehniline Instituut, automaatika kateeder, diplomitöö, 1985.

8. Teadustöö põhisuunad

Süsteemide automaatse testimise meetodid
Reaktiivsete süsteemide mudelipõhine testimine

APPENDIX C: CURRICULUM VITAE

1. Personal data

Name:	Andres Kull
Date and place of birth:	17.09.1961, Tallinn, Estonia

2. Contact information

Address:	Merirahu 49, Tallinn, Eesti
Phone:	+372 5021203
E-mail:	andres.kull@elvior.ee

3. Education

Educational institution	Graduation year	Education (field of study/degree)
Tallinna 46. Keskkool	1980	Secondary
Tallinna Polütehniline Instituut	1985	System engineer/M. Sc.

4. Language competence/skills

Language	Level
Estonian	mother tongue, native
English	very good
Russian	very good
Finnish	very good
German	beginner

5. Professional Employment

Period	Organisation	Position
1992 – current	Elvior OÜ, Tallinn, Estonia	CEO
1990 – 1992	VTT Electronics, Oulu, Finland	researcher
1985 – 1990	Laboratory of Process Control, Tallinn University of Technology, Tallinn, Estonia	junior researcher
1983 – 1985	Laboratory of Process Control, Tallinn University of Technology, Tallinn, Estonia	assistent

6. Scientific work

1983 – 1990:	Automating of benzoic acid plant.
1985 – 1990:	Analysis and design of process control systems, knowledge presentation in real-time process control systems, modeling of process control systems. I worked out a formal operational modeling notation for process control system and its environment (*Discrete Event Operational Specification Model*) and developed a prototype CASE tool for evaluating the concept.
1990 – 1992:	Embedded systems modelling and code generation. Testing embedded software in simulated environment.
2003 – 2009:	Model based testing of reactive systems.

7. Defended theses

"Distributed process control for benzoic acid plant", Tallinna Polütehniline Instituut, Department of Automatic Control, M. Sc., 1985.

8. Current research topics

Automated testing methods of systems
Model based testing of reactive systems