

**Software Engineering: Testing Real-Time
Embedded Systems Using Timed Automata
Based Approaches**

A Thesis submitted for the degree of Doctor of Philosophy

By

Mohammad Saeed Abou Trab

Department of Information Systems and Computing,

Brunel University

May 2012

ABSTRACT

Real-time Embedded Systems (RTESs) have an increasing role in controlling society infrastructures that we use on a day-to-day basis. RTES behaviour is not based solely on the interactions it might have with its surrounding environment, but also on the timing requirements it induces. As a result, ensuring that an RTES behaves correctly is non-trivial, especially after adding time as a new dimension to the complexity of the testing process. This research addresses the problem of testing RTESs from Timed Automata (TA) specification by the following. First, a new Priority-based Approach (PA) for testing RTES modelled formally as UPPAAL timed automata (TA variant) is introduced. Test cases generated according to a proposed timed adequacy criterion (clock region coverage) are divided into three sets of priorities, namely boundary, out-boundary and in-boundary. The selection of which set is most appropriate for a System Under Test (SUT) can be decided by the tester according to the system type, time specified for the testing process and its budget.

Second, PA is validated in comparison with four well-known timed testing approaches based on TA using Specification Mutation Analysis (SMA). To enable the validation, a set of timed and functional mutation operators based on TA is introduced. Three case studies are used to run SMA. The effectiveness of timed testing approaches are determined and contrasted according to the mutation score which shows that our PA achieves high mutation adequacy score compared with others.

Third, to enhance the applicability of PA, a new testing tool (GeTeX) that deploys PA is introduced. In its current version, GeTeX supports Control Area Network (CAN) applications. GeTeX is validated by developing a prototype for that purpose. Using GeTeX, PA is also empirically validated in comparison with some TA testing approaches using a complete industrial-strength test bed. The assessment is based on fault coverage, structural coverage, the length of generated test cases and a proposed assessment factor. The assessment is based on fault

coverage, structural coverage, the length of generated test cases and a proposed assessment factor. The assessment results confirmed the superiority of PA over the other test approaches. The overall assessment factor showed that structural and fault coverage scores of PA with respect to the length of its tests were better than the others proving the applicability of PA.

Finally, an Analytical Hierarchy Process (AHP) decision-making framework for our PA is developed. The framework can provide testers with a systematic approach by which they can prioritise the available PA test sets that best fulfils their testing requirements. The AHP framework developed is based on the data collected heuristically from the test bed and data collected by interviewing testing experts. The framework is then validated using two testing scenarios. The decision outcomes of the AHP framework were significantly correlated to those of testing experts which demonstrated the soundness and validity of the framework.

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervisor Dr Steve Counsell for the continuous support, patience, motivation and advice. His guidance helped me conducting this study, communicating with academics and writing of the Thesis. I could not have imagined having a better or friendlier supervisor.

It is also an honour for me to express my deep thanks to Professor Robert Hierons. His valuable and immense comments and advices gave me more insight through the research area.

I am also grateful to the academic staff in Northumbria University especially Dr Michael Brockway, Dr David Kendall, Dr William Henderson and my sincere PhD colleague Ali Mohammad. Their help in offering me a Lab access and providing me with all needed documents and advice is much counted and appreciated.

I wish to extend my warmest thanks to all academic staff and colleagues especially those who have helped me with my study in the Department of Information Systems and Computing in Brunel University.

I owe my loving thanks to my wife Nesreen and my kids Sameer and Seema. It would have been impossible for me to finish this study without their continuous support, encouragement and understanding. My special gratitude is due to my brother, my sister and their families for their loving support.

Last but not least, the financial support of Damascus University in Syria, my beloved Country, is gratefully acknowledged.

DEDICATION

This Thesis is dedicated to my parents Sameer and Amal for all these years they spent taking care of me. There are no words that can sufficiently describe their constant source of support, emotions and efforts during my postgraduate years. My achievements would certainly not have existed without them.

ABBREVIATIONS

AF	Assessment Factor
AHP	Analytical Hierarchy Process
ANP	Analytical Network Process
ASP	Adding a new Starting Point of a clock
B	Boundary set
BCT	Boundary Checking Technique
BDD	Binary Decision Diagram
CAN	Controller Area Network
CE	Coupling Effect
CFG	Control Flow Graph
CI	Consistency Index
CPH	Competent Programmer Hypothesis
CR	Consistency Ratio
CRC	Clock Region Coverage
CSP	Communicating Sequential Process
CSS	Communication and Concurrency Systems
DBM	Difference Bound Matrix
DEA	Data Envelopment Analysis
DMU	Decision Making Units
DOM	Document Object Model
DS	Distinguishing Sequences
ECC	Extending Clock Conditions
EFSM	Extended Finite State Machine
EIA	Exchanging Input Actions
EIP	Exchanging Input Parameters
EOA	Exchanging Output Actions
EOP	Exchanging Output Parameters
ERA	Event Recording Automata

FC	Fault Coverage
FSM	Finite State Machine
GA	Grid Automata
IB	In-Boundary set
ioco	Input-Output Conformance
K-T Decision Analysis	Kepner-Tregoe Decision Analysis
LTL	Linear-time Temporal Logic
LTS	Labelled Transition System
MAT	Mutation Analysis Technique
MAUT	Multi-Attribute Utility Theory
MBT	Model-Based Testing
MSC	Message Sequences
NCC	Narrowing Clock Conditions
NCR	Number of Clock Regions
NRC	Not-Resetting a Clock
OB	Out-Boundary set
PA	Priority-based Approach
PFSM	Probabilistic Finite State Machine
RA	Region Automata
RC	Resetting a Clock
RSP	Removing an existing Starting Point of a clock
RTC	Restricting Timing Constraints
RTES	Real-Time Embedded System
rtioco	Relativized Timed Input Output Conformance
SCC	Shifting Clock Conditions
SCT	State Characterization Technique
SDL	Specification and Description Language
SM	Scalable Method

SMA	Specification Mutation Analysis
SMAR	Simple Multi-Attribute Ratio technique
STC	Shifting Timing Constraints
SUT	System Under Test
TA	Timed Automata
TDL	Transferring Destination Locations
TET	Test Execution Time
TIOA	Timed Input Output Automata
tioco	Timed Input-Output Conformance
TP	Transition Path
TTI	Timed Trace Inclusion
TTL	Test Traces Length
TTS	Timed Transition System
TTTS	Testable Timed Transition System
UIO	Unique Input Output
UTA	UPPAAL Timed Automata
WP	Working Piece
WTC	Widening Timing Constraints
XML	eXtensible Mark-up Language
ZG	Zone Graph

LIST OF PUBLICATIONS

- Aboutrab, M. S. and Counsell, S. (2010) Fault Coverage Measurement of a Timed Test Case Generation Approach. *17th IEEE International Conference on the Engineering of Computer-Based Systems*, Oxford, UK, pp. 141-149.
- Aboutrab, M. S., Alrouh, B., Counsell, S., Hierons, R. and Ghinea, G. (2010) A Multi-criteria Decision Making Framework for Real Time Model-Based Testing. *Testing – Practice and Research Techniques*, London, UK: Springer Berlin / Heidelberg, pp. 194-197.
- Aboutrab, M. S., Counsell, S. and Hierons, R. M. (2011) GeTeX: A Tool for Testing Real-Time Embedded Systems Using CAN Applications. *18th IEEE International Conference on the Engineering of Computer-Based Systems*, Las Vega, USA, pp. 61-70
- Aboutrab, M. S., Counsell, S. and Hierons, R. M. (2012) Specification Mutation Analysis for Validating Timed Testing Approaches Based on Timed Automata. *IEEE Signature Conference on Computers, Software, and Applications (COMPSAC 2012)*, Izmir, Turkey (accepted).
- Aboutrab, M. S., Brockway, M., Counsell, S. and Hierons, R. M. (2012) Testing Real-time Embedded Systems using Timed Automata Based Approaches. *Journal of Systems and Software* (under second review).
- Aboutrab, M. S., Alrouh, B., Counsell, S., Hierons, R. and Ghinea, G. (2012) Prioritising Timed Automata Based Test Sets: A Multi-Criteria Decision Making Approach. *Journal of Systems and Software* (submitted).

TABLE OF CONTENTS

Chapter 1: Introduction	1
1.1 TOPIC OVERVIEW	1
1.2 RESEARCH MOTIVATION	3
1.3 RESEARCH AIM AND OBJECTIVES.....	5
1.4 SUMMARY OF THE CONTRIBUTIONS	6
1.5 THESIS OUTLINE.....	7
Chapter 2: Literature Review	10
2.1 OVERVIEW	10
2.2 SOFTWARE TESTING	11
2.3 TEST SELECTION PRINCIPLES	12
2.3.1 <i>Test Selection Strategies</i>	13
2.3.2 <i>Mutation Analysis Technique (MAT)</i>	14
2.4 TESTING TYPES	16
2.5 FORMAL METHODS IN SOFTWARE TESTING	19
2.6 FORMAL VERIFICATION.....	20
2.6.1 <i>Model Checking</i>	20
2.6.2 <i>Formal Verification and Testing</i>	21
2.7 MODEL-BASED TESTING (MBT).....	22
2.7.1 <i>Specification Formal Languages</i>	25
2.7.2 <i>Conformance Relations</i>	30
2.7.3 <i>Specification Mutation Analysis</i>	32
2.8 TIMED AUTOMATA BASED TESTING.....	34
2.8.1 <i>Timed Automata Specification Language</i>	35
2.8.2 <i>Timed Automata Abstraction Methods</i>	38

2.8.3	<i>TA Test Selection</i>	40
2.8.4	<i>TA Test Generation</i>	42
2.8.5	<i>Timed Conformance Relations</i>	44
2.8.6	<i>Related Work</i>	45
2.8.7	<i>Motivation for Automatic Testing from a TA</i>	50
2.9	SUMMARY	51
Chapter 3: A Priority-Based Approach for Testing Real-Time Embedded Systems		53
3.1	OVERVIEW	53
3.2	PROBLEM AREA	54
3.3	PRELIMINARIES	56
3.3.1	<i>Timed Automata (TA)</i>	56
3.3.2	<i>Clock Region Abstraction</i>	60
3.4	TIMED ADEQUACY CRITERION: CLOCK REGION COVERAGE (CRC).....	62
3.4.1	<i>CRC Considerations</i>	63
3.4.2	<i>Number of Clock Regions (NCR)</i>	65
3.4.3	<i>Feasibility Issue of CRC</i>	69
3.5	PRIORITY-BASED APPROACH (PA)	71
3.5.1	<i>Test Hypotheses</i>	72
3.5.2	<i>Test Selection</i>	72
3.5.3	<i>Test Generation Algorithms</i>	74
3.6	EMPIRICAL VALIDATION	78
3.6.1	<i>Mutation Operators for TA</i>	79
3.6.2	<i>Mutation Execution</i>	82
3.6.3	<i>Mutation Analysis</i>	83

3.6.4	<i>TA-based Testing Approaches</i>	84
3.6.5	<i>Case Studies</i>	86
3.6.6	<i>Results and Discussion</i>	88
3.7	SUMMARY	95
Chapter 4: Automatic Test Case Generation and Execution using the Priority-Based Approach		97
4.1	OVERVIEW	97
4.2	PROBLEM AREA	98
4.3	PRELIMINARIES	100
4.3.1	<i>Timed Input Output Conformance Theory (tioco)</i>	101
4.3.2	<i>Controller Area Network (CAN)</i>	102
4.4	GETEX TOOL DEVELOPMENT	103
4.4.1	<i>GeTeX Design</i>	103
4.4.2	<i>GeTeX Implementation</i>	106
4.4.3	<i>GeTeX Trail</i>	109
4.5	TESTING ASSESSMENT CRITERIA	114
4.5.1	<i>Structural Coverage Assessment Criterion (CRC)</i>	114
4.5.2	<i>Fault Coverage Assessment Criterion (MAT)</i>	115
4.5.3	<i>Test Traces Length Assessment Criterion (TTL)</i>	118
4.5.4	<i>Combined Assessment Factor (AF)</i>	119
4.6	EMPIRICAL ASSESSMENT BASED ON A COMPLETE TEST BED	120
4.6.1	<i>Production-Cell Test Bed</i>	120
4.6.2	<i>Specification Models</i>	122
4.6.3	<i>Test Generation and Execution</i>	126
4.6.4	<i>Assessment Discussion</i>	130

4.6.5 <i>Lessons Learned and Problems Encountered</i>	132
4.7 SUMMARY	134
Chapter 5: A Multi-Criteria Decision Making Approach for Prioritising the Test Sets of the Priority-Based Approach.....	135
5.1 OVERVIEW	135
5.2 PROBLEM AREA	136
5.3 PRELIMINARIES	138
5.3.1 <i>Decision Making Methods</i>	138
5.3.2 <i>Analytical Hierarchy Process (AHP)</i>	141
5.4 THE AHP FRAMEWORK.....	146
5.4.1 <i>Decision Problem</i>	146
5.4.2 <i>Decision Alternatives</i>	147
5.4.3 <i>Decision Criteria</i>	147
5.4.4 <i>Data Collection</i>	150
5.4.5 <i>Raised Power Matrices</i>	155
5.4.6 <i>Normalised Matrix and Eigenvector</i>	155
5.5 TESTING SCENARIOS	157
5.5.1 <i>Scenario 1: Control System</i>	158
5.5.2 <i>Scenario 2: Medical System</i>	163
5.6 SUMMARY	168
Chapter 6: Conclusions	169
6.1 TOPIC OVERVIEW	169
6.2 RESEARCH SUMMARY	170
6.3 MEETING THE RESEARCH OBJECTIVES	172
6.4 SUMMARY OF RESEARCH CONTRIBUTIONS	173

6.4.1 Timed Adequacy Criterion (CRC).....	173
6.4.2 Priority-based TA-based Testing Approach (PA).....	174
6.4.3 Specification Mutation Analysis.....	175
6.4.4 The application of TA-based Approaches on an industrial-strength Test Bed.....	175
6.4.5 A multi-Criteria Decision Making Framework.....	176
6.5 RESEARCH LIMITATIONS AND FUTURE WORK.....	177
6.5.1 The Class of TA Specification Model.....	177
6.5.2 Timed Adequacy Criterion.....	178
6.5.3 Case Studies.....	178
6.5.4 More insights for the Multi-Criteria Decision Making Approach.....	179
References.....	180
Appendix A.....	201
Appendix B.....	208
Appendix C.....	211
Appendix D.....	214

LIST OF TABLES

Table 3.1: The count of generated test cases.....	89
Table 3.2: SMA application on the lamp controller.....	90
Table 3.3: SMA application on the multimedia system.....	91
Table 3.4: SMA application on the phone system	92
Table 4.1: MAT Application on the control panel.....	127
Table 4.2: MAT Application on the conveyor	128
Table 4.3: MAT Application on the robot-in.....	128
Table 4.4: MAT Application on the robot-out.....	129
Table 4.5: Assessment results	129
Table 5.1: Comparisons of decision-making approaches	140
Table 5.2: Pairwise comparison scale for AHP preferences	143
Table 5.3: Random consistency indices	145
Table 5.4: Pair-wise comparison matrix of alternatives with respect to FC	151
Table 5.5: Pair-wise comparison matrix of alternatives with respect to CRC....	152
Table 5.6: Pair-wise comparison matrix of alternatives with respect to TTL.....	152
Table 5.7: Pair-wise comparison matrix of alternatives with respect to TET.....	153
Table 5.8: Pair-wise comparison matrix of alternatives with respect to the 'importance' (E1).....	154
Table 5.9: Pair-wise comparison matrix of alternatives with respect to the	

‘complexity’ (E4).....	155
Table 5.10: Squared matrix of alternatives with respect to the ‘importance’ (E1)	155
Table 5.11: Normalised matrix and eigenvector of alternatives with respect to the ‘importance’ (E1).....	156
Table 5.12: Integrated ranking of alternatives with respect to all sub-criteria (geometric mean)	157
Table 5.13: Pair-wise comparison matrix and eigenvector of the main criteria with respect to the decision goal (E2, Scenario 1)	159
Table 5.14: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘test adequacy’ (E2, Scenario 1)	159
Table 5.15: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘test cost’ (E2, Scenario 1).....	160
Table 5.16: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘application domain’ (E2, Scenario 1).....	160
Table 5.17: Integrated local and global weights for Scenario 1 (geometric mean)	161
Table 5.18: Final ranking results (Scenario 1).....	162
Table 5.19: AHP ranking VS experts’ ranking outcomes (Scenario 1).....	163
Table 5.20: Kendall’s and Spearman’s correlation coefficients between the experts’ integrated ranks and AHP ranks (Scenario 1)	163
Table 5.21: Pair-wise comparison matrix and eigenvector of the main criteria with respect to the decision goal (E3, Scenario 2)	164
Table 5.22: Pair-wise comparison matrix and eigenvector of the sub-criteria with	

respect to the ‘test adequacy’ (E3, Scenario 2).....	165
Table 5.23: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘test cost’ (E3, Scenario 2).....	165
Table 5.24: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘application domain’ (E3, Scenario 2).....	166
Table 5.25: Integrated local and global weights for Scenario 2 (geometric mean)	166
Table 5.26: Final ranking results (Scenario 2).....	167
Table 5.27: AHP ranking VS experts’ ranking outcomes (Scenario 2).....	167
Table 5.28: Kendall’s and Spearman’s correlation coefficients between the experts’ integrated ranks and AHP ranks (Scenario 2)	168

LIST OF FIGURES

Figure 2.1: The V model of software development cycle.....	16
Figure 2.2: Testing types.....	17
Figure 2.3: Model-based testing with relation to other testing types.....	23
Figure 2.4: Early model-based testing	24
Figure 2.5: Formal model-based testing.....	25
Figure 2.6: FSM model of a traffic system	26
Figure 2.7: TA model of a train system	36
Figure 3.1: Simple lamp controller	60
Figure 3.2: Clock regions.....	63
Figure 3.3: Regions with one clock	66
Figure 3.4: Regions with two clocks.....	67
Figure 3.5: Regions with three clocks.....	69
Figure 3.6: Two-clock automaton.....	70
Figure 3.7: Feasible clock regions	71
Figure 3.8: Algorithm 1	75
Figure 3.9: Algorithm 2	77
Figure 3.10: Generated test cases.....	78
Figure 3.11: Lamp controller automaton	87

Figure 3.12: Multimedia automaton.....	87
Figure 3.13: Phone automaton	88
Figure 3.14: Fault detection ratio of the timed testing approaches with respect to mutation operators.....	94
Figure 3.15: Overall fault coverage of the timed testing approaches	95
Figure 4.1: GeTeX chain structure.....	104
Figure 4.2: XML DOM tree of PA test suite	105
Figure 4.3: GeTeX packages.....	107
Figure 4.4: GeTeX GUI	108
Figure 4.5: GeTeX test generation engine outcomes	110
Figure 4.6: Actions/CAN messages convertor.....	111
Figure 4.7: A part of the test suite log file	112
Figure 4.8: Production-cell physical layout	121
Figure 4.9: Production-cell schematic.....	122
Figure 4.10: Load sensor automaton.....	122
Figure 4.11: Unload sensor automaton	122
Figure 4.12: Conveyor load sensor automaton	123
Figure 4.13: Conveyor unload sensor automaton	123
Figure 4.14: Control panel automaton	124
Figure 4.15: Conveyor automaton	124

Figure 4.16: Robot-in automaton 125

Figure 4.17: Robot-out automaton 125

Figure 4.18: AF factor of each testing approach according to production-cell components 132

Figure 5.1: AHP hierarchal model 146

Chapter 1: Introduction

1.1 Topic Overview

Real-Time Embedded Systems (RTESs) have a crucial role in controlling and monitoring modern society infrastructures. Most of them interact closely with their environments such as transportation, air traffic control systems, telecommunication networks and health care devices. Any failures encountered can range from a slight system aberration to financial loss and even loss of human life. As a result, it is necessary to thoroughly test systems to ensure that they are as fault-free as possible before release (En-Nouaary et al., 1998; En-Nouaary and Hamou-Lhadj, 2008; Hessel et al., 2008; Rollet, 2003).

Software testing, a widespread validation method, is a systematic method which aims to increase confidence about software correctness. Different from other validation methods (e.g., verification), testing is based on running software under a controlled environment and analysing its outcomes (Rollet, 2003). In other words, the process of testing relies on providing solid test scenarios (i.e., test cases) that mimic the actual interactions between software and its environments to detect any deficiencies. Testing software with more test cases thus increases the confidence about its quality. However, testing software by all possible interaction scenarios is infeasible due to the infinite space of input data domain. Accordingly, test *adequacy criteria* are used to guide the selection of test cases by which certain properties of software can be examined.

Testing is a complex and expensive validation activity that accounts for approximately 50% of development costs. Many testing approaches and strategies have been developed with the aim of minimising cost and achieving high fault

detection capabilities. One of the most promising approaches is Model-Based Testing (MBT). MBT can reduce test costs due to its ability to capture and validate system behaviour from an early stage of the software development cycle; it also promotes the use of tools to automate the process of test case generation, execution and evaluation (Grieskamp et al., 2011). The process of MBT relies on building models to represent system requirements. These models therefore form an efficient source for deriving test cases and a test oracle. A system's validity can be thus shown by comparing actual system behaviour with the system specification models according to conformance relations (e.g., 'ioco') (Mitsching et al., 2009; Hessel et al., 2008; Tretmans, 1996).

To be a valid source for deriving test cases and capturing software behaviour precisely, specification models have to be formal and rigorous (Beizer, 1990). To formally build specification models and to represent different system behaviour, properties, structures and domains, several formal languages have been proposed and can be categorised as following. First, *finite state-based languages* include those which are capable of presenting system behaviour in a finite set of constructs (e.g., states, transitions, actions etc.). Finite State Machines (FSMs) (Lee and Yannakakis, 1996), Extended Finite State Machines (EFSMs) (Ural and Yang, 1991), Specification and Description Languages (SDLs) (ITU-T., 1997) and Statecharts (Harel and Gery, 1997; Harel and Naamad, 1996) are some examples in this category. Second, *Process algebra languages* such as Communicating Sequential Process (CSP) (Hoare, 1985), Communication and Concurrency Systems CCS (Milner, 1989) and LOTOS (ISO., 1989) can be used to describe system behaviour as a set of concurrent processes. Third, *Hybrid languages* such as Timed Automata (TA) (Alur and Dill, 1994) are used if a System Under Test (SUT) shows hybrid behaviour: continuous behaviour over time and discrete behaviour (e.g., actions). As a result, TA can be safely used for modelling RTES behaviour that interacts with the environment using continuous and discrete signals.

A Timed Automata (TA) (Alur and Dill, 1994) is one of the most widespread formalisms due to its ability to express real-time behaviour of an SUT. It provides

an easy and powerful means of extending finite-state machines with clock variables that track timing progress and incorporate timing constraints through the state-transition graph. The TA comprises a finite set of locations, transitions, actions, clocks and clock conditions to represent system behaviour. Semantically, a TA state identifies the machine location and at which time.

Testing RTEs from TA models can be a complex process due to the requirement of checking timing in addition to functional correctness. Determining correct SUT behaviour relies not only on its correct reactions to test cases, but also on their times (Merayo et al., 2008; Mitsching et al., 2009; Harel and Pnueli, 1985). The process of TA-based testing involves generating test cases according to selection criteria. Test cases are then executed on the SUT (i.e., sent to the SUT to observe its reactions). A suitable timed conformance relation according to which SUT observed behaviour can be compared with the TA specification model is used. If a match occurs, the SUT passes a test case. Otherwise, it fails (Blom et al., 2005; Hessel and Pettersson, 2007b).

The aforementioned themes play an important role in the Thesis chapters and contents. The next section summarises the motivation for conducting this study which leads to the set of stated contributions (Section 1.4).

1.2 Research Motivation

Due to its positive properties, MBT is increasingly used in checking RTEs. Several TA-based testing algorithms have been proposed with the aim of generating few test cases, but with high fault detection capability. They differ from each other in the effort expended in their use, the number of test cases they produce and their effectiveness in detecting logical as well as timing faults (Clarke and Lee, 1997b; En-Nouaary and Hamou-Lhadj, 2008; En-Nouaary, 2008; En-Nouaary and Dssouli, 2003). However, most of these approaches fail to explore the entire state space, are incapable of achieving full coverage, experience the state space explosion problem, come at a high cost in terms of expended efforts and have not been used significantly in tools (Mitsching et al., 2009).

Reviewing TA-based testing methods, a set of observations motivating the research in this Thesis can be made. First, most of the proposed testing approaches rely on generating tests using a random search of the state space or un-timed coverage criteria (e.g., state or transition coverage). In both cases, SUT timing behaviour will not be fully checked. The lack of a definition for a mature timed selection criterion that sets clear rules to select timed test cases is still an issue.

Second, the power of any test suite can be determined by its fault coverage; the higher the fault coverage, the more powerful the test suite (En-Nouaary and Hamou-Lhadj, 2008; En-Nouaary et al., 1999). However, the capability of the proposed approaches to detect potential timing faults has not been fully investigated. In other words, the fault coverage of many approaches has not been measured despite the existence of timing fault models identifying the possible faults that might be encountered. The closest attempt to measure fault coverage discussed the possibility of a testing approach to cover timing faults without any actual measurements (En-Nouaary, 2008; En-Nouaary and Hamou-Lhadj, 2008; En-Nouaary et al., 2002). One of the well-known methods of measuring fault coverage is the application of the Specification Mutation Analysis technique (SMA). To our knowledge, no study has addressed the application of SMA in a TA context. Proposing well-suited mutation operators for TA becomes thus a necessity.

Third, the lack of automation is also noted in a review of relevant literature. Despite the wide number of proposed timed testing approaches, there are few tools to automate the timed test generation. No tools for automating the execution of tests in a real-time context can be found. The absence of automation reduces the possibility of applying the proposed testing approaches due to the difficulties in understanding their mechanism and manual efforts for generating and executing test cases.

Fourth, the software community still lacks serious and detailed industrial applications for validating the proposed timed testing approaches especially for testing SUT timing properties. Although some industrial applications exist, the validated testing approaches are based on testing functional behaviour using an un-timed coverage criterion or random search in generating and executing test cases. In other words, more industrial test beds are still necessary especially for validating the

application of timed testing approaches that focus on testing SUT timing behaviour. The execution of a testing approach in a real-time context induces many problems (e.g., the time synchronisation issue) that still need to be tackled.

Fifth, due to the lack of automation and industrial application, to our knowledge, no study has compared the performance of similar timed testing approaches on real-world applications based on well-identified assessment criteria.

Sixth, the existence of several timed testing approaches leaves the tester with a big decision to make on which approach most suits a testing project. The selection of a candidate testing approach is totally dependent on a tester's intention and experience. In other words, the decision may differ from one tester to another. The existence of factors that contribute to the testing process in different ways increases the complication of making the right decision. This implies a high risk especially for testing safety critical systems. A formal decision-making method by which the consistency in making decisions is guaranteed would be a contribution.

As a result, it is still important to develop techniques that can handle large real-time specifications and generate relatively small test suites with high structural and fault coverage.

1.3 Research Aim and Objectives

Considering the research motivation discussed in Section 1.2, the aim of this research is thus:

To develop, automate and validate a flexible TA-based testing approach based on a timed selection criterion for testing real-time embedded systems.

To fulfil this aim, a number of objectives are necessary:

Objective 1: To introduce a timed adequacy criterion for selecting timed test cases.

Objective 2: To develop a timed testing approach based on the TA formalism and the proposed timed selection criterion for generating test cases divided into different test sets.

Objective 3: To develop a tool for automating the generation and execution of timed test cases.

Objective 4: To evaluate the proposed timed testing approach at the specification and implementation level compared with a set of similar testing approaches based on proposed assessment criteria.

Objective 5: To develop and validate a decision-making framework for the proposed timed testing approach to formalise the selection of the best test set suiting a testing project.

1.4 Summary of the Contributions

The main contributions of the Thesis are:

- 1- The proposal of Clock Region Coverage (CRC) as a timed adequacy criterion for covering timing behaviour of a TA specification. The proposal of the Priority-based Approach (PA) for generating timed test cases from UPPAAL TA (UTA) including its algorithms, according to CRC.
- 2- The validation of PA in comparison with other four similar TA testing approaches based on SMA application. To enable the SMA application, this study proposes timed mutation operators based on the previously proposed timing fault models in the literature.
- 3- The automation of the process of test case generation, execution and report based on PA and the ‘*tioco*’ conformance theory by the development of a new timed testing tool, called GeTeX. GeTeX is validated using a lamp controller prototype modelled as UTA and implemented as one of Controller Area Network (CAN) applications.
- 4- A comparison between the performance of PA and two similar testing approaches using a complete industrial-strength test bed according to proposed assessment criteria. The use of a combined assessment factor that

considers fault coverage, structural coverage (i.e., clock region) and the length of test cases. Fault coverage is enhanced by the application of a Mutation Analysis Technique (MAT) at the implementation level to measure fault coverage of a testing approach.

- 5- The development of an Analytical Hierarchy Process (AHP) decision model for prioritising PA test sets for a particular testing project. The AHP framework is validated using two testing scenarios by examining the degree of match between the AHP decision outcomes and those of testing experts.

1.5 Thesis Outline

The rest of the Thesis is structured as follows.

Chapter 2 emphasises the importance of testing RTEs behaviour. Testing is generally defined and test selection methods are discussed. The chapter also presents an overview of testing types according to the V model and three-dimension model. Testing suffers from a high cost in terms of time, effort and resources. This suggests potential benefits of applying formal methods in a testing context. Formal methods can thus be used to build software specification to be explored and analysed to find any potential faults. The formal specification forms a sound reference according to which the source code can be analysed and validated either by the use of verification or testing (Model-Based Testing (MBT)). Formal languages used to build the software specification are discussed under three categories: Finite state-based languages, process algebra state-based languages and hybrid languages. Checking the match between the SUT and the specification model needs a conformance relation. The chapter thus reviews well-known conformance relations from the literature.

As an important selection and validation method for MBT approaches, the chapter discusses the application of Specification Mutation Analysis (SMA). Due to the continuous and discrete behaviour of RTEs, TA is usually used for building the specification model. As a result, TA has been discussed in terms of conformance

relations and methods used for test selection, generation and algorithms. A set of related work of testing from TA is presented and discussed to highlight the research motivation of this Thesis.

Chapter 3 proposes a new component-based offline test case generation method for RTEs modelled as UPPAAL Timed Automata (UTA). The approach called the Priority-based Approach (PA) is based on a proposed Clock Region as a timed adequacy criterion for generating timed test cases. To enhance the use of clock regions, a set of mathematical equations are defined and proved to calculate the number of clock regions to be covered by test cases. The algorithms of PA are presented and discussed with examples. The chapter also proposes Specification Mutation Analysis (SMA) to validate the performance of PA in comparison with four other timed testing approaches based on TA. A set of timed and functional mutation operators is introduced. Three TA models are used to validate the testing approaches. The validation and comparison processes are based on the mutation score calculated for each chosen timed testing approach with respect to the proposed mutation operators.

Chapter 4 develops and validates a tool for automating the generation and the execution of test cases based on PA. The tool, called GeTeX, can be considered a complete offline testing tool which focuses on checking the correctness of SUT timing properties according to a timed selection criterion. The chapter also runs PA tests on an industrial-strength test bed to validate the performance of PA in comparison with other TA-based testing approaches according to three assessment criteria (fault coverage, structural coverage and the length of test cases). As a result, the chapter presents a set of code-based timed and functional mutation operators to enable the use of the Mutation Analysis Technique (MAT) for estimating fault coverage as one assessment criterion. An assessment factor that combines how many faults are detected and how many clock regions are covered in terms of the length of test cases generated by a testing approach is proposed. A set of lessons learned showing the difficulties encountered especially for testing timing properties is then highlighted.

Chapter 5 develops an Analytical Hierarchy Process (AHP) as a decision-making framework for PA. The framework helps testers select available PA test sets that best fulfil their testing requirements. The AHP framework developed is based on data collected heuristically from the test bed and data collected by interviewing testing experts. The chapter also validates the AHP framework by applying it on two different testing scenarios and comparing the decision outcomes of the framework with those of the experts.

Chapter 6 summarises the research contributions and findings. Finally, the chapter describes the limitations of this study and opportunities for future work.

Chapter 2: Literature Review

2.1 Overview

Modern societies are hugely dependent on embedded systems to monitor or control different hardware infrastructures (En-Nouaary et al., 1998). ‘Embedded system’ is a generic term that refers to computerised systems interacting closely with the real world through sensors, networks and actuators (Broekman and Notenboom, 2003; Hessel et al., 2008). Systems like mobile phones, transportation monitoring systems, air traffic control systems, patient monitoring systems and many others can be considered as examples of embedded systems (Rollet, 2003; Broekman and Notenboom, 2003). Close interactions with the environment induce timing requirements that need to be satisfied for accepted behaviour in the case of Real-Time Embedded Systems (RTESs) (En-Nouaary and Hamou-Lhadj, 2008). For instance, an air bag system should inflate no more than 0.1 second after an accident occurs. Real-time requirements increase the complexity of developing satisfactory RTESs (Hessel et al., 2008; Zheng et al., 2008).

Software is one of the core and most error-prone components of RTESs. Any failures encountered can range from a slight system aberration (e.g., coffee machine malfunction) to financial loss and even loss of human life (e.g., in safety-critical systems) due to the time dependent behaviour. Thoroughly checking the correctness of RTES’s software before deployment using various validation activities (e.g., testing) therefore becomes necessary (En-Nouaary et al., 1998; Mandrioli et al., 1995).

The rest of the chapter is organised as follows. Section 2.2 introduces the concept of software testing. Selecting test cases is the key role of any testing approach. An overview of the most well-known test selection principles is presented in Section 2.3. Different testing methods have been used in the literature. Highlighting some of testing categories according to the V model and three-dimension model is introduced in Section 2.4. To overcome some testing problems such as the high cost, formal methods were used as a complement to software testing (Section 2.5). Formal methods are mainly used to build software specification. The formal specification forms a sound reference by which an SUT is verified or tested. The process of the formal verification is therefore summarised and compared with testing in Section 2.6 whereas Section 2.7 presents the principles of Model-Based Testing (MBT). Different formal languages used to build software specifications, conformance relations and selection methods for MBT are discussed. Due to the continuous and discrete behaviour of RTEs, a Timed Automata (TA) formalism is usually used for building the specification model. As a result, Section 2.8 discusses testing from TA in terms of language properties, abstraction methods, selection criteria, conformance relations and test generation algorithms. A set of related work is also presented and discussed to highlight the research motivation. Section 2.9 concludes the chapter.

2.2 Software Testing

The increasing need to develop high quality software satisfying the requirements of users suggests the need for, and application of, sound engineering disciplines throughout the software development cycle (Abran et al., 2003). The more software deals with aspects of everyday life, the larger and more complex software becomes. Issues related to faults after delivery and failing to satisfy end-user needs are common. For years, it has been thought that delivering software with a minimum amount of faults relies on having a good design and competent programmers. However, experimentation has shown that it is necessary to have a

separate process responsible for checking software correctness, quality and reliability (Pressman, 2010; Briones, 2007).

Testing is a systematic process of finding software errors by running the software in a controlled environment and analysing its outcomes before its deployment (Rollet, 2003). The more test experiments are performed, the more confidence in the SUT's correctness (Dijkstra, 1970). The testing process is a complex and expensive validation activity that accounts for approximately 50% of development costs. One strategy which significantly reduces the test cost is to decrease human involvement and automate the test process through the use of verified testing tools (Hierons et al., 2009; Pinto Ferraz Fabbri et al., 1994; Sugeta et al., 2004; Boehm, 1981).

The process of software testing involves the generation and execution of test cases on software (En-Nouaary, 2008). The generated test cases need to be executed on the SUT to collect the produced outputs. The observed outputs are then analysed and compared with those expected according to a derived test oracle. A test oracle can be defined as the rules by which the expected and actual outputs are compared to decide whether the SUT is correct or not (Utting and Legeard, 2007).

2.3 Test Selection Principles

A test case represents a scenario where input data is applied to an SUT and the consequent outputs are observed. The generation of test cases is based on the software input domain (i.e., all possible input values). If test cases are capable of covering the entire input domain, the SUT is thoroughly tested and a level of confidence about the correctness of the SUT is increased. However, the input domain from where test cases are derived in many cases is significantly large. As a result, generating all possible test cases that cover the entire input domain is costly and infeasible (Utting and Legeard, 2007). For instance, let's consider a program whose main task is to sum two natural numbers $z = x + y$. One of the possible test cases is to set $x=1$ and $y=2$. Here, the input domain represents all sets

of natural numbers (i.e., $x, y \in \mathbb{N}$). The input domain in this example is clearly infinite which makes the generation of all possible test cases impossible.

To obtain a finite number of test cases without badly affecting their fault detection capabilities, a set of test selection hypotheses have been proposed and followed in the literature. Some, called *uniformity hypotheses* (Gaudel, 1995), presume that an SUT shows uniform behaviour under a subset of the input domain. *Uniformity hypotheses* can be interpreted in different ways. One states that if an SUT correctly behaves for some values within a certain input subset, the SUT will behave similarly for the rest of its values. Another states that if an SUT correctly behaves under some of input values that trigger a certain path, the SUT will behave similarly for all input values that trigger that path. Another class of hypotheses, called *regularity hypotheses* (Gaudel, 1995), imply that an SUT shows regular behaviour when the input data size increases. In other words, if an SUT correctly behaves for data whose sizes are 1, 2 and 3, it will show the same correct behaviour for all data sizes. These hypotheses help in replacing the large number of test cases by few useful representatives to test the SUT. However, identifying those representatives is still an issue.

2.3.1 Test Selection Strategies

Based on the test selection hypotheses, several strategies have been proposed to help select representative test cases. Firstly, *equivalence partitioning strategies* (Beizer, 1990; Broekman and Notenboom, 2003) involve dividing the input domain into a set of equivalence subdomains forming the source of the test case derivation process. Each subdomain comprises a set of input data for which an SUT shows uniform behaviour. In other words, all input data belonging to a certain subdomain has an equal opportunity of detecting the same fault. As a result, selecting one or some representative values from each subdomain can be considered sufficient to derive efficient finite test cases.

Secondly, *boundary value analysis strategies* (Broekman and Notenboom, 2003) can be considered complementary to the equivalence partitioning strategies. The boundary values or their neighbours are selected for deriving test cases since boundary values are more likely to be a fault-prone. Let's consider the predicate $1 < x < 3$ as an example. The possibility of replacing an operation type (e.g., ' \leq ' for '<') or changing a boundary value (e.g., 5 instead of 3) is a fault which is more likely to occur while coding than any other.

Thirdly, *adequacy criteria selection strategies* guide the selection of test cases to satisfy an adequacy criterion (Rapps and Weyuker, 1985). The adequacy criteria proposed can be divided into two main categories, namely structural and fault.

Structural adequacy criteria are used to select test cases that cover the structural properties of an SUT such as statements, conditions or branches. For instance, in statement coverage, test cases are generated based on selecting input data that executes each statement at least once. In the indicated example (i.e., that adds two natural numbers), a single test suffices to achieve statement coverage.

Fault adequacy criteria are used to guide the selection process of test cases to detect a pre-defined set of faults injected into an SUT. The test strategy based on fault adequacy criteria is called Mutation Analysis Technique (MAT) (Lipton, 1971; Jia and Harman, 2010).

2.3.2 Mutation Analysis Technique (MAT)

MAT was proposed to increase the confidence about SUT correctness. It is based on simulating real faults in an SUT to either validate or identify adequate test data capable of revealing such faults (Andrews et al., 2005). The process of mutation analysis is based on two hypotheses. First, the Competent Programmer Hypothesis (CPH) (DeMillo et al., 1978) which states that programmers are capable of 'almost' producing correct programs. As a result, programs developed by competent programmers will suffer only from simple syntactical faults. Second, the Coupling Effect (CE) states that 'test data that distinguishes all programs

differing from a correct one by only simple errors is so sensitive that it would also implicitly distinguish more complex errors' (DeMillo et al., 1978). In other words, the test suite that is capable of revealing a fault represented by a single syntactical change to the SUT can reveal more complex faults represented by any combination of such syntactical changes.

The process of MAT comprises three main stages: mutant generation, mutant execution and mutation adequacy analysis. Mutants (i.e., faulty versions of an SUT) are produced by syntactically changing the SUT according to the rules given by *mutation operators*. Each mutation operator is thus linked with the fault that is to be revealed in the SUT. The generated mutants are called first-order mutants. In the second stage, the generated mutants are executed using a given test suite. If a mutant shows different behaviour from the correct version of the SUT, the mutant is *killed* and the fault identified. Otherwise, the mutant is said to be *alive*. In other words, the test suite is not capable of killing the mutant because the test suite is not able to detect the fault or the mutant is equivalent to the SUT. The equivalent relation implies that the SUT and the generated mutant should show same behaviour for the whole set of the input domain. A mutation analysis oracle seeks to achieve a high mutation adequacy score (DeMillo, 1980). The mathematical representation of the test suite adequacy score is given by Equation (2.1).

$$Adequacy\ Score = \frac{\text{no of mutants Killed by a test suite}}{\text{total no of generated mutants} - \text{no of equivalent mutants}} \quad (2.1)$$

On the other hand, MAT encounters difficulties; large amount of human effort would be needed to generate and analyse large numbers of mutants. Moreover, the identification and elimination of equivalent mutants is an un-decidable problem. Literature suggests several solutions to reduce the cost of generating mutants and the identification of equivalent mutants. With regards to reducing MAT cost, several techniques have been used such as mutant sampling, mutant clustering and selective mutation. *Mutation sampling* (Acree, 1980) reduces MAT cost by

randomly choosing a small subset of generated mutants to be executed. *Mutant clustering* (Hussain, 2008) involves selecting mutants according to a clustering algorithm. *Selective mutants* (Mathur, 1991) can also be applied by reducing the number of mutation operators used. With respect to eliminating equivalent mutants, several techniques have been used such as avoiding operators that may generate them, using compiler optimization techniques (Baldwin and Sayward, 1979), constraint solving (Offutt and Jie, 1996) and program slicing techniques (Harman et al., 2001).

2.4 Testing Types

Different types of testing can be categorised in terms of software development stages according to the V model (see Figure 2.1).

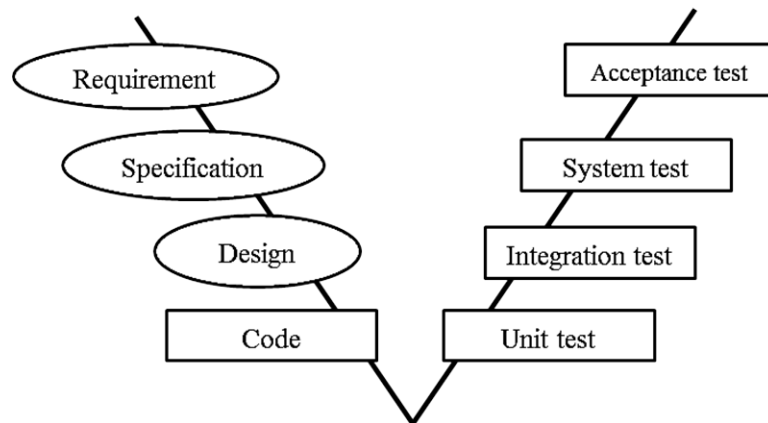


Figure 2.1: The V model of software development cycle (Hierons et al., 2009)

The V model highlights the source information available for each test activity. *Unit testing* relates directly to the code whereas *integration testing* depends on the design information that identifies the available connections between SUT units and components. In order to test the SUT as a whole, *system testing* is used according to an available specification. Being confident about SUT behaviour as a whole is not enough. *Acceptance testing* should be used to check whether the developed SUT satisfies user requirements. Contributing to finding faults early in

the software development cycle, the V model correlates various testing activities along with the development activities (Ammann and Offutt, 2008).

Moreover, different test types can concentrate on various SUT aspects and can be performed at several levels to increase the overall confidence about its quality. Figure 2.2 depicts different types of testing categorised in three dimensions (i.e., testing level, testing accessibility and testing aspects). Note that different types of testing can be performed together (Briones, 2007).

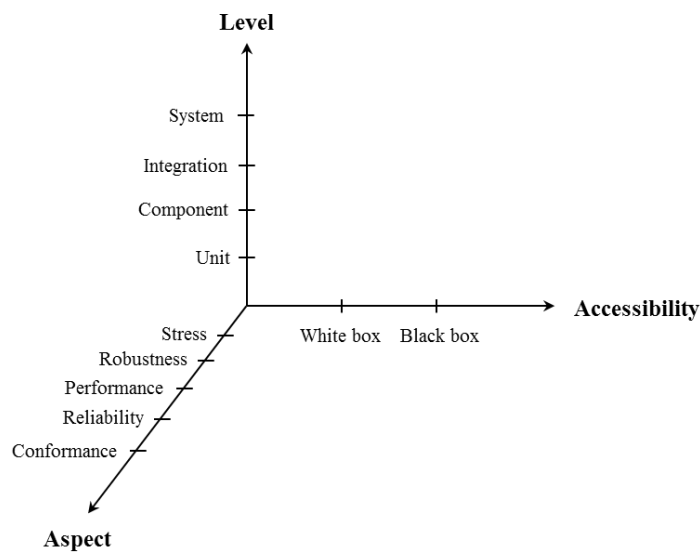


Figure 2.2: Testing types (Briones, 2007)

With respect to which level of the SUT testing is applied, four types of testing can be identified: unit, component, integration and system-based testing. *Unit testing* checks the correctness of the smallest unit of the SUT alone (e.g., a procedure, function or method). *Component testing* concentrates on testing each subsystem individually. *Integration testing* checks the working order for a set of correct components interacting with each other. To check if the system works correctly as a whole, *system testing* is used (Briones, 2007; Utting and Legard, 2007).

In addition to identifying which abstract layer of the SUT needs to be tested, deciding which aspects of the SUT are to be fully checked is equally important. Several testing types have been proposed that cover different aspects of the SUT

such as stress, robustness, performance, reliability and conformance. *Stress testing* checks if the SUT has consistent behaviour under a heavy load. *Robustness testing* involves investigating the reaction of the SUT under unexpected circumstances such as inputs being out of range or hardware failure. *Performance testing* checks the execution time of tasks performed by the SUT. *Reliability testing* ensures that the SUT is almost fault-free before its deployment. Finally, *conformance testing* aims at testing the functionality of the SUT to determine whether its behaviour conforms to that specified (Briones, 2007; Utting and Legeard, 2007).

The third axis in Figure 2.2 shows two types of testing (white box and black box) used according to the SUT visibility to the tester. *White box testing* is used to test the internal structure of the SUT whose algorithms and code are visible to the tester. Test cases are then designed using the information available about the SUT internal structure using different test selection methods (Section 2.3). White box testing is supported by a Control Flow Graph (CFG) which graphically represents the code through its notations. As a result, test selection criteria can be complemented through the use of CFG. The oracle problem of white box testing concentrates on checking the correctness of SUT implemented behaviour at various levels such as unit-based or system-based. However, white box testing fails to check SUT behaviour according to a reference specification (Ferrante et al., 1987; Briones, 2007; Utting and Legeard, 2007).

On the other hand, *black box testing* involves testing the functionality of the SUT according to a reference specification. The SUT internal structure (e.g., code) in black box testing is not visible to the tester. The specification forms the source from which test cases are generated. Test cases are then sent to the SUT which emits output sequences. Several test selection strategies can be used in the case of black box testing such as adequacy criteria (e.g., state or transition coverage). In contrast to white box testing, black box testing is effective in testing SUT behaviour according to the specification but cannot guarantee whether SUT internal behaviour is correct (Briones, 2007; Utting and Legeard, 2007; En-Nouaary et al., 2002).

2.5 Formal Methods in Software Testing

Formal methods are based on mathematics and rigorous logic in building sound artefacts (Bowen et al., 2002). Software testing and formal methods can complement each other in several ways. Instead of using a natural language, building the software specification using formal methods helps to remove ambiguity and assert expected behaviour of the developed software. Faults inherited from the specification during the software development due to misunderstanding of expected properties and functionalities can be thus reduced. Building a software specification using formal methods might be costly and time consuming. However, the cost will be repaid by reducing need to redevelop the software if it does not match user requirements (Hierons et al., 2009). Moreover, a formal specification can be explored and analysed to find any potential faults that might be encountered during subsequent software development activities. Remedying faults at an early stage of the software development cycle usually has a significant effect in reducing overall development cost (Kemmerer, 1985).

A formal specification forms a sound reference according to which source code can be analysed and validated either through the use of proofs or testing (DeMillo et al., 1979). Due to their capability of capturing SUT behaviour, well-defined formal models can be used to represent software specifications. Models can contribute to the generation process of test cases, form the basis of a test oracle and enhance test automation. This type of testing, called *model-based testing*, used to check whether SUT behaviour conforms to the specification can be cost effective (Hierons et al., 2009; Briones, 2007; Nicolescu and Mosterman, 2009).

Although helping to express and understand abstract behaviour of software to be developed, a formal specification does not guarantee correctness. Issues related to a mismatch between user requirements and specification models or failing to satisfy certain modelling properties can negatively affect the creation of formal specifications. *Formal verification* can be thus used to detect such issues (Hierons et al., 2009; Briones, 2007).

2.6 Formal Verification

Formal verification of the software specification is a static validation activity that performs a complete analysis on entire specification models using various mathematical logic. One of the most widely used approaches for software verification is *Model checking* (Clarke et al., 2000).

2.6.1 Model Checking

Model checking is a verification technique often used for concurrent systems. It is based on using various axioms such as temporal logic model checking by which properties are constructed and automatically checked over the specification models (Clarke et al., 1986; Queille and Sifakis, 1982). This approach is widely supported by automated tools known as model checkers such as SPIN (Holzmann, 2003) and UPPAAL (Behrmann et al., 2004). The specification model and a verification property are fed to a model checker to detect whether the specification model satisfies that property. *Temporal logics* (Wolper, 1981; Bradfield and Strling, 2001; Emerson, 1990) are mathematical-based languages used to define verification properties the specification model has to satisfy. The most widespread temporal logics used are LTL (Pnueli, 1977) and CTL (Emerson and Clarke, 1982). LTL is a linear-time temporal logic which defines properties to be checked over the entire execution paths of the specification model. CTL is a branching-time temporal logic which allows properties to be expressed over the entire set of execution paths. Tool support allows these properties to be checked.

The verification process thus aims to increase the correctness of the formal specification by asserting certain properties to be satisfied such as reachability, safety and liveness. *Reachability* is considered as the simplest property that any specification model has to satisfy. This property ensures that every state defined within the specification model is reachable. As a result, a deadlock where the system remains in one state for unlimited time should be detected. *Safety properties* assert that the system will never express a faulty scenario. For instance,

for a temperature controller system, a safety property checks the possibility of a temperature variable in the specification exceeding a specified limit according to the requirements. *Liveness properties* ensure that sets of correct behaviour will eventually happen (Behrmann et al., 2004; Bouyer, 2009).

One of the main challenges facing model checkers is the increase in the complexity of developed systems. Increased complexity leads to more expressive specification models (i.e., to represent all expected behaviour). As a result, those models grow in size. Checking such models may suffer from the state explosion problem where insufficient memory to store all possible states is available (Hierons et al., 2009; Bouyer, 2009).

Such issues can be avoided in several ways. Model checkers use different data structures which makes data retrieval easy and fast. For instance, a Binary Decision Diagrams (BDD) (McMillan, 1993) data structure is used in the NuSMV model checker (Cimatti et al., 1999) and a Difference Bound Matrix (DBM) (Bouyer, 2009) is used in UPPAAL (Behrmann et al., 2004). *Partial order reduction* (Godefroid, 1997) is another technique used to reduce the search space of model checkers by identifying the independence of executed events resulting in the same state. Several *Model abstraction* techniques can be also used to reduce computation complexity. One is based on reducing the number of variables used by resetting the variables when they are not in use or changing their types (e.g., integers to Booleans). Another technique is based on compressing the specification model by using symbolic representation and can be used when it is impossible to handle large systems comprising large numbers of properties (Burch et al., 1992).

2.6.2 Formal Verification and Testing

While formal verification is a static validation activity that checks a specification model exhaustively, software testing is a dynamic validation activity where the SUT is executed within a real environment. Automated verification can complement the process of software testing. The use of automated verification

tools such as model checkers has started to be used for automatic software testing (En-Nouaary et al., 1999; Mandrioli et al., 1995).

Model checkers check whether a specification model of state-based systems satisfies certain temporal properties. If a property has been violated, a counter-example is produced. A counter-example represents the correct path suggested by the model checker where the temporal logic holds. Counter-examples can thus represent test candidates. Model checkers can be forced to produce counter-examples automatically using their search algorithms (e.g., reachability analysis). Accordingly, test cases are generated by feeding model checkers with ‘false’ temporal properties (Hierons et al., 2009; Clarke et al., 2000).

Test case generation using model checkers has been supported by the use of several techniques. One is based on deriving temporal properties in a structural way according to proposed testing purposes (Clarke et al., 2000). Another can derive counter-examples (i.e., test cases) that satisfy coverage criteria such as state and transition coverage (Hong et al., 2001; Hong et al., 2002; Hong et al., 2003). In addition, mutating the specification model to derive counter-examples is another technique for testing based on model checkers (Ammann et al., 1998).

Generating test cases using model checkers however suffers from several problems. Writing temporal properties for model checkers is still a manual process that consumes significant amounts of time. Moreover, the testing process suffers from the state explosion problem especially when the model size grows exponentially (Clarke et al., 2000; Hierons et al., 2009).

2.7 Model-Based Testing (MBT)

The use of models to formally represent a specification reduces ambiguity and helps for a better understanding of SUT behaviour. The process of building specification models has to be formal and rigorous to precisely capture SUT behaviour (Beizer, 1990). As a result, several formal languages are used such as Z

(Spivey, 1992) and B (ABRIAL, 1996) that define a set of constructs and operators to represent SUT properties.

Formal specification models are the source for software development and testing. When test cases and a test oracle are derived from the specification model, the test process is termed as *Model-Based Testing* (MBT). The process of model-based testing can cover various testing activities at different dimensions as depicted in Figure 2.3.

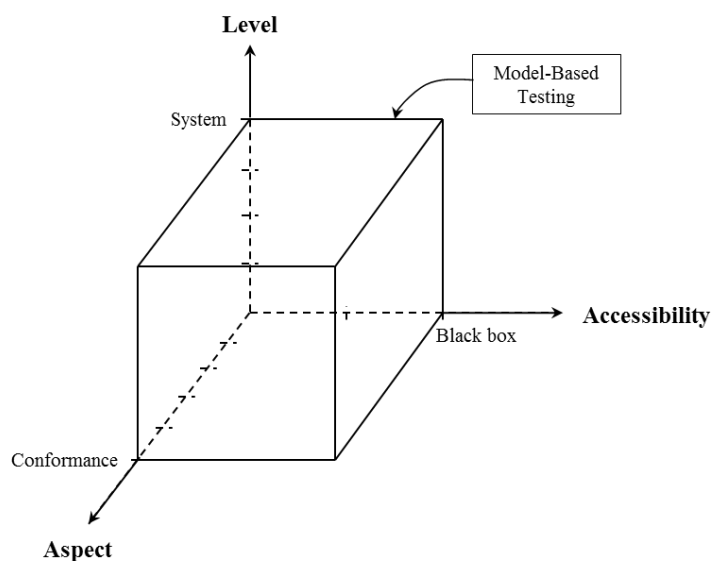


Figure 2.3: Model-based testing with relation to other testing types (Briones, 2007)

MBT is considered as a form of black-box testing since test cases are generated from the specification model without accessing the implementation. MBT can also be used at any software level (e.g., component, integration or system). However, testing at the system level can be considered the most common use for MBT. Moreover, using MBT for testing other software aspects such as robustness is possible. The rationale for adopting MBT, however, is to examine conformance between SUT functional behaviour and a reference specification model (Utting and Legeard, 2007; Briones, 2007).

The use of MBT can be also clarified with connection to software development as shown in Figure 2.4.

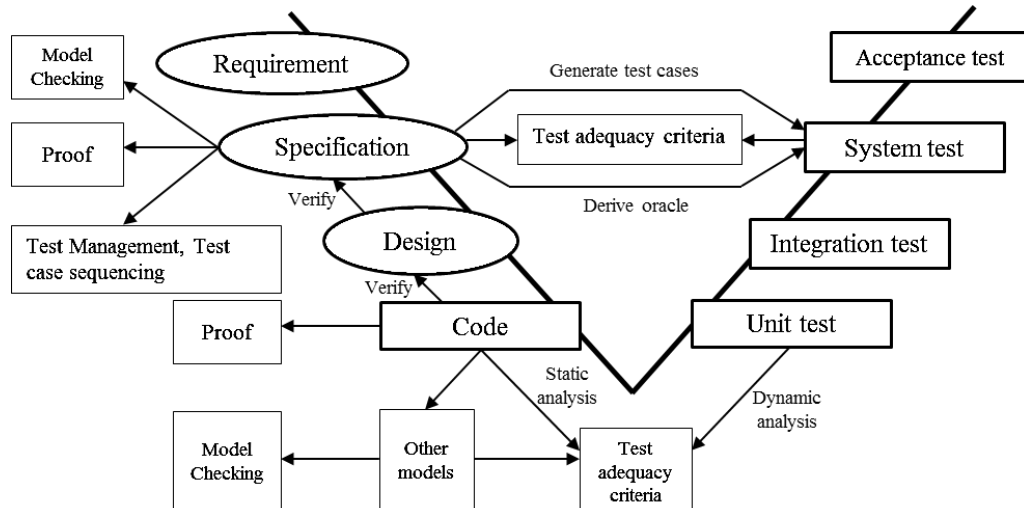


Figure 2.4: Early model-based testing (Hierons et al., 2009)

The V model clarifying the main milestones of software development can identify the possible MBT processes that can be performed. When the specification model has been developed, it should be validated either by a proof of correctness or the application of verification rules. Test cases can then be derived from the validated specification by using one of the test selection methods such as test adequacy criteria. The generated tests are executed at the system level to detect any missing behaviour according to the test oracle derived from the specification. The test process can thus be managed at the specification level. The software design and the implementation code can be verified according to the specification by building an execution model of the code and suggesting coverage criteria. The execution models can be verified using model checkers (Hierons et al., 2009).

MBT is deployed with the aim of achieving high fault detection capabilities and minimising cost through early capture of system behaviour and the automation of test case generation, execution and evaluation. Test cases are generated from the specification and executed on the SUT. A system's validity can thus be checked by comparing actual system behaviour with the formal semantics representing the

system specification according to a conformance relation as shown in Figure 2.5 (Grieskamp et al., 2011; Mitsching et al., 2009; Hessel et al., 2008).

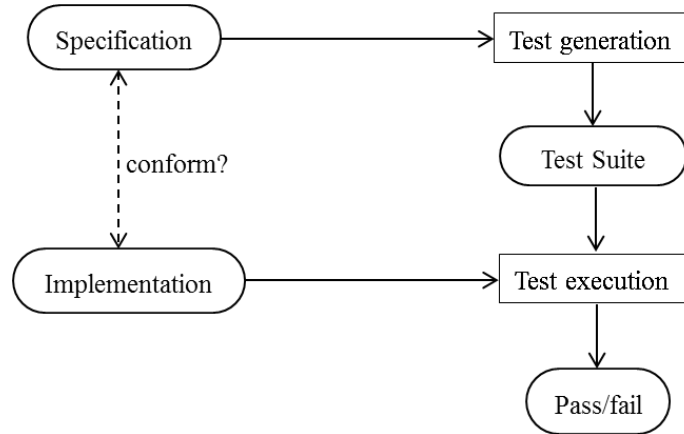


Figure 2.5: Formal model-based testing

2.7.1 Specification Formal Languages

Building specification models precisely is considered a key factor in MBT. The application of formal methods helps us to propose formal languages for accurately representing the specification. The syntax of such languages can be textual or graphical. Several languages are proposed to cover the variety of SUT behaviour, structure and domain; the most popular languages and their use in testing are introduced in the following subsections.

2.7.1.1 Finite State-Based Languages

Finite state-based languages were necessary for presenting SUT behaviour in a finite number of states. Languages such as Finite State Machines (FSMs) (Lee and Yannakakis, 1996), Extended Finite State Machines (EFSMs) (Ural and Yang, 1991), Specification and Description Language (SDL) (ITU-T., 1997) and Statecharts (Harel and Gery, 1997; Harel and Naamad, 1996) can be represented graphically (e.g., direct graph (Aho et al., 1991)) and uses finite sets of constructs to model system behaviour. In this subsection, we will discuss two widely used languages in terms of testing: FSM and EFSM.

FSM is a formal modelling language widely used to capture control behaviour of an SUT. An FSM comprises a finite set of constructs such as states, transitions, input and output actions to represent system behaviour. The FSM specification model has an initial state from which all operations start. The existence of transitions connecting states is necessary to move the machine from one state to another. A transition is fired when an input action is applied to the machine. An output action is accordingly produced and the machine moves to another state. For instance, let's consider the FSM specification model of a traffic controller system presented in Figure 2.6.

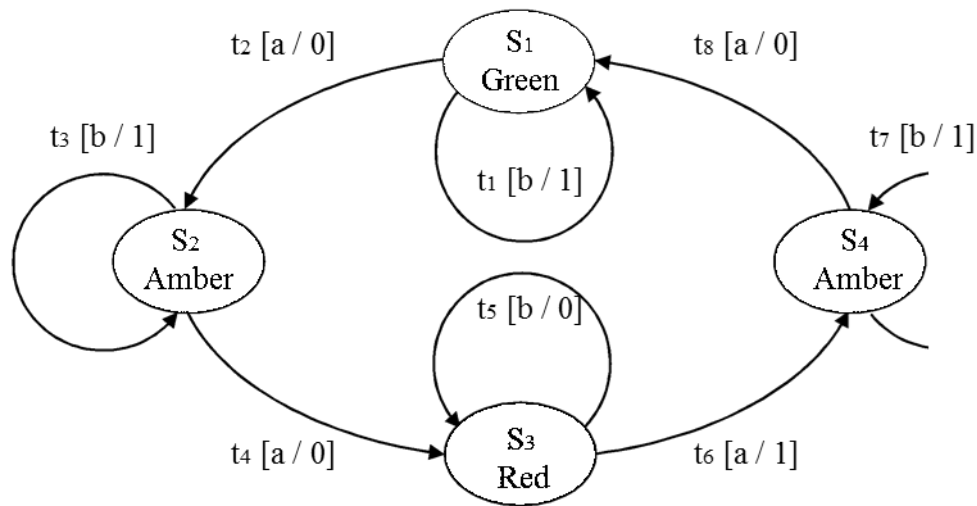


Figure 2.6: FSM model of a traffic system (Kalaji, 2010)

The machine consists of four states, eight transitions, two input actions and two output actions. Note that each transition has a label representing a sequence of input/output. The machine moves from the state S_1 to S_2 by applying an input a . A transition t_2 will accordingly be triggered and an output '0' emitted.

An FSM has several properties that need to be considered for the testing process. To begin with, an FSM is said to be *deterministic* if only one transition can be fired by an input action regardless of the state the machine is in. On the other hand, an FSM can be *non-deterministic* when more than one transition can be fired by the same input action at a state. In addition, for every state in an FSM, if a transition is fired due to the application of an input action, the FSM is said to be

completely specified. Otherwise, the FSM is described as *partially specified* when not every input action can fire a transition from every state. The FSM is said to be *initially connected* when any state can be reached from the initial state by the application of an input sequence. The FSM can also be *strongly connected* when any state can be reached from any other state in the machine. Moreover, the FSM is *minimal* when it is not possible to replace it with an equivalent machine with fewer states. Finally, two states of an FSM are said to be *distinguishable* if two different outputs sequences can be produced as a result of applying the same input sequence on both states. Otherwise, the two states are *equivalent*.

The process of MBT based on FSM might concentrate on testing a specific transition by following three main steps. Firstly, an input sequence has to be applied to reach the source state of the transition that needs to be tested. Adequacy criteria can be used to guide the selection process of suitable input sequences. Secondly, the tested transition has to be triggered by the application of a suitable input to enable the tester to observe the resulting output. If the produced output does not match that expected, a fault is detected. Thirdly, the destination state has to be verified to check whether it is the correct one. A *reset* function that brings the machine back to the initial state is necessary to enable testing another transition (Kohavi, 1978; Rivest and Schapire, 1989; Hierons, 2004; Bouquet and Legnard, 2003).

Detecting output faults is straightforward as it depends on the tester's observations. However, a state fault (i.e., transfer fault) would be more difficult to detect. *State identification techniques* have been proposed to verify the machine states such as Distinguishing Sequences (DS), Characterisation Sequences (W-method) and the Unique Input Output (UIO) method.

DS (Gonenc, 1970) is an FSM-based testing method that looks for an input sequence for identifying each state of the machine. However, it is not guaranteed to find that sequence for some states. The W-method (Chow, 1978) is another method to find state identification sets. This method suffers from long test execution time due to firing the same transition several times for every input in

the set. UIO (Sarıkaya and Bochmann, 1984) also comprises a set of input sequences for identifying each state of the FSM by producing different output sequences.

The main purpose of testing from an FSM is to test control behaviour of an SUT. However, SUT behaviour cannot be merely restricted to an interaction between sets of inputs and outputs. When SUT behaviour requires data to be presented, an EFSM can be considered a better choice for formally modelling the specification. An EFSM thus extends FSM with the use of variables, conditions and operations defined on them. Two different types of variables are used in an EFSM. *State variables* store the logical state such as idle. *Context variables* store the actual data such as ID number.

Triggering a transition in an EFSM requires both an input action to be supplied and conditions of context variables to be satisfied. As a result, the machine will move to another state, an output action will be emitted and an operation on variables will be executed. Four types of transitions can be identified: spontaneous, non-spontaneous, conditional and unconditional. *Spontaneous* transitions do not require an input action to be fired while *non-spontaneous* transitions do. *Conditional* transitions have guards that need to be satisfied for triggering, while *unconditional* ones do not. An EFSM is said to be *deterministic* if at any state there is no possibility for more than one transition to be triggered. Otherwise, an EFSM is said to be *non-deterministic*.

Testing from an EFSM is commonly based on deriving test cases according to test adequacy criteria such as state coverage, transition coverage and path coverage (Tahat et al., 2001). In *state coverage*, selected test cases should cover each state of an EFSM at least once. Similarly, *transition coverage* ensures that generated test cases cover each transition of an EFSM at least once. *Path coverage* also generates test cases that cover all possible paths in an EFSM at least once and is restricted to the models that do not have self-loops. Otherwise, the number of paths will be infinite.

Several problems can be encountered while testing from an EFSM. A *Feasibility issue* is one problem. To achieve test adequacy criteria, a suitable set of inputs that satisfy transition predicates is required for triggering a set of transitions (i.e., Transition Paths (TP)). However, not all TPs are feasible for triggering. For instance, a transition in a TP can update a variable once it is triggered in a way makes it unable to satisfy its condition on the following transition in that TP. Finding a feasible TP can be considered un-decidable problem (Dssouli et al., 1999). Another problem in testing from an EFSM is finding suitable test cases (i.e., input actions) that trigger the feasible TPs once identified (Ural and Yang, 1991). One way to overcome this problem is to abstract the data by transforming an EFSM model to a corresponding FSM for generating test cases. Other issues associated with this solution can be identified. For instance, the large number of resulting FSM states may lead to a state explosion problem (Hierons and Harman, 2004; Hierons et al., 2001).

2.7.1.2 Process Algebra State-Based Languages

Process algebra languages such as Communicating Sequential Process (CSP) (Hoare, 1985), Communicating and Concurrent Systems (CCS) (Milner, 1989) and LOTOS (ISO., 1989) have a rich theory to describe SUT behaviour as a set of concurrent processes.

Testing concurrent systems may use a Labelled Transition System (LTS) language capable of describing SUT behaviour written in process algebra. An LTS supports concurrency in the sense that the specification model is defined by concurrent events. Events in an LTS can be observable or internal (i.e., not observable). Implementation relations (i.e., conformance) are supported by LTS notations that capture SUT interactions with the environment (i.e., traces of inputs and outputs).

An LTS language defines testing as an interaction process between the SUT model and a test case model where both models are represented by LTSs. The test case model maps a state transition system to test verdicts. The set of test cases is called a *test suite*. As a result, different interactions will lead to different test

verdicts (e.g., pass or fail). A *pass verdict* can be assigned if the SUT shows expected behaviour during a test run. Otherwise, a *fail verdict* is assigned. Test cases that might require several runs on the SUT to ensure that the test verdict persists in the presence of internal actions aim to satisfy some desirable properties such as soundness and completeness. A test suite is *sound* if the correct SUT can pass test cases and a faulty SUT can fail some of them. A test suite is *complete* if passing all test cases can ensure that the SUT is correct (Tretmans, 1996; Hierons et al., 2009; Briones, 2007).

2.7.1.3 Hybrid Languages

Most control systems (i.e., embedded systems) deal and interact with various types of signals to control and monitor their environment via a set of actuators and sensors within a real-time context. Continuous behaviour (e.g., time) and discrete behaviour (e.g., actions) should be combined and represented by a single language. Hybrid languages such as Timed Automata (TA) (Alur and Dill, 1994) have been developed to capture such behaviour. More details about TA as a modelling language in general and a source of generating test cases in particular will be discussed in Section 2.8.

2.7.2 Conformance Relations

Determining the testing oracle is one of the most problematic issues that need to be tackled by software testing. MBT is based on conformance relations in deriving the test oracle from the specification. To enable the use of conformance relations assumes that the SUT can be modelled formally in a similar way to the specification. This test hypothesis is necessary to rectify communications between the specification model and the SUT by considering both as formal objects. SUT behaviour is tested by observing its reaction to test cases being applied (i.e., test execution). The sequence of observable actions is called a *test trace*. Several conformance relations to determine whether an SUT pass a test case and accordingly decide whether SUT behaviour conforms to the specification model

have been proposed such as trace preorder, testing preorder, conf and ioco (Hierons et al., 2009; Briones, 2007).

To begin with, *trace preorder* can be considered the simplest conformance relation. It implies a conformance between an SUT '*i*' and specification model '*s*' if the observations as a result of applying a test case '*t*' on '*i*' are a subset of those resulting from the application of '*t*' on '*s*'.

Considered as a more restricting conformance relation, a *testing preorder* is based on the observations made by test cases that eventually lead to deadlock. To clarify, let's denote $tr(t, s)$ as a set of traces that can lead to a deadlock in the specification model '*s*' when applying the test '*t*'. Let's also denote $obs(t, s)$ as a set of the traces that can be observed when applying the test '*t*' on the specification model '*s*'. An SUT '*i*' conforms to a specification model '*s*' iff for every generated test case '*t*', $tr(t, i) \subseteq tr(t, s)$ and $obs(t, i) \subseteq obs(t, s)$. A testing preorder relation cannot be satisfied until all possible test traces are generated and executed, and is considered expensive.

The proposal of the *conf relation* overcomes the disadvantage of the testing preorder. Test traces are generated from the specification model to check SUT behaviour. Let us denote $traces(s)$ as a set of all possible action sequences which can be identified in the specification. An SUT '*i*' thus conforms to a specification model '*s*' iff for every generated test case '*t*', $tr(t, i) \cap traces(s) \subseteq tr(t, s)$ and $obs(t, i) \cap traces(s) \subseteq obs(t, s)$. The conf relation is concerned with detecting any deadlock in the SUT for traces in the specification. In other words, the SUT may have additional traces which add more functionality to the SUT but not controlled by the specification model.

The conformance relations discussed so far have been proposed when communications between the SUT and specification model (i.e., tester model) are seen as synchronized actions. However, passing messages is another type of communications between the SUT and the tester. SUT behaviour will be thus dependent on output messages sent back to the tester. In such a case, the *ioco*

relation has been used to decide SUT correctness according to the specification model. Let $out(i \text{ after } \sigma)$ denote the set of outputs that occur due to the application of a test trace σ on the SUT '*i*'. The SUT '*i*' thus conforms to a specification model '*s*' iff for every generated test trace $\sigma \in traces(s)$ from the specification, $out(i \text{ after } \sigma) \subset out(s \text{ after } \sigma)$. In other words, if the specification model states that an output can (not) be generated after the application of the test trace, the SUT should (not) produce that output (Tretmans, 1996).

2.7.3 Specification Mutation Analysis

The test selection criteria proposed for testing (some of which were mentioned in Section 2.3) can be adjusted and applied for MBT. In this subsection, the application of Mutation Analysis Technique (MAT) in an MBT context as a method for test selection and a method for validity is highlighted and discussed. MAT was first proposed to validate or identify a test suite at the implementation level (white box) with different programming languages such as Fortran (Offutt and King, 1987; Budd et al., 1978), Ada (Bowser, 1988; Offutt and Xu, 1996), C (Untch et al., 1993), (Vilela et al., 2002) and Java (Ma et al., 2002; Ma et al., 2005).

Moreover, MAT has been successfully applied to the design level (Gopal and Budd, 1983; Budd and Gopal, 1985). It has been referred to as Specification Mutation Analysis (SMA). Similar to the original MAT, SMA injects single faults into a specification model by syntactically changing the specification according to pre-defined operators. The generated first-order specification mutants are accordingly executed against a set of generated test traces. The specification mutants are killed if their outputs are different from those of the original specification. SMA is useful in validating MBT techniques by identifying their capabilities of finding faults related to SUT functional behaviour (Budd and Gopal, 1985; Vadim Okun 2004; Jia and Harman, 2010). Different formalisms were then incorporated with SMA such as FSM (Pinto Ferraz Fabbri et al., 1994; Hierons and Merayo, 2007), State Charts (Trakhtenbrot, 2007; Yoon et al., 1998), Petri Nets (Fabbri et al., 1996) and SDL (Sugeta et al., 2004). Since then, research

interest in applying SMA to different specification formalisms has increased to cover Hybrid languages (Aichernig et al., 2010).

To begin with, SMA was used as a test selection criterion for testing embedded systems within a real-time environment (Aichernig et al., 2010). SUT hybrid behaviour was modelled using classical action systems (Aichernig et al., 2009). However, timing behaviour was abstracted away and replaced by temporal orders of discrete states - a drawback of that study. A fault model comprising a set of functional mutation operators was proposed to mutate the specification model; test cases that would kill the mutants were then generated using a conformance checker (Brandl et al., 2010). Pass/fail verdicts were assigned based on the ioco relation.

Several research studies have investigated the application of SMA in the context of FSMs. FSM-based mutation operators were introduced to validate FSM-based specifications (Pinto Ferraz Fabbri et al., 1994). The effectiveness of W- and TT-test methods were compared using a Transport Protocol by calculating a mutation score. A later tool was proposed to support an automatic application of SMA using their proposed mutation operators (Fabbri et al., 1999a). SMA on FSM was extended to Probabilistic Finite State Machines (PFSMs) (Hierons and Merayo, 2007; Hierons and Merayo, 2009). The authors used SMA to show how test sequences that killed mutants were generated. Other work on EFSMs has used SMA to support the test generation process using a model checker. The mutation operators were introduced to the temporal logic level to force the model checker to generate a counter-example (test case) (Ammann et al., 1998).

Several mutation operators have also been produced to support SMA within the context of statecharts (Fabbri et al., 1999b). Other sets of statecharts-based mutation operators were also proposed to assess the quality of generated tests at the specification as well as the implementation level (Trakhtenbrot, 2007).

The Estelle specification language is another formalism taking advantage of SMA. SMA was applied to the Estelle language by introducing a set of mutation

operators. The validation of tests generated from Estelle-based specifications was studied using an Alternating-bit protocol specification model (Souza et al., 1999). A testing technique was also proposed based on the application of SMA on an Estelle-directed Mutation based Protocol Testing (E-MPT). It first generated the mutants from the Estelle-based specification and converted them into C programs using the Estelle compiler. The programs were then executed and the acquired results were compared (Probert and Guo, 1991).

Besides FSM, EFSM, Statecharts and Estelle, several applications of SMA on other specification formalisms exist. SMA was used to measure the effectiveness of the test suite generated from the formal Calculus language (Gopal and Budd, 1983; Budd and Gopal, 1985). Similar work used a refinement class of the calculus language (Aichernig, 2003). An automatic testing approach based on an algebraic specification was also introduced (Woodward, 1992; Woodward, 1993; Woodward and Halewood, 1988). SMA was applied to Petri Net specifications by Petri Net-based mutation operators (Fabbri et al., 1996). A set of mutation operators was proposed for SDL specifications. SMA and its dependent testing approach were illustrated using the Alternating-Bit protocol (Sugeta et al., 2004). SMA was also used for generating test cases from SDL specification (Kov et al., 2003) and validating Lotos-based specifications (Bousquet et al., 2000).

2.8 Timed Automata Based Testing

Testing RTEs is a complex process due to the requirement for checking timing correctness. The number of test cases could be infinite if they are generated and executed within different time intervals. As a rigorous approach, MBT has been used for testing RTEs. Test cases are generated from a reference specification and sent to the RTE SUT. Correct behaviour of the SUT is dependent on its correct reaction to test cases and on their times. In other words, MBT requires testing timing and functional behaviour of the SUT (Merayo et al., 2008; Mitsching et al., 2009; Harel and Pnueli, 1985). The process of timed MBT includes several steps. Firstly, since a specification specifying SUT desired behaviour is responsible for

guiding the testing process, an appropriate formal language capable of capturing real-time behaviour should be used. Secondly, test cases should be generated according to selection criteria. Thirdly, suitable conformance relations according to which real-time behaviour of the SUT is considered correct should be selected and used. Finally, test generation algorithms that automate test cases are also proposed (Blom et al., 2005; Hessel and Pettersson, 2007b).

2.8.1 Timed Automata Specification Language

Timed Automata (TA) (i.e., timed safety automata) (Alur and Dill, 1994) is one of the most widespread formalisms due to its ability to express real-time behaviour of an SUT. It provides an easy and powerful means of extending finite-state machines with clock variables that track timing progress and incorporates timing constraints through the state-transition graph.

A TA comprises a finite set of locations, transitions, actions, clocks and clock conditions to represent SUT behaviour. TA locations represent the position that a machine is currently in. A TA specification model has an initial location where the operations on the model start and its clocks restart. Semantically, a TA can use an LTS to represent TA states that identify the machine location and at what time. A TA thus has an infinite state space. Clock conditions constraining SUT behaviour are used over transitions (i.e., clock guards) or locations (i.e., invariants). *Clock guards* are used to constrain firing transitions. *Location invariants* are applied to assert progress by which the machine is not permitted to stay in a location for an unlimited time. The existence of transitions connecting locations is necessary to move the machine from one state to another. Triggering a transition will require both an action to be supplied and clock guards to be satisfied. As a result, the machine will move to another state and the clock value reset. SUT behaviour is thus shown as sequences of transition executions (i.e., traces) (Hessel et al., 2008; Alur and Dill, 1994; Bengtsson and Yi, 2004). Figure 2.7 presents the TA model of a train system. It consists of five locations, six transitions and one clock. A train informs the gate before approaching it. If the

gate is open, the train will be allowed to cross and leave. Otherwise, the train has to stop and wait for a gate signal. Once that is received, the train is allowed to cross and leave.

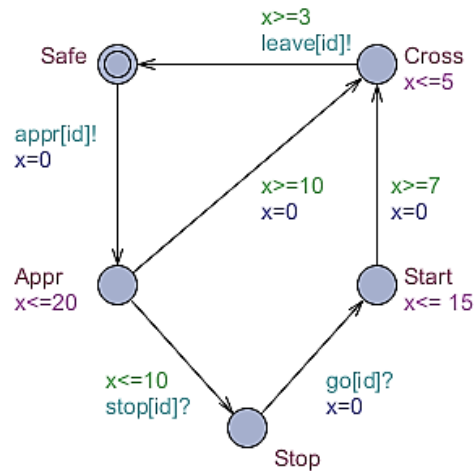


Figure 2.7: TA model of a train system

For instance, the train can move from the ‘Start’ location to the ‘Cross’ location by triggering the transition connecting them. The train has to leave the ‘Start’ location up to ‘15’ time units and the transition can be triggered within ‘7’ time units.

Several classes of TA formalisms have been proposed with different properties representing a wide range of applications. When a TA can classify actions as inputs or outputs, a TA is called as a Timed Input Output Automata (TIOA) (Nicollin et al., 1992; Lynch and Attiya, 1992; Springintveld et al., 2001). Outputs are usually marked with ‘!’ while inputs are marked with ‘?’. TAs can also communicate with other TAs through a range of concurrent clocks and actions comprising a *Network of Timed Automata* (NTA). The TA model of the train system in Figure 2.7 is an example of an NTA by replacing ‘id’ with a certain number (e.g., ‘6’). In other words, a vector of six trains interacting together is so formed. Such an example clarifies the importance of the network representation as it serves real applications such as the train monitoring system. To analyse the NTA, a parallel composition is used to combine all network models into one

single model. The resulting model would suffer from large number of states which increases its complexity and analysis (Hessel et al., 2008).

Moreover, a TA can be extended with special data variables (e.g., integers or Boolean) and certain properties (e.g., urgent channels) as in an UPPAAL Timed Automata (UTA) (Behrmann et al., 2004) can increase TA expressiveness in modelling more applications. One of the main advantages of UTA is the possibility of modelling the SUT environment separately. Identifying the actual interactions between the SUT and its environment can thus reduce the number of test cases generated for a specific environment rather than for all possible environments (Hessel et al., 2008). An Event Recording Automata (ERA) is another class of TA (Alur et al., 1999). Similar to TA, an ERA consists of actions and a set of clocks used to constrain transitions. However, in the ERA model, each clock, called an *event clock*, monitors a unique action called an *event*. The event clock thus measures the elapsed time since the last execution of its event. Once the event has occurred, its clock is automatically reset.

Similar to FSM, a TA specification can be *deterministic* if there is only one transition enabled, regardless of the location the system is in or at which time. Otherwise, the TA is declared as *non-deterministic*. For instance, Figure 2.7 presents a non-deterministic TA model. At ‘Appr’ location, two transitions are enabled at ‘10’ time units. Non-determinism allows flexibility in modelling SUT behaviour but negatively contributes to the test generation process; it is not known how the SUT responds to a test case.

Another class of TA where outputs are *isolated* and *urgent* has been suggested. In this class, it is permitted for only one isolated output to be emitted at any given location. Moreover, urgent outputs, if they exist, can be emitted at no time (i.e., without allowing any time to pass). The expressivity of the TA can thus be affected; a case such as ‘if an SUT receives an input, an output *a* or output *b* can be emitted with certain time’ cannot be presented.

In terms of inputs, a TA can be *input-complete* by allowing any state to accept inputs. In addition, a *non-blocking* TA does not block time even if a TA doesn't receive any input. A TA can be also *fully observable* or *partially observable*. The former uses only actions that can be externally observed to trigger transitions. The latter permits the use of internal actions to increase TA expressiveness. For instance, the transition connecting 'Start' and 'Cross' in Figure 2.7 is triggered by an un-observable action (Krichen and Tripakis, 2005; Krichen and Tripakis, 2004).

The clock variables used in a TA formalism can also be represented by several models such as a discrete-time model, fictitious clock model and dense-time model. Clocks in a *discrete-time model* are represented by integer variables. Clocks defined for a TA run with the same speed. The use of this clock model can be useful in digital circuits where actions are taken just after the arrival of a clock signal. A discrete-time model which approximates the time to the nearest integer would limit the time accuracy especially for very sensitive time-dependent behaviour. A *Fictitious clock model* is similar to the discrete-time model where clocks are represented by a sequence of integer variables. Actions occur in a real-time context but only the upper nearest integer values of clocks are recorded. More naturally, a *dense-time model* considers clocks as real values. Time increases without any bound. Using a TA with a dense-time model complicates the test generation process due to its infinite clock values (Alur and Dill, 1994).

2.8.2 Timed Automata Abstraction Methods

The more we increase the expressiveness of a TA, the more applicable the TA formalism and the more difficult to check its behaviour (especially for non-deterministic ones). For instance, a non-deterministic TA supported with a continuous-time model can be chosen to model an SUT. This would enhance the modelling process but result in an infinite state space leading to the state explosion problem. Choosing the class of TA to model the specification thus has a great impact on verifying or testing SUT behaviour. To avoid the state explosion

problem, several model abstraction methods have been proposed that can reduce the SUT state space without greatly affecting its behaviour, such as regions and zones.

To begin with, defining a proper *equivalence relation* (Alur and Dill, 1994; Larsen and Wang, 1997) enables the classification of equivalent states into groups or *regions*. The proposed relation depends on the fact that several states can be similar in terms of the actions applied and transitions enabled. Each region thus contains all states that make a TA respond with same behaviour. Having regions instead of states, the TA with infinite state space is replaced by Region Automata (RA) with a finite set of regions. The RA serves as a good replacement for the TA for the verification and testing process. However, the equivalence relation partitioning the state space is considered as fine-grained. In other words, the number of regions produced may be very large and lead to the state explosion problem. In fact, the number of regions grows fast with respect to both the number of clocks used in a TA and their upper bounds. For instance, the number of regions in the case of one clock with ‘1’ as an upper bound is ‘8’ regions. However, the number of regions in the case of two clocks with ‘1’ as an upper bound for both is ‘18’ (Bengtsson and Yi, 2004). One of the solutions proposed to overcome this problem is to reduce the number of clocks used in a TA (Daws and Yovine, 1996).

A coarser equivalence relation for partitioning the state space is proposed for an ERA. Similar states are accordingly categorized in equivalent classes from where test cases are generated. The partitioning relation depends on clock valuations of ERA states by which two states belong to one class if they enable the same transitions. The abstracted specification graph preserving all SUT behaviour consists of states representing a set of locations and equivalent classes (Nielsen and Skou, 2003; Briones and Röhl, 2005).

Another abstraction method depends on much coarser partitioning of the state space by forming *zones*. A zone can contain all states satisfying a clock constraint. As a result, a zone which does not depend on the number of clocks and is

represented by DBMs can lead to a more compact model. Replacing the infinite state space with finite zones provides a Zone Graph (ZG) which identifies zones in a symbolic way. Each symbolic state in ZG will thus consist of a location and zone. Similar to the RA, a ZG may also be infinite if clock values are unbounded. To solve this problem, a maximum constant is assigned. All clock values below this constant will be used in zones but the larger values will be disregarded. In other words, further state abstraction is applied to reduce the number of zones (Bengtsson and Yi, 2004; Briones, 2007).

2.8.3 TA Test Selection

Real-time impacts all steps of the testing process. A TA formal specification used to present a real-time specification of an SUT forms the source of test cases. It is not possible to thoroughly test SUT real-time behaviour due to an infinite state space. The correct selection of which parts of the specification to be tested plays a key role in efficiently testing such systems. Several test selection methods are used in testing RTEs similar to those used in testing un-timed systems (e.g., test purposes, structural and fault adequacy criteria) (Hessel et al., 2008).

2.8.3.1 Test Purpose

A test purpose is considered as specific behaviour of the SUT that needs to be fully checked. A test purpose is modelled in several ways. One can be represented as a property to be checked in the specification using model checkers (Hessel et al., 2003). Another might involve representing test purposes as extra flags added to specification models (Hessel et al., 2008). Moreover, a special formal representation can initially be used such as Message Sequences (MSC-2000). The Message Sequences are then converted to suit the formal language used for modelling the specification (En-Nouary and Liu, 2004). Test purposes can also be used as a test selection method for non-deterministic TA models (Bertrand et al., 2011a). However, a method such as a game approach may be required for transforming the non-deterministic model to a deterministic one (Bertrand et al.,

2011b). Using test purposes as a test selection method reduces the number of generated tests because checking of the entire specification is not required. However, generated test cases cannot guarantee efficient fault detections.

2.8.3.2 Structural Adequacy Criteria

Adequacy criteria are often used in testing to assess the level of thoroughness of a test suite. The aim is to measure to what extent test cases cover a specification model. Different types of adequacy criteria are discussed and used for testing un-timed systems. However, the research line concerning timed adequacy criteria in the literature still suffers from immaturity (Hessel et al., 2008). As a result, un-timed adequacy criteria were adopted in selecting test cases from a TA such as location, edge and definition-use coverage criteria (Hessel et al., 2008).

Location coverage selects test cases visiting each location of a TA at least once. *Edge coverage* emphasises the selection of test cases that traverse all transitions in a TA model. Where location and edge coverage target the structural components of a specification model, *definition-use coverage* criterion focuses on the data level. This coverage criterion is suitable for an extension class of TA such as UTA but not for a regular TA model where no data is used. The idea of definition-use coverage is to select test cases that trigger a test path from where a data variable has been defined to where it has been used (Hessel et al., 2008).

2.8.3.3 Fault Adequacy Criteria

The effectiveness of test cases can be measured by their ability to detect major faults in an SUT (i.e., *fault coverage*). Fault coverage is used for measuring the power of derived test cases. As a result, test generation methods can be compared and validated according to fault coverage. However, if fault coverage is used as a basis for selecting test cases, the number of generated test cases will be dramatically reduced. To facilitate this concept, the potential faults an SUT might suffer from should be clearly defined in a *fault model* (En-Nouaary et al., 1999; Wang et al., 2009; Clarke and Lee, 1997b). The fault model is usually consistent

with the specification formal language. In other words, faults are defined using the same formal language as the specification. For instance, a fault model using a constraint graph is similar to the system specification (Clarke and Lee, 1997b).

In a TA, two kinds of faults were defined according to a proposed fault model; namely functional and timing faults. Three types have been proposed in terms of timing faults. Firstly, a *clock-reset fault* occurs in an SUT whenever a clock is reset or not reset in an opposite way to that stated in the specification. Secondly, a *time constraint restriction fault* occurs in an SUT when it narrows down the timing bounds by which it rejects inputs satisfying timing constraints defined in the specification. As a result, the number of states in the faulty model decreases in comparison with those of the original specification. Thirdly, the SUT has a *time constraint widening fault* if it increases the timing bounds by which it accepts inputs not satisfying timing constraints defined in the specification. Accordingly, the number of states in the faulty model increases in comparison with those of the specification. *Functional faults*, on the other hand, occur when an SUT moves to a state which is different from that expected (i.e., *transfer faults*) or responds with missing or incorrect actions (i.e., *action faults*) (En-Nouaary et al., 1999). Some faults do not affect SUT correctness. One reason for this might be due to *fault masking*. In other words, the occurrence of multiple faults even if they can be detected alone, can hide faulty behaviour of an SUT (Batth et al., 2006; Uyar et al., 2005; Wang et al., 2009).

2.8.4 TA Test Generation

Different approaches have been followed in generating timed test cases; namely offline, online and model checking (Hessel et al., 2008). An *Offline testing approach* involves generating all possible test cases using one of the test selection methods prior to executing them on the SUT. In other words, the test generation and execution phases are separate. Adopting an offline test generation method has advantages and disadvantages. Test cases generated can cover several aspects of the specification according to the selection criteria used. As a result, test cases are

cheap, fast and easy to execute as they are selected *a priori*. On the other hand, analysing and covering the entire specification for generation test cases might suffer from the state explosion problem. In the other words, an offline method may not be able to handle a complex and large specification. Moreover, an offline method cannot deal with a non-deterministic specification. Generating test cases is based on searching for all possible paths through the specification model according to a selection criterion. In terms of non-determinism, test cases can be very large and the outputs cannot be predicted. In this case, use of a deterministic class of TA is advised.

In an online testing approach, test case generation and execution processes are performed at the same time. A test case is generated from the specification and directly executed on the SUT; the generated output and its timing are then compared with those in the specification. Another test case is generated and so on until termination of the test is decided or a fault is discovered. In an online approach, the test generator selects test cases from the specification randomly way. Choosing an online approach has several advantages. The possibility of the state explosion problem is dramatically decreased because only one test case needs to be stored before execution. An online approach can also deal with non-determinism as the generation and execution process are completed step-by-step. The test path followed by a test case can be known according to the observed outputs. On the other hand, a random selection of test cases does not guarantee coverage of the entire specification and detecting all faults. The test run in an online approach can continue for hours and even days. As a result, it is difficult to analyse test failure when it occurs and identify its location. An efficient test algorithm is required for dealing with RTESs where time should be accurately synchronised between the test generation and execution processes.

Model checking is a verification method that checks the entire specification model according to some logical properties. Using a model checker tool (e.g., UPPAAL) can provide an easy and powerful technique for searching the state space and thus generate test cases. In addition to producing counter-examples (as discussed in

Section 2.6.2) model checkers can guide the test generation process in combination with other techniques. *Observer*, as an example, is a technique that monitors and guides the model checker in selecting test cases according to adequacy criteria. Each adequacy criterion is represented by an observer that monitors the generation of test cases and replies with an acceptance if adequacy criterion is satisfied (Blom *et al.*, 2005).

2.8.5 Timed Conformance Relations

Several timed conformance relations have been proposed for RTEs to decide on the correctness of their timing behaviour such as Timed Trace Inclusion (TTI), Relativized Timed Input-Output Conformance relation (*rtioco*) and Timed Input-Output Conformance relation (*tioco*).

To start with, TTI (Hessel *et al.*, 2003) is a simple conformance relation used for a restricted class of TA (i.e., deterministic with isolated and urgent outputs). The SUT conforms to the specification *iff* timed traces of the SUT are a subset of those of the specification. In other words, the SUT should not emit an output after an input sequence if the specification does not allow it to. In a similar way, the SUT has to emit an output or delay if the specification allows it to.

For more generic TA models, *rtioco* (Larsen *et al.*, 2005a) has been proposed. *rtioco* was initially derived from and applied the notion of the *ioco* relation. The SUT conforms to the specification if the SUT does not have behaviour not permitted by the specification when taking a given environment into account. In other words, to compare the SUT and the specification, a parallel composition with an environment model is required for both. The SUT should then produce an output at a time when one is required by the specification. No output should be expected from the SUT when it is not permitted by the specification. The notion of *rtioco* extends that of *ioco* by considering time. Moreover, *rtioco* is more generic than TTI since it deals with input-enabled non-blocking specifications

taking the environment into account. It was also used for an online testing approach.

tioco (Krichen and Tripakis, 2009) can be considered as another extension of *ioco*. The conformance between the SUT and the specification can thus occur if observed outputs of the SUT after any recorded behaviour must be part of all possible observable behaviour of the specification. The observable behaviour includes outputs and time delays. The proposal of a generic relation like *tioco* was to deal with a non-deterministic partially observable specification with normal outputs. *tioco* is not as strict as TTI; *tioco* allows the SUT to accept inputs not defined in the specification as long as they do not contradict it.

2.8.6 Related Work

Many algorithms and methods for testing real-time systems from TA have been proposed. However, the majority are based on un-timed selection criteria for generating timed test cases. In addition, only a few have been supported by tools and empirically studied (Hessel et al., 2008).

Blom et al. (Blom et al., 2005) introduced a formalism called ‘Observer Automata’ to monitor and generate test cases offline. Well known un-timed coverage criteria adopted were edge, location, definition-use pair, definition and affect-pair coverage. The formalism was supported by developing an offline model-based test generation tool called CO \checkmark ER (Hessel and Pettersson, 2007a). CO \checkmark ER developed at Uppsala University extended the UPPAAL model checker with coverage criteria expressed by the Observer Automata formalism. Hessel and Pettersson (2007b) took a step further by empirically validating the observer automata based on the UPPAAL model checker using an industrial real-time test bed based on WAP protocol modelled as a NTA. The test bed used CO \checkmark ER to automate the generation of tests and existing tools from Ericsson for automating test execution. The study focused on showing the process of generating and executing test cases according to the proposed approach rather than validating its performance.

Although it reported some discrepancies, the testing approach was based on un-timed coverage criteria for testing timed systems. Besides, CO \sqrt ER is just a test generation tool which needs the assistance of other tools to execute test cases on the SUT.

A UTA was an input language for another MBT approach (Cardell-Oliver, 2000). The generation of timed test cases involved three steps. Firstly, a UTA was transformed into a Testable Timed Transition System (TTTS) (i.e., a deterministic model using a discrete-time model) to capture its timing behaviour. Secondly, a concise choice of test cases was made by the use of *test views* (i.e., test purposes) to explore certain aspects of SUT behaviour. Using test views helped reduce the number of generated test cases. Thirdly, trace equivalence was used as a notion of conformance according to which the SUT can be declared faulty or correct. A fault model was used in the approach to prove its fault detection capability. The approach was also supported by a prototype of test generation tool (Glover and Cardell-Oliver, 1999) which automated the construction of TTTS under different test view scenarios. In spite of generating fewer test cases in comparison with others, the testing approach cannot explore most of or identify missed SUT behaviour. The use of test views reduces the number of tests and thus cost, but does not guarantee SUT correctness. The fault model used in this study considered several functional faults and omits timed ones.

UPPAAL Tron (Larsen et al., 2005b) is another timed testing tool based on the UPPAAL model checker and UTA as an input language. In contrast to CO \sqrt ER, UPPAAL Tron is an online testing tool where test case generation and execution take place at the same time. As a result, the choice of next inputs to be applied on the SUT is determined randomly rather than following selection criteria. UPPAAL Tron has been used in several industrial case studies such as the railway signalling case study (Mitsching et al., 2009), a protocol to help secure DNS (Rütz and Schmaltz, 2011) and the DANFOSS EKC-201 refrigeration controller (Larsen et al., 2005b). UPPAAL Tron consumes significant time to finish and cannot

guarantee to find all faults, especially timed ones due to its random state exploration.

TorX (Fitzgerald et al., 2005) is another online MBT tool that has been extended with time. TorX is based on the timing extension of *ioco* conformance theory including quiescence (i.e., the case of output absence). The drawbacks of the UPPAAL Tron also apply. In addition, it is difficult to represent the idea of quiescence in real-time systems (Krichen and Tripakis, 2009).

Nielsen and Skou (2001) introduced a class of TA (i.e., ERA) supported by a prototype tool called RTCAT. The authors (Nielsen and Skou, 1998) applied a coarse partitioning relation to reduce the state space. Symbolic reachability analysis was then applied on the abstracted model to generate test cases satisfying Hennessy test theory (De-Nicola and Hennessy, 1984). The application of the tool as a timed test case generator was applied to the Philips Audio Protocol. The main drawback of this approach was the complexity of the model used as an input language (especially when the model was large with a high number of clocks). Moreover, it did not guarantee the discovery of all timing faults since it followed a coarser state partitioning class.

A more generic TA specification model permitting non-determinism and internal actions was used as a base for generating test cases based on proposed analogue and digital testing approaches (Krichen and Tripakis, 2005). For the analogue testing approach, a non-deterministic TA supported with a dense-time clock model was transformed online during the execution of test cases to a deterministic TA. Tests were selected randomly and the tester reaction time was reduced. A more realistic approach considered the use of digital clock models to reduce the state space. Offline test generation can thus be supported with several test selection criteria such as edge and state coverage. The process of test generation was automated by developing a prototype tool 'TTG' and validated using a Bounded Retransmission Protocol (Krichen and Tripakis, 2009). The authors claimed that the approach produced few tests. However, no solid validation to their claim was found. In addition, although the analogue approach deals with

non-determinism, online determinism implies risks especially with respect to guiding the test selection process and termination of the tests. The tool also does not support the test execution process.

Other TA-based testing approaches have yet to be automated or empirically validated. In (Springintveld et al., 2001), a theoretical framework was proposed for generating test cases from a Timed Input Output Automata (TIOA) specification based on the W-method (Chow, 1978). The authors admitted that the approach was impractical due to the high number of generated test cases for a simple TA model.

In (En-Nouaary et al., 2002; En-Nouaary et al., 1998; En-Nouaary and Liu, 2004; En-Nouaary et al., 1999), the authors adopted the Wp-method (Fujiwara et al., 1991) for generating timed test cases from a TIOA model. The proposed method relied on sampling (Larsen and Yi, 1993) the specification according to a clock valuation equivalence rule to reduce the infinite state space. A testable automaton, called Grid Automaton (GA), was introduced as a result of the sampling process with the coarsest granularity related to the number of clocks. The timed specification was then transformed to an un-timed one to enable application of the Wp-method. The authors discussed that test cases generated could discover main (known) timing faults. However, the number of test cases generated for a small specification was still large, since the method aimed to cover all states of the produced GA model.

Selecting the granularity for sampling the RA affects the size of the resulting GA and therefore the number of generated tests. As a result, a dynamic selection of the granularity would lead to a more compact GA. This idea was the base of a new testing approach (Bonifácio and Moura, 2011). Tests were generated from the GA using test purposes; a synchronous production of the specification with test purpose models was created. The study did not address the notion of correctness or how the specification model could be covered. Although it is a promising approach, a robust validation along with an automated tool is still needed.

A TIOA supported with a discrete-time model was used for generating timed test cases (Khoumsi et al., 2000). The test generation process involved abstracting time by transforming the TIOA model into an un-timed one to enable the application of the Wp-method for generating test cases. Notably, a large number of test cases are likely to be generated, some of which might be not executable. Another transformation from a TIOA to an FSM has been tried to ease the test generation process. The transformation process produced an equivalent non-deterministic model which might lead to a state explosion problem (Khoumsi, 2002). To avoid this, a TIOA was transformed to a special case of FSM called Set-Exp automata by creating special events for representing timing behaviour (i.e., clocks and their reset). The aim was to use the test generation approaches proposed for un-timed systems for testing real-time ones (Ouedraogo et al., 2010). Another study generated timed test cases from a TIOA (Fouchal et al., 2000). The proposed approach transformed a TIOA model to an un-timed LTS to generate test cases based on a set of test purposes. A pass, fail, inclusive (i.e., when the SUT passes and fails the same test) notions were defined. The approach failed to explore the state space of the specification model. Besides, no validation to the claims of this study was presented.

En-Nouaary (2008) introduced a timed scalable testing approach based on a TIOA. To avoid generating a large number of test cases, the GA produced was traversed by covering each transition just twice at two time points (the earliest and latest) to generate test cases. The approach appeared to be scalable and produced few test cases. However, the traversal of GA to generate test cases was expensive time-wise. In (En-Nouaary and Hamou-Lhadj, 2008), a further method for testing real-time systems modelled as a TIOA was presented. This method generated test cases by covering every transition of the TIOA at three different times (soonest, latest and 'between' two executions). The authors claimed that the approach ensured good fault coverage of the system. However, they did not validate their claim.

A timed test case generation method based on the specification modelled as a temporal logic was introduced (Mandrioli et al., 1995). However, a discrete time model was used to represent timing behaviour. The UIOv-method for generating test cases from a TA was introduced in (Higashino et al., 1999). The authors presented an algorithm for selecting each executable transition. The main drawback of this method was the amount of time consumed by the method.

Other formalisms have been used for timed test generation approaches. Similar to the TA, a Timed Transition System (TTS) was used to model SUT behaviour and generate timed test cases. The model used a discrete-time model to represent timing behaviour. The process of generating test cases involved abstracting timing behaviour by transforming a TTS to a LTS to use the W-method. This approach produced a large number of test cases due to the large state space resulting from the transformation (Cardell-Oliver and Glover, 1998). Clarke et al. (1997b) introduced a testing method based on a constraint graph. Generating test cases (Dasarathy, 1985; Taylor, 1980), the authors proved that their method achieved full fault coverage. However, the constraint graph from which those timed test cases were generated was not general and was restricted to the minimum and maximum delays between two consecutive events.

It is clear that methods that provide higher coverage will tend to produce more test cases. Moreover, the lack of a clearly defined timed coverage criterion in generating the test cases means that the majority of timed state space is omitted. As a result, the capability of the generated test cases for detecting timing faults is still questionable.

2.8.7 Motivation for Automatic Testing from a TA

The application of MBT on RTEs is still considered relatively new and complicated compared with un-timed systems. Timing behaviour which increases the state space to be explored increases the cost by generating a large number of test cases.

Many TA-based test algorithms based on TA were proposed with the aim of generating few test cases but with high fault detection capability. They differ from each other in the effort expended in their use, the number of test cases they produce and their effectiveness in detecting logical as well as timing faults (Clarke and Lee, 1997b; En-Nouaary and Hamou-Lhadj, 2008; En-Nouaary, 2008; En-Nouaary and Dssouli, 2003). However, most of these approaches fail to explore the entire state space and come at a high cost in terms of expended efforts (Mitsching et al., 2009). The used test selection criteria (i.e., adequacy criteria) for generating timed test cases are un-timed. SUT timing behaviour which is not fully checked can hide many faults. Moreover, there are very few tools which automate the generation and execution of test cases despite the wide number of proposed testing approaches; to our knowledge no tools exist for automating the execution of tests in real-time contexts.

As a result, developing techniques that can handle real-time specifications and generate relatively small test suites with high structural and fault coverage is still necessary. Adoption of an efficient timed adequacy criterion is thus an urgent need. Automating the process of generating and executing test cases is also a high priority for reducing time and cost.

2.9 Summary

Computer-based systems have an increasing role in controlling and monitoring modern society infrastructures. Time-dependent systems (i.e., RTEs) which interact closely with their environment and satisfy its real-time requirements are built. The effects of violating such time requirements may range from slight system misbehaviour to loss of human life. As a result, the most important development task is to ensure that an RTE implementation is as fault-free as possible before its use.

Using formal methods, verification and model-based testing can ensure that the system is correctly implemented. While verification validates the specification

against functional and timing requirements, testing targets the correctness of the implementation. In general, testing strategies can be achieved by submitting a set of test cases to the SUT and observing its outputs. SUT behaviour can be declared correct or faulty after comparing observed outputs with a formal specification according to pre-defined conformance relations. Suitable formal languages are used to build the system specification from which test cases are then extracted.

Compared with un-timed systems, testing RTEs is far more difficult since it requires checking of timing correctness as a new dimension. Many testing approaches have been developed. However, most of them either suffer from high cost due to a large number of test cases or generate few tests without achieving high fault coverage. Moreover, research in real-time adequacy criteria is still immature and all existing coverage criteria are un-timed (i.e., do not take timing properties into account). As a result, there is a need for a new approach that it is both efficient in handling real-time specification, practical in use and derives a small number of test cases that achieve timed adequacy criteria and high fault coverage.

Chapter 3: A Priority-Based Approach for Testing Real-Time Embedded Systems

3.1 Overview

The problem of testing RTEs from a TA is tackled in this chapter by proposing the Priority-based Approach (PA) as a new component-based offline test case generation method for an RTEs modelled as a UTA (Aboutrab and Counsell, 2010). Test cases are selected according to Clock Region Coverage (CRC) as a proposed timed adequacy criterion supported by mathematical representations (Aboutrab et al., 2012b). CRC considers covering timing as well as functional behaviour of the RTEs under test by executing each transition within the UTA at specified time points. Considering clock guards, PA divides the generated test cases into three sets (namely boundary, out-boundary and in-boundary). The existence of three different test sets adds greater flexibility to the proposed PA in choosing suitable sets for a particular SUT.

To validate the performance of PA in comparison with four other similar TA-based testing approaches, the chapter proposes the application of Specification Mutation Analysis (SMA) in a TA context. A set of timed and functional mutation operators representing a set of incorrect behaviour is introduced. Three TA specification models are then used as case studies from which mutants are generated according to proposed mutation operators. The validation and comparison process is based on the mutation score calculated for each chosen timed testing approach with respect to a particular mutation operator (Aboutrab et al., 2012c).

The remainder of this chapter is organised as follows. The problem area this chapter tackles is highlighted in Section 3.2. Section 3.3 introduces preliminaries that explain TA and TA-based testing. The proposal of a timed adequacy criterion is presented in Section 3.4. The proposed PA is then explained in detail in Section 3.5 including its testing algorithms. Validating and comparing the performance of PA and other four TA-based testing approaches according to SMA is also presented in Section 3.6. Section 3.7 concludes the chapter.

3.2 Problem Area

Real-Time Embedded Systems (RTESs) have an increasing importance in modern society due to the close interaction with their environment. Testing an RTES implementation to ensure that it is fault-free before its deployment is necessary (En-Nouaary et al., 1998; En-Nouaary and Hamou-Lhadj, 2008; Hessel et al., 2008; Rollet, 2003). Model-Based Testing (MBT) is one of the testing approaches developed with the aim of achieving high fault detection capabilities and minimising cost through early capture of system behaviour and the automation of test case generation, execution and evaluation. A system's validity can be shown by comparing actual system behaviour with the formal model representing the system specification according to a conformance testing theory (Mitsching et al., 2009; Hessel et al., 2008; Tretmans, 1996). MBT can test timing behaviour of an SUT in addition to testing its functional behaviour if specification formalisms capable of capturing the required timing properties exist. A TA formalism is one of the most widespread due to its ability to express real-time behaviour of an SUT. It provides an easy and powerful means of extending finite-state machines with clock variables which track timing progress and incorporate timing constraints through the state-transition graph.

Testing from TA is problematic due to the need for discrete as well as continuous behaviour to be tested. Continuous behaviour of an SUT such as time has an infinite nature. As a result, generating test cases that entirely cover such behaviour is not possible. To tackle this problem, several TA-based testing algorithms have been proposed and differ from each other in the TA variant formalism they adopt,

the effort expended in their use, the number of test cases they produce and their effectiveness in detecting logical as well as timing faults (Clarke and Lee, 1997b; En-Nouaary and Hamou-Lhadj, 2008; En-Nouaary, 2008; En-Nouaary and Dssouli, 2003). Regardless of the TA variant used, its testing algorithms can be categorised based on how they handle infinite continuous behaviour as follows:

1. Time can be abstracted by using different equivalence relations that reduce the infinite state space of the specification model to be finite. Continuous behaviour is thus converted to discrete to enhance the application of un-timed test algorithms (Khoumsi et al., 2000; En-Nouaary et al., 1998). However, time abstraction may lead to the state explosion problem due to the large number of resulting states.
2. A discrete-time model is used to model clocks in TA to reduce the number of timed states (Krichen and Tripakis, 2009). However, the use of a discrete-time model contradicts the continuous behaviour of clocks.
3. Un-timed test selection criteria (e.g., transition coverage) or random search can be used for selecting test cases (Blom et al., 2005; Hessel et al., 2008). In other words, one or more random time points that satisfy clock guards are selected to trigger transitions. In spite of generating a relatively small test suite, timing behaviour is barely tested.

Adopting an appropriate test selection criterion can be considered as a key factor to handle testing RTEs. Literature has addressed two types of test selection criteria: structural and fault coverage. The aim of structural coverage (e.g., transition coverage) is to measure to what extent test cases cover the specification model. Coverage criteria proposed for un-timed systems were used for testing timed ones due to the lack of research studying formal timed coverage criteria for real-time systems. As a result, timing behaviour of an SUT will not be tested. It is thus essential to consider a timed coverage criterion for testing real-time systems.

On the other hand, fault coverage seeks tests capable of detecting potential faults in an SUT. Fault coverage needs to be facilitated by a fault model identifying the possible faults that might be encountered (Hessel et al., 2008; En-Nouaary et al., 1999). The power of any test suite can thus be determined by its fault coverage; the

higher the fault coverage, the more powerful the test suite (En-Nouaary and Hamou-Lhadj, 2008; En-Nouaary et al., 1999). The use of fault coverage as an assessment or selection criterion can be more effective if it is used in a controlled way by application of Specification Mutation Analysis (SMA). In the literature, to our knowledge no study has addressed the application of SMA on TA. Proposing well-suited TA-based mutation operators becomes a necessity for facilitating the application of SMA in a TA context.

The problem tackled by this chapter is to develop a timed testing approach that can handle real-time specifications based on a TA variant (UTA) and generate relatively small test suites with high structural and fault coverage. The primary contributions of this chapter are:

- 1- The proposal of Clock Region Coverage (CRC) as a timed adequacy criterion for covering timing behaviour of a TA specification.
- 2- The proposal of the Priority-based Approach (PA) including its algorithms for generating timed test cases from TA variant (UTA).
- 3- The proposal of timed mutation operators based on previously proposed timing fault models in the literature to facilitate the application of SMA in a TA context.
- 4- The validation of PA in comparison with four other similar TA testing approaches based on SMA application.

3.3 Preliminaries

This section introduces the mathematical definitions of the TA model and its variants, Timed Input Output Automata (TIOA) and UPPAAL TA (UTA). The model is then illustrated with an example to clarify its properties. We also highlight some definitions related to testing from a TA.

3.3.1 Timed Automata (TA)

TA (Alur and Dill, 1994) has been used by many researchers (Alur and Dill, 1994; En-Nouaary et al., 1999; Springintveld and Vaandrager, 1996; Springintveld et

al., 2001) for modelling real-time specifications. The popularity of the TA formalism comes from its ability to express most of RTESs behaviour. A TA provides an easy and powerful means of extending finite-state machines with real-valued clocks to model real-time processes over continuous time. More than one clock can be used to express time. However, the more clocks added, the more complex the model analysis.

Definition 3.1 Timed Automaton (TA): Let $\mathbb{R}_{\geq 0}$ be a set of non-negative reals. Let C be a set of $\mathbb{R}_{\geq 0}$ valued variables called clocks; $|C| = n$ (the number of clocks). Let $\mathcal{G}(C)$ denote the set of guards on clocks as conjunctions of constraints of the form $c \bowtie x$, where $c \in C, x \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$. Let $\mathcal{U}(C)$ denote the clock valuation function: $C \rightarrow [\mathbb{R}_{\geq 0} \cup \{\infty\}]^n$ as a dense time model. A timed Automaton TA is a tuple (L, l_0, C, A, E, I) , where:

- L : A set of locations that represent the system status after executing a transition.
- $l_0 \in L$: The initial location.
- C : A set of clocks. All clocks are initialized to zero at l_0 and may be reset after executing a transition.
- A : A set of actions.
- $I : L \rightarrow \mathcal{G}(C)$: An invariant which assigns guards to locations.
- $E \subseteq L \times A \times \mathcal{G}(C) \times 2^C \times L$: A set of transitions with an action, a guard, a set of clocks.

A transition in a TA is denoted by $l \xrightarrow{a,g,r} l'$ where:

- l : The source location.
- l' : The destination location.
- a : The action that fires the transition.
- g : The clock guard that should hold to execute the transition.
- r : The subset of clocks to be reset when the transition is fired.

Definition 3.2 Semantics of TA: Let $TA = (L, l_0, C, A, E, I)$ be a timed automaton. Its semantics are defined as a labelled transition system $\langle S, s_0, \rightarrow \rangle$, where:

- $S \subseteq L \times \mathbb{R}^C$: The set of states.
- $s_0 = (l_0, u_0)$: The initial state where $u_0(x) = 0$ for all $(x \in C)$.
- $\rightarrow \subseteq S \times (\mathbb{R}_{\geq 0} \cup A) \times S$: The transition relation such that:
 - $(l, u) \xrightarrow{d} (l, u + d)$ if $\forall d': 0 \leq d' \leq d \Rightarrow u + d' \in I(l)$. That allows the time delay by d .
 - $(l, u) \xrightarrow{a} (l', u')$ if there exists $e = (l, a, g, r, l') \in E$, $u' = [r \mapsto 0]u$ and $u' \in I(l')$, where:
 - $d \in \mathbb{R}_{\geq 0}$.
 - $u + d$ maps each clock x in C to the value $u(x) + d$.
 - $[r \mapsto 0]u$ denotes the clock valuation which maps clocks in r to 0 after firing a transition.

An action a is received or sent at a clock valuation $u \in \mathcal{U}(C)$. If u satisfies the clock guard g denoted by $u \models g$, a transition $l \xrightarrow{a, g, r} l'$ will be fired in which the automaton changes its location and subsequently its state.

Definition 3.3 Timed Input Output Automata (TIOA): A TIOA extends a TA by partitioning the set of actions into sets of inputs and outputs. A TIOA ‘A’ is a tuple $(I_A, O_A, L_A, l_A^0, C_A, T_A)$, where:

- I_A : A finite set of inputs received. Marked with ‘?’.
- O_A : A finite set of outputs sent. Marked with ‘!’.
- L_A : A set of locations that represent the system status after executing a transition.
- $l_A^0 \in L_A$: The initial location.
- C_A : A set of clocks. All clocks are initialized to zero at l_A^0 and may be reset after executing a transition.
- T_A : A set of transitions.

Definition 3.4 UPPAAL Timed Automata (UTA): UTA formalism is based on the theory of TA. It uses its pre-defined properties and offers additional features such as modelling the environment explicitly. The environmental model can then communicate with the system model by sending inputs (marked with‘?’) and receiving outputs (marked with ‘!’) through synchronized channels. Modelling the

environment allows the production of test scenarios compatible with a given environment and thus reduces the number of required tests. Moreover, a UTA facilitates the construction of large models by building parallel synchronized networks of UTAs. A UTA uses notations such as initial, committed and urgent locations. The initial location is represented by double circles and is the location from which the model starts. When reached, the committed location ‘C’ is used to indicate that its un-constrained transition should be triggered directly. Finally, the use of an urgent location ‘U’ indicates that the model cannot stay at this location for any length of time (Behrmann et al., 2004). A UTA consists of a network of timed automata over a common set of clocks and actions, consisting of n timed automata $TA_i = (L_i, l_i^0, C, A, E_i, l_i), 1 \leq i \leq n$.

Definition 3.5 Semantics of a Network of Timed Automata: Let $TA_i = (L_i, l_i^0, C, A, E_i, l_i)$ be the i^{th} branch of a network of n timed automata. Let $\bar{l}_0 = (l_1^0, \dots, l_n^0)$ be the initial location vector. Its semantics is defined as a transition system $\langle S, s_0, \rightarrow \rangle$, where:

- $S = (L_1 \times \dots \times L_n) \times \mathbb{R}^C$: The set of states.
- $s_0 = (\bar{l}_0, u_0)$: The initial state where $u_0(x) = 0$ for all $(x \in C)$.
- $\rightarrow \subseteq S \times S$: The transition relation defined by:
 - $(\bar{l}, u) \rightarrow (\bar{l}, u + d)$ if $\forall d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(\bar{l})$.
 - $(\bar{l}, u) \rightarrow (\bar{l}[l'_i/l_i], u')$ if there exists $l_i \xrightarrow{a, g, r} l'_i$. $u' = [r \mapsto 0]u$ and $u' \in I(\bar{l})$.
 - $(\bar{l}, u) \rightarrow (\bar{l}[l'_j/l_j, l'_i/l_i], u')$ if there exist $l_i \xrightarrow{a, g_i, r_i} l'_i$ and $l_j \xrightarrow{a, g_j, r_j} l'_j$. $u \in (g_i \wedge g_j), [r \mapsto 0]u$ and $u' \in I(\bar{l})$.

Definition 3.6 TA Test Suite: Let $TA = (L, l_0, C, A, E, I)$ be a timed automaton specification. Let $\mathbb{R}_{\geq 0}$ be a set of non-negative reals. Given $A = A? \cup A!$ as a finite set of input and output actions. $TR = \{tr_1, \dots, tr_n\}$ where $tr_i \in (\mathbb{R}_{\geq 0} \times A)^*$ represents a test suite comprising n test traces (i.e., test cases) represented by sequences of timed actions.

Figure 3.1.a shows a UTA model of a simple lamp controller. The user controls the brightness of the light by interacting with a touchpad within certain time intervals. The light brightness shows three levels: OFF, LOW and BRIGHT. The first press by the user turns the lamp on with low brightness. If the user presses the button again within ‘4’ time units, the light becomes brighter. Otherwise, the lamp turns off. The environment model representing the user in our example is shown in Figure 3.1.b. As an example of the semantics, the lamp may have the following sequence of transitions:

$$\begin{aligned}
 &(\text{OFF}, x = 0) \xrightarrow{2} (\text{OFF}, x = 2) \xrightarrow{\text{press?}} (\text{low}, x = 0) \xrightarrow{\text{low!}} (\text{LOW}, x = 0) \xrightarrow{4.23} (\text{LOW}, x \\
 &= 4.23) \xrightarrow{\text{press?}} (\text{off2}, x = 0) \xrightarrow{\text{off!}} (\text{OFF}, x = 0).
 \end{aligned}$$

We can form an observable trace in the UTA representing those semantics as a sequence of inputs, outputs and delays:

$$2. \text{press?} \rightarrow \text{low!} \rightarrow 4.23. \text{press?} \rightarrow \text{off!}$$

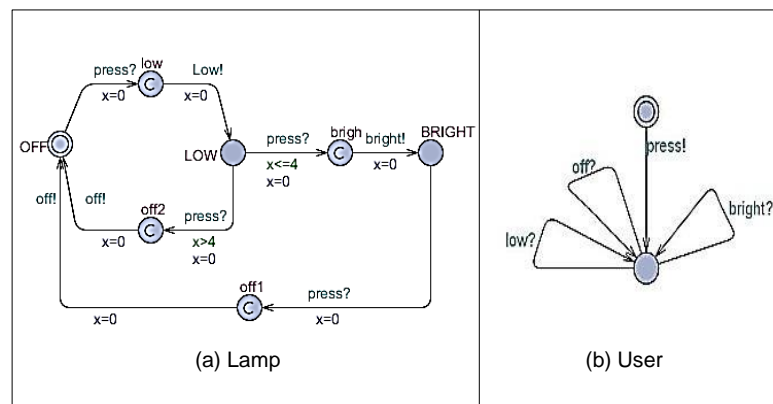


Figure 3.1: Simple lamp controller

3.3.2 Clock Region Abstraction

Since clock values are non-negative real numbers, the set of possible values of a clock is infinite. Covering entire clock values during the test case generation is impossible. As a result, the equivalence relation defined in (Alur and Dill, 1994) addressed this issue. The rationale behind defining such a relation was to divide

the TA clock valuation domain into equivalent regions. The clock values belonging to a certain region forces the TA to respond with same behaviour.

Definition 3.7 Equivalence Relation between Clock Valuations: Let $TA = (L, l_0, C, A, E, I)$. Two clock valuations $u, u' \in \mathcal{U}(C)$ are said to be equivalent ($u \sim u_0$) iff $\forall x, y \in C$:

- $\lfloor u(x) \rfloor = \lfloor u'(x) \rfloor$.
- $((u(x) \neq \infty) \wedge (u(y) \neq \infty)), (\text{fract}(u(x)) \leq \text{fract}(u(y)) \Leftrightarrow \text{fract}(u'(x)) \leq \text{fract}(u'(y)))$.
- $u(x) \neq \infty, (\text{fract}(u(x)) = 0 \Leftrightarrow \text{fract}(u'(x)) = 0)$.

Here, $\lfloor m \rfloor$ and $\text{fract}(m)$ denote the integral and fractional parts, respectively of the real number m . The \sim relation between two clock values is met if the integral parts and the ordering of the fractional parts of two clock values are equal. Integral parts are required to determine if a timing constraint has been met or not, while the ordering of fractional parts is required to know which clock changes its integral part first. The groups of equivalent clock values are called clock regions. The clock region of a clock valuation u is denoted by $\lfloor u \rfloor$. The set of all clock regions of a TA is denoted by the Region Automata $RA(TA)$.

Definition 3.8 Region Automata (RA): Let $TA = (L, l_0, C, A, E, I)$. The finite region automaton $RA(TA) = \langle S, s_0, \rightarrow \rangle$ where:

- S : The set of tuples $(l, \lfloor u \rfloor)$ in which each state comprises a location l and a clock region $\lfloor u \rfloor$ of a TA.
- $s_0 = (l_0, \lfloor u_0 \rfloor)$: is the initial state of the region automaton where $u_0(x) = 0$ for all $(x \in C)$.
- $\rightarrow \subseteq S \times S$: The transition from $s = (l, \lfloor u \rfloor)$ to $s' = (l', \lfloor u' \rfloor)$ where:
 - $s \xrightarrow{a} s'$ is an action transition $l \xrightarrow{a, g, r} l'$ with $u \models g$ and $u' = [r \mapsto 0]u$ exist.

- $s \xrightarrow{d} s'$ is a delay transition in a RA(TA). According to Definition 3.7, the least time delay d that moves the region automation from one state to another should fall between $]0, 1[$.

The importance of region automata comes from its compact nature in which we obtain a finite number of regions instead of an infinite number of clock valuations.

3.4 Timed Adequacy criterion: Clock Region Coverage (CRC)

Many structural coverage criteria have been proposed and studied for un-timed systems such as transition, state and definition-use coverage criteria. Due to the lack of research studying formal timed coverage criteria for real-time systems, the coverage criteria proposed for un-timed systems were mostly used for testing timed ones (Blom et al., 2005; Hessel et al., 2008). As a result, timing behaviour of an SUT will not be tested. Proposing a timed coverage criterion for testing real-time systems is essential. Timing behaviour of an SUT is represented by a set of timers or clocks whose values (i.e., non-negative real numbers) are infinite. As a result, generating test cases that cover each clock value is impossible. The equivalence relation in Definition 3.7 divides the clock valuation domain into a set of regions. Each region comprises equivalent clock valuations that cause the SUT to respond with the same behaviour. One clock valuation can thus safely represent the whole region to which it belongs. Figure 3.2 presents the clock valuation space where the x and y axes correspond to the values of clock x and y , respectively. For the sake of clarification, the regions are divided into three categories: corner point regions, open line segment regions and open area regions. For instance, $(1 < x < 2, 0 < y < 1 \text{ and } x > y)$ is an open area region in which the clock valuations $u_1 = (1.85, 0.5)$ and $u_2 = (1.63, 0.35)$, as an example, are equivalent according to Definition 3.7. This means that if the state (l, u_1) accepts a trace, then the state (l, u_2) also accepts that trace.

This concept was initially proposed by Alur and Dill (Alur and Dill, 1994) to significantly reduce the infinite timed state space by replacing a TA specification

with a finite region automaton. The region automaton is constructed using the equivalence relation by considering the number of clocks used in the specification model and the maximum length of each clock. Figure 3.2 shows that the total number of regions is '82' considering two clocks with '3' as a maximum length of both. '82' regions should be then constructed for every transition to create the region automaton.

Several studies (Springintveld et al., 2001; En-Nouaary et al., 2002) have the RA being used as an initial step for generating timed test cases. A Grid Automaton (GA) was then produced by sampling the RA (i.e., choosing representative points from each region of RA). However, the number of generated test cases from the GA was still large for two reasons. First, the RA is a very fine-grained abstraction technique; the number of clock regions increases significantly when the number of clocks or their upper bounds increase. Second, choosing a fixed granularity (i.e., sampling rate) for producing the GA leads to the selection of several representative values (i.e., time delay) from each region.

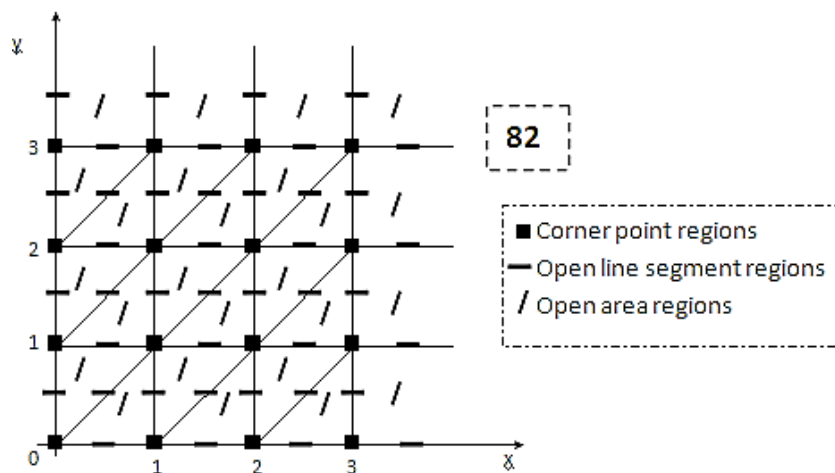


Figure 3.2: Clock regions

3.4.1 CRC Considerations

The idea of clock regions constitutes a timed adequacy criterion. A clock region is a fine-grained abstraction method that does not miss any clock value that might change system behaviour. CRC can thus be used to select and measure whether

the test suite covers all clock regions identified for each transition in the specification model. To facilitate using the concept of clock regions as a timed adequacy criterion, we propose a set of considerations to be taken into account to tackle its negative issues (i.e., the fast growth of clock regions by increasing the number of clocks and their upper bounds).

First, the clock regions need to be calculated at the transition level rather than at the model level, as proposed. The calculation of clock regions should thus consider (1) the number of clocks with their upper and lower bounds for a particular transition and (2) whether the transition is input (i.e., triggered by an input) or output (i.e., triggered by an output). In the case of an input transition, the test suite should consider all clock regions calculated for that transition, since inputs can be controlled by the tester. In other words, the tester can provide a set of inputs at certain times to cover the considered regions. However, clock regions calculated for output transitions need to be combined into one region since outputs are driven by an SUT and are not controlled by the tester. Any emitted output at a certain time triggering its transition is enough to confirm that the combined region defined for that transition is covered.

Second, proposing an appropriate process for determining the clock regions to be covered for each transition is essential for deploying the concept of CRC. Definition 3.7, as depicted in Figure 3.2, shows how to form regions. However, constructing the clock regions manually is a time consuming process especially if there is more than one clock controlling SUT timing behaviour. Calculating the number of clock regions to be covered can thus ease the process. Alur and Dill (Alur and Dill, 1994) proposed a mathematical Equation (3.1) for calculating the upper bound of clock regions.

$$NCR = |C|! \times 2^{|C|} \times \prod_{x \in C} (2c_x + 2) \quad (3.1)$$

Where:

- NCR : The number of clock regions.
- $|C|$: The number of clocks.

- c_x : The length of a timing constraint (i.e., the upper bound - the lower bound).

Applying Equation (3.1) to calculate the number of regions of Figure 3.2 gives:

$$|C| = 2 \text{ clocks; } c_x = c_y = 3.$$

$$NCR = (2 \times 1) \times 2^2 \times ((2 \times 3 + 2) \times (2 \times 3 + 2)) = 512$$

Counting the regions as per Figure 3.2, we find that the actual number of regions is 82. There is a large difference between the upper bound of the regions calculated according to Equation (3.1) and the actual number. Filling this gap, we propose a mathematical equation with the same notation as Equation (3.1) to calculate the number of regions accurately for up to three clocks used in the TA model (Section 3.4.2).

Third, we notice that not every clock region can be feasibly covered when more than one clock is used within the specification model. Identifying and calculating the number of feasible regions can also help to reduce the number of required regions and thus the generated test cases (Section 3.4.3).

3.4.2 Number of Clock Regions (NCR)

The proposed equations are proved according to the graphical representation of the clock regions for one clock, two clocks and three clocks leaving the generalised form for future work.

a. $|C| = 1$:

Figure 3.3 shows the least number of clock regions when there is one clock and $c_x = 1$. The number of clock regions is '4' ('2' corner regions + '2' open line segment regions). Minimum increase of c_x by '1' adds '1' corner region and '1' open line segment region. In other words, each increase in c_x adds '2' regions.

This can be represented mathematically by $NCR = 4 + (2 \times (c_x - 1))$ which leads to Equation (3.2).

$$NCR = 2 \times (c_x + 1) \quad (3.2)$$

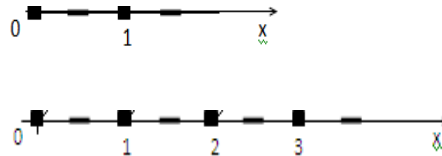


Figure 3.3: Regions with one clock

b. $|C| = 2$:

Similarly, when two clocks are controlling SUT behaviour, their values can be represented by a two-dimensional graph comprising sets of connected squares where each axis represents one of the clocks. Figure 3.4 shows the least number of clock regions when $c_x = c_y = 1$. The number of clock regions is ‘18’ (‘4’ corners + ‘9’ open line segments + ‘5’ open areas). The equation which calculates the number of clock regions can be derived in the following way. First, we consider the case where $c_x = 1$ and $c_y \geq 1$. The following clock regions are then obtained:

$$R_1 = (2(c_y+1) \text{ corners} + 4(c_y+1) \text{ open line segments} + c_y \text{ diagonal edges} + (3c_y+2) \text{ open areas}) = 10 c_y + 8.$$

Second, we consider the case for each increase in c_x by 1 (from some k to $k+1$). The following additional clock regions are obtained:

$$R_2 = ((c_y+1) \text{ corners} + 2(c_y+1) \text{ open line segments} + c_y \text{ diagonal edges} + (2c_y+1) \text{ open areas}) = 6 c_y + 4.$$

Thus, the general equation for $c_x \geq 1$ and $c_y \geq 1$ is formed by taking R_1 when $c_x = 1$ plus $(c_x - 1)$ times R_2 as c_x increases by 1. This gives:

$$NCR = R_1 + (c_x - 1) \times R_2 = (10 \times c_y + 8) + (c_x - 1) \times (6 c_y + 4)$$

By adjusting the equation, we obtain:

$$NCR = (6 \times c_x \times c_y) + 4 \times (c_x + c_y + 1) \tag{3.3}$$

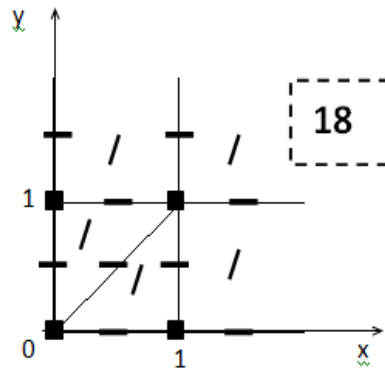


Figure 3.4: Regions with two clocks

Equation (3.3) represents the number of clock regions for two clocks with respect to c_x and c_y . By applying Equation (3.3) on Figure 3.2:

$$NCR = (6 \times (3 \times 3) + 4 \times (3 + 3 + 1)) = 82.$$

As seen, the result from our equation matches to the count from Figure 3.2.

c. $|\mathbf{C}| = 3$:

The more clocks, the more dimensions in the graphical representation. When the automaton uses three clocks, the clock valuation space consists of sets of connected cubes. Figure 3.5 shows the least number of clock regions when $c_x = c_y = c_z = 1$. The number of clock regions is ‘84’ (‘8’ corners + ‘37’ open line segments + ‘26’ open areas + ‘13’ open volumes). The equation which calculates the number of regions can be derived in the following way.

First, we consider the case where $c_x = c_z = 1$ and $c_y \geq 1$. The following clock regions are obtained:

$$R_1 = (4(c_y+1) \text{ corners} + 12(c_y+1) \text{ open line segments} + (6c_y + 1) \text{ diagonal edges} + (19c_y+13) \text{ open areas} + (9c_y+4) \text{ open volumes}) = 50 c_y + 34.$$

Second, we consider the case for each increase in c_x or c_z by 1 (from some k to $k+1$). The following additional clock regions are obtained:

$$R_2 = (2(c_y+1) \text{ corners} + 6(c_y+1) \text{ open line segments} + (5c_y + 1) \text{ diagonal edges} + (12c_y+7) \text{ open areas} + (7c_y+2) \text{ open volumes}) = 32 c_y + 18.$$

Third, we consider the case for each increase in c_x and c_z by 1. The following additional regions are obtained:

$$R_3 = ((c_y+1) \text{ corners} + 3(c_y+1) \text{ open line segments} + (4c_y + 1) \text{ diagonal edges} + (8c_y+4) \text{ open areas} + (6c_y+1) \text{ open volumes}) = 22 c_y + 10.$$

Thus, the general equation which calculates the number of clock regions for $c_x \geq 1$, $c_y \geq 1$ and $c_z \geq 1$ is formed by taking R_1 when $c_x = c_z = 1$ plus $(c_x - 1)$ times R_2 as c_x increases by 1 plus $(c_z - 1)$ times R_2 as c_z increases by 1 plus $(c_x - 1)(c_z - 1)$ times R_3 as c_x and c_z increase by 1. This gives:

$$\begin{aligned} NCR &= R_1 + (c_x - 1) \times R_2 + (c_z - 1) \times R_2 + (c_x - 1) \times (c_z - 1) \times R_3 = \\ &= (50 \times c_y + 34) + (c_x - 1) \times (32 c_y + 8) + (c_z - 1) \times (32 c_y + 8) + (c_x - 1) \times (c_z - 1) \times (22 c_y + 10) \end{aligned}$$

By adjusting the equation, we obtain Equation (3.4):

$$NCR = (22 \times c_x \times c_y \times c_z) + 10 \times (c_x \times c_y + c_x \times c_z + c_y \times c_z) + 8 \times (c_x + c_y + c_z + 1) \quad (3.4)$$

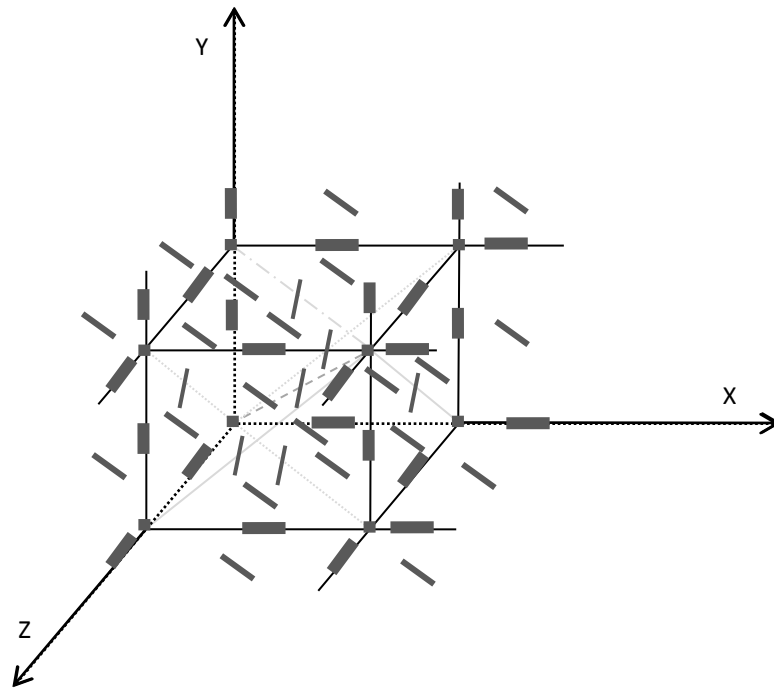


Figure 3.5: Regions with three clocks

3.4.3 Feasibility Issue of CRC

Covering all clock regions in a transition constrained by more than one clock is infeasible. The purpose of using several clocks in a specification model is to measure the elapsed time from different points in the model. The clocks need to be reset in different locations - no means exists for using several clocks resetting at same locations as they act as one clock. Time elapses in all clocks at the same speed. It is thus impossible for one clock at a certain transition, constrained by several clocks, to have values greater and, at the same time, less than the values of other clocks.

To clarify, consider Figure 3.6 that depicts a TA model controlled by two clocks.

The use of two clocks x , y over the transition: $(l_2 \xrightarrow{c?, x \leq 2 \& \& y \leq 3, (x, y)} l_3)$ is to ensure that the automaton reaches l_3 with no more than 2 time units from l_2 and no more than 3 time units from l_1 . Note that the clocks y and x reset together once the transition $(l_0 \xrightarrow{a?, 0 \leq x \leq 3, (x, y)} l_1)$ is fired where only the clock x

resets once the transition ($l_1 \xrightarrow{b?, x < 2, (x)} l_2$) is fired. The values of clock y in the transition ($l_2 \xrightarrow{ackAll!, x \leq 2 \& \& y \leq 3, (x, y)} l_3$) should be always greater than or equal to those of clock x since it starts earlier than clock x . As a result, half of the space representing the values of x, y (as in Figure 3.7) can be omitted without losing the clock regions in the middle line where the values of x are equal to the values of y .

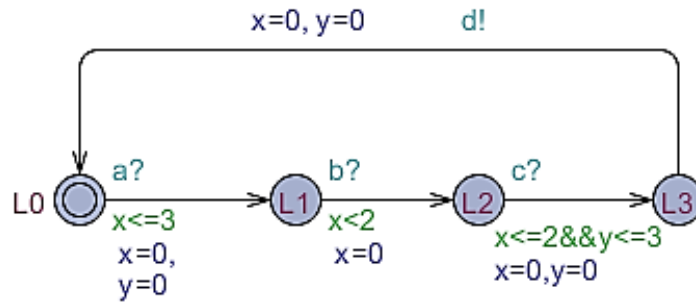


Figure 3.6: Two-clock automaton

The number of clock regions to be covered can be thus calculated according to Equation (3.5) in the case of two clocks.

$$NCR' = NCR - [(NCR_{min\ c_x} - NCR_{middle}) / 2] \quad (3.5)$$

Where:

- NCR : The total number of clock regions calculated for two clocks according to Equation (3.3).
- $NCR_{min\ c_x}$: The number of clock regions calculated according to Equation (3.3) by making the length of all clock guards over a transition equal to the minimum length among them.
- NCR_{middle} : The number of clock regions of the middle line which is calculated using Equation (3.2) by just considering the clock with the minimum length.

From Figure 3.7, $NCR = 82$ regions according to Equation (3.3). Since $c_x = c_y = 3$, $NCR_{min\ c_x} = NCR = 82$. $NCR_{middle} = 8$ according to Equation (3.2). Applying Equation (3.5), the number of effective regions to be covered:

$$NCR' = 82 - [(82 - 8)/2] = 45$$

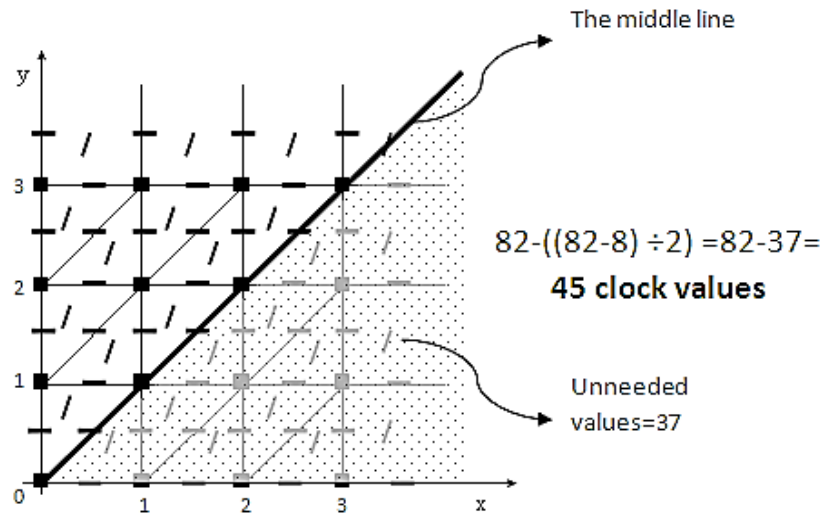


Figure 3.7: Feasible clock regions

The aim is thus to generate timed test cases that are able to cover all feasible clock regions for the whole specification model.

3.5 Priority-based Approach (PA)

This section presents the priority-based approach for generating timed test cases for an RTES modelled as UTA. A set of test hypotheses is first introduced to ensure that our approach is accurately used. We then explain the structure of our approach as well as presenting the accompanying algorithms that generate the timed test cases.

3.5.1 Test Hypotheses

The rationale for using test hypotheses is to specify the properties of the SUT and its tests since the implementation of and testing any system can be achieved in an infinite number of ways.

1. The testing process is applicable at the component level of an SUT.
2. The SUT and the specification are formally modelled by UTA in order to create the conformance relation between them.
3. The SUT is deterministic and fully observable. In other words, there are no transitions fired at the same time and no internal actions exist.
4. Minimal number of clocks should be used to express SUT timing behaviour to reduce the complexity of the model.
5. At the end of each test case, there is an implicit reset transition that brings the SUT to the initial state.
6. The SUT and the specification always accept inputs from test cases.
7. To highlight the test selection criterion that covers SUT timing behaviour of the SUT, no data variables are allowed in the specification model.

3.5.2 Test Selection

The test selection in PA is based on the proposed CRC. The CRC in TA-based testing relies on providing timed inputs capable of firing each transition (at least once) at different time points equal to the feasible NCR calculated for that transition. In other words, it is enough to choose an input with a time delay (i.e., clock value) to represent the region it belongs to. The selected timed inputs thus form test cases. In special cases, test cases are selected as follows:

- If the guard is always true (i.e., no guard over a transition exists to constrain timing behaviour), the time delays accompanied by a suitable input are incrementally chosen for each firing of the transition. In other words, starting from '0', clock values should be chosen from consecutive clock regions. This way of selection might help uncover more timing faults without increasing the number of test cases.

- If a transition starts from a committed or urgent location or represents an urgent channel, no time delays are permitted when applying the input to trigger that transition.

The core concept of the PA is based on dividing the generated test cases into three sets. Test sets are called ‘priorities’ as the priority of choosing a particular test set is different from one test set to another according to the testing environment specified by the criticality of an SUT, the allowable time and budget specified for the testing process (Aboutrab et al., 2010). Each test set (priority) is named and constructed according to the structure of clock guards.

a. Boundary Set (B)

B contains test cases that achieve transition coverage by considering the boundary values of clock guards defined for each transition they cover. The boundary values represent the clock values chosen from the boundary regions of a clock guard of the model $x \bowtie c$, where $x \in C$, $c \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$. In the case of $\bowtie \in \{\leq, =, \geq\}$, this set contains the exact boundary values of a clock guard. For instance, the boundary values of the clock guard $(1 \leq x \leq 4)$ are ‘1’ and ‘4’. Otherwise ($\bowtie \in \{<, >\}$), this set contains clock values from the direct neighbouring interior region by ε , where $0 < \varepsilon < 1$. For instance, the boundary values of the clock guard $(x < 2)$ are ‘0’ and ‘1.5’ by having $\varepsilon = \frac{1}{2}$.

b. Out-Boundary Set (OB)

OB contains test cases that achieve transition coverage by considering the out-boundary values of clock guards defined for each transition they cover. The out-boundary values represent the clock values chosen from the neighbouring region located out of clock guard boundaries by ε where $0 < \varepsilon < 1$. For instance, the out-boundary values of the clock guard $(1 \leq x \leq 4)$ are ‘0.5’ and ‘4.5’ by having $\varepsilon = \frac{1}{2}$.

c. In-Boundary Set (IB)

IB contains test cases that achieve transition coverage by considering the in-boundary values of clock guards defined for each transition they cover. The in-boundary values represent all clock values covering the remaining clock regions that have not been covered by the ‘boundary’ and ‘out-boundary’ sets. For instance, the in-boundary values of the clock guard ($1 \leq x \leq 4$) are ‘1.5’, ‘2’, ‘2.5’, ‘3’ and ‘3.5’.

Note that test cases achieving full CRC are: $\{B \cup OB \cup IB\}$.

3.5.3 Test Generation Algorithms

This section introduces the algorithms responsible for generating timed test cases for an SUT modelled as UTA according to CRC. Algorithm 1 (Figure 3.8) generates test cases responsible for achieving CRC in co-operation with algorithm 2 by which the priority sets are chosen. Algorithm 1 starts by placing the initial location l_0 in the set ‘*Wait*’ acting as a stack to store all destination locations reached by the transitions covered. The set ‘*Wait*’ directs the algorithm in choosing the following transitions to be covered and guarantees that no transition has been missed. Starting from l_0 , the algorithm creates a test trace comprising all transitions commencing from l_0 and ending at l_0 if it is possible. A depth-first search algorithm is used to cover as many transitions as possible in each test trace. If any branches are encountered, all their destination locations are added to the set ‘*Wait*’. One branch is then chosen by the search algorithm. Once this branch has been covered, the other branches are then consecutively retrieved from the set ‘*Wait*’. Each transition covered is represented by a pair (starting location, destination location) stored in the set (*Pass*). The main role of *Pass* is to ensure that the self-loop transition is not covered more than once. The pair (g, a) represented each transition covered within a test trace is then added to an array $W[]$. Each row of this array thus comprises a complete test trace whose components are a set of duals (g, a) representing the transitions covered by this trace. The algorithm then picks the following location in ‘*Wait*’ to form another test trace until all transitions are

covered. Once the whole automaton is covered, the complete test traces stored in $W[]$ are ready. The guards in $W[]$ are then traversed by the set of clock values according to the chosen priority (Algorithm 2) to produce $TC[]$; each row of $TC[]$ is thus a test case candidate.

ALGORITHM 1. TEST CASE GENERATION

```

1  Generate(Input:  $UTA$ , Output:  $TC[]$ ;  $N$ : traces count)
2  |    $Pass = \emptyset$ ;  $Wait = \{l_0\}$ ;  $W[] = 0$ ;  $TC[] = 0$ ;
   |    $n = 0$ ;
3  |   While (one transition at least not yet processed)
   |   do:
4  |        $m = 0$ ;
5  |       While( $Wait \neq \emptyset$ )
6  |           |   pick  $l$  from  $Wait$ ;
7  |           |   select ( $Transition\ l \xrightarrow{a,g,r} l'$  in  $UTA$ ) not
   |           |   yet processed
8  |           |   if  $((l' = l) \&\& (l, l') \text{ in } Pass) \text{ or } (l' = l_0)$ 
9  |           |       |   continue
10 |           |       |   else
11 |           |           |   add  $(g, a)$  to  $W[n, m]$ 
12 |           |           |    $m = m + 1$ ;
13 |           |           |   add  $(l, l')$  to  $Pass$ ;
14 |           |           |   add  $l'$  to  $Wait$ ;
15 |           |   if  $W[n]$  is already in  $W$ 
16 |           |       |   delete  $W[n]$ ;
17 |           |    $n = n + 1$ ;
18 |           |   For each row in  $W$ 
19 |           |       |   Generate  $u$  values according to chosen
   |           |       |   priority:  $u \models g$ ;
20 |           |       |   add the  $(u, a)$  to  $TC$ ;
21 |           |       |   if  $u$  does not exist
22 |           |       |       |   apply the next priority;
23 |           |   return  $TC$ ,  $N = \text{rowSize } TC$ ;

```

Figure 3.8: Algorithm 1

Algorithm 2 (Figure 3.9) assigns clock values to the resulted test traces stored in $W[]$ to compose and store timed test traces in $TC[]$ as follows. Each row (i.e., test trace) of $W[]$ is repeated in $TC[]$ until covering all clock regions of its transitions according to a chosen priority. The repeat of a test trace should be based on the largest set of clock regions to be covered by a transition within the test trace. The transition regions of a chosen priority can be covered before the last repeat of the test trace. In such a case, a set of clock regions of the next priority are selected to be covered until the last repeat of the test trace. This would help in decreasing the number of test cases required for covering the entire clock regions. In the case of

two clocks, the clock values of a transition constrained by two clocks should rely on the clock values of previous transitions when no clock reset exists.

ALGORITHM 2. Three-Set Priorities

Generate test point (input: g , priority, output: U)

In the case of Priority 1: (Boundary Points)

• g is a one-clock guard : $g=(x \bowtie c_x)$

$$U_{B(x)}[] = \left\{ \begin{array}{ll} \{(0, c_x)\} & , \text{ if } \bowtie \in \{\leq\} \\ \{c_x\} & , \text{ if } \bowtie \in \{\geq, =\} \\ \{(c_x - \varepsilon)\} & , \text{ if } \bowtie \in \{<\} \\ \{(c_x + \varepsilon)\} & , \text{ if } \bowtie \in \{>\} \end{array} \right\} \text{ Where } 0 < \varepsilon < 1$$

• g is a two-clock guard: $g=(x \bowtie_1 c_{x1}) \&\&(y \bowtie_2 c_{x2})$:

$$U_{B(x,y)}[] = \{ (U_{B(x)}, U_{B(y)}) \mid U_{B(y)} \geq U_{B(x)} \}$$

In the case of Priority 2: (Out-Boundary Points)

• g is a one-clock guard: $g=(x \bowtie c_x)$

$$U_{OB(x)}[] = \left\{ \begin{array}{ll} \{c_x\} & , \text{ if } \bowtie \in \{<, >\} \\ \{(c_x - \varepsilon)\} & , \text{ if } \bowtie \in \{\geq, =\} \\ \{(c_x + \varepsilon)\} & , \text{ if } \bowtie \in \{\leq\} \end{array} \right\} \text{ Where } 0 < \varepsilon < 1$$

• g is a two-clock guard: $g=(x \bowtie_1 c_{x1}) \&\&(y \bowtie_2 c_{x2})$:

$$U_{OB(x,y)}[] = \{ (U_{OB(x)}, U_{OB(y)}) \mid U_{OB(y)} \geq U_{OB(x)} \}$$

In the case of Priority 3: (In-Boundary Points)

• g is a one-clock guard: $g=(x \bowtie c_x)$

$$U_{IB(x)}[] = \left\{ \begin{array}{ll} \left\{ \left(\frac{k}{2} \right) \right\} & , \text{ if } \bowtie \in \{\leq, <\} \\ \left\{ \left(c_x + \frac{k'}{2} \right) \right\} & , \text{ if } \bowtie \in \{\geq, >\} \end{array} \right.$$

Where: $k \in \mathcal{N}$: Natural varies from 1 to $2c_x - 1$,

$k' \in \mathcal{N}$: Natural varies from 1 to a chosen natural.

• g is a two-clock guard: $g=(x \bowtie_1 c_{x1}) \&\&(y \bowtie_2 c_{x2})$

$$U[] = \left\{ \begin{array}{ll} \left\{ \left(\frac{k_1}{2}, \frac{k_2}{2} \right), \left(\frac{k_3}{3}, \frac{k_4}{3} \right) \right\} & , \text{ if } \bowtie_1, \bowtie_2 \in \{<, \leq\} \\ \left\{ \left(c_{x1} + \frac{k'_1}{2}, c_{x2} + \frac{k'_2}{2} \right), \left(c_{x1} + \frac{k'_3}{3}, c_{x2} + \frac{k'_4}{3} \right) \right\} & , \text{ if } \bowtie_1, \bowtie_2 \in \{\geq, >\} \\ \left\{ \left(\frac{k_1}{2}, c_{x2} + \frac{k'_2}{2} \right), \left(\frac{k_3}{3}, c_{x2} + \frac{k'_4}{3} \right) \right\} & , \text{ if } \bowtie_1 \in \{<, \leq\}, \bowtie_2 \in \{>, \geq\} \\ \left\{ \left(c_{x1} + \frac{k'_1}{2}, \frac{k_2}{2} \right), \left(c_{x1} + \frac{k'_3}{3}, \frac{k_4}{3} \right) \right\} & , \text{ if } \bowtie_1 \in \{>, \geq\}, \bowtie_2 \in \{<, \leq\} \end{array} \right.$$

Where:

$k_1 \in \mathcal{N}$: Natural varies from 0 to $2c_{x1}$ if $\bowtie_1, \bowtie_2 \in \{\leq\}$, from 0 to $2c_{x1} - 1$ if $\bowtie_1, \bowtie_2 \in \{<\}$

$k_3 \in \mathcal{N}$: Natural varies from 1 to $3c_{x1} - 1$.

$k_2 \in \mathcal{N}$: Natural varies from 1 to $2c_{x2}$ if $\bowtie_1, \bowtie_2 \in \{\leq\}$, from 1 to $2c_{x2} - 1$ if $\bowtie_1, \bowtie_2 \in \{<\}$

$k_4 \in \mathcal{N}$: Natural varies from 1 to $3c_{x2} - 1$.

$k_2 > k_1, k_4 > k_3$

k'_1, k'_2 : Naturals vary from 0 to a chosen natural if $\bowtie_1, \bowtie_2 \in \{\geq\}$, from 1 to a chosen natural if $\bowtie_1, \bowtie_2 \in \{>\}$

k'_3, k'_4, t'_3, t'_4 : Naturals vary from 1 to a chosen natural.

$k'_2 > k'_1, k'_4 > k'_3$.

$\frac{k_3}{3} \neq n + \frac{k_4}{3}, \frac{k'_3}{3} \neq n + \frac{k'_4}{3}$: n natural

$k_3, k_4, k'_3, k'_4 \notin \{3 \times m\}$: m natural not equal 0

$c_{x1} + \frac{k_1}{2} \leq \frac{k_2}{2}, c_{x1} + \frac{k'_3}{3} \leq \frac{k_4}{3}$.

$(\frac{k_1}{2}, \frac{k_2}{2}) \neq (c_{x1}, c_{x2}), (c_{x1} + \frac{k_1}{2}, c_{x2} + \frac{k_2}{2}) \neq (c_{x1}, c_{x2})$.

$(\frac{k_1}{2}, c_{x2} + \frac{k_2}{2}) \neq (c_{x1}, c_{x2}), (c_{x1} + \frac{k_1}{2}, \frac{k_2}{2}) \neq (c_{x1}, c_{x2})$.

Figure 3.9: Algorithm 2

As an instance of how the algorithms work, consider Figure 3.1.a. Two test traces will be generated (OFF-low-LOW-off2-OFF, OFF-low-LOW-bright-BRIGHT-off1-OFF). After composing the test traces, Algorithm 1 extracts the actions and guards from the transitions which the test traces cover to be stored in $W[]$ in the following way. $W[0] = (\text{true.press?}).(0.\text{low!}).(x > 4.\text{press?}).(0.\text{off!})$. $W[1] = (\text{true.press?}).(0.\text{low!}).(x \leq 4.\text{press?}).(0.\text{bright!}).(0.\text{off!})$. The clock values that cover certain clock regions are then selected according to a chosen priority set. The algorithms will ask the tester to specify the upper bound of x when $\bowtie \in \{\geq, >\}$ (e.g., $x \geq 4$).

Choosing the clock upper bound as 6 and $\varepsilon = \frac{1}{2}$ for the model in Figure 3.1.a, PA generates 15 test cases as depicted in Figure 3.10. The tests were verified manually. The outputs in this example are urgent (i.e., the outputs are generated with no delays). Note that the clock values of unconstrained transitions are incrementally chosen as we mentioned before in Section 3.5.2.

<p>Boundary priority set: <i>press?.low!.4.5.press?.off!</i> <i>0.5.press?.low!.press?.bright!.press?.off!</i> <i>1.press?.low!.4.press?.bright!.0.5.press?.off!</i></p> <p>Out-boundary priority set: <i>press?.low!.4.press?.off!</i> <i>press?.low!.4.5.press?.bright!.press?.off!</i></p> <p>In-boundary priority set: <i>1.5.press?.low!.5.press?.off!</i> <i>2.press?.low!.5.5.press?.off!</i> <i>2.5.press?.low!.6.press?.off!</i> <i>3.press?.low!.0.5.press?.bright!.1.press?.off!</i> <i>3.5.press?.low!.1.press?.bright!.1.5.press?.off!</i> <i>4.press?.low!.1.5.press?.bright!.2.press?.off!</i> <i>4.5.press?.low!.2.press?.bright!.2.5.press?.off!</i> <i>5.press?.low!.2.5.press?.bright!.3.press?.off!</i> <i>5.5.press?.low!.3.press?.bright!.3.5.press?.off!</i> <i>6.press?.low!.3.5.press?.bright!.4.press?.off!</i></p>
--

Figure 3.10: Generated test cases

3.6 Empirical Validation

This section aims to validate the proposed PA by assessing its capability of detecting popular timing faults in comparison with four of its counterparts. To control the assessment process, Specification Mutation Analysis (SMA) is used. Similar to the original mutation analysis, SMA injects single faults into a formal specification model by syntactically changing the specification according to pre-defined SMA operators. The generated first-order specification mutants are accordingly executed against a set of generated test cases. Specification mutants are killed if their outputs are different from those of the original specification. SMA is useful in validating model-based testing techniques by anticipating their capabilities for finding faults within the SUT (Budd and Gopal, 1985; Jia and Harman, 2010).

SMA was mainly based on simulating functional faults according to a set of proposed mutation operators. It is also essential to ensure that the timed test suite is valid and effective in terms of finding all possible timing as well as functional faults. Proposing the use of SMA in the timed specification context forms the key

factor in achieving the validation objective. To facilitate the use of SMA in validating PA based on the TA formalism, a set of timed and functional mutation operators is proposed and their execution and adequacy score are highlighted. Three TA-based specification models are used as case studies from which mutants are generated according to proposed mutation operators. The validation and comparison processes are based on the mutation score and the number of generated test cases.

3.6.1 Mutation Operators for TA

In the literature, to our knowledge no study has addressed the application of SMA on TA. To propose well-suited mutation operators for TA, all known faults defined in previously proposed timing fault models should be included and represented. As a result, our proposed TA mutation operators include the previously formalised fault models in the literature such as that proposed for TIOA by En-Nouaary (En-Nouaary et al., 1999; En-Nouaary et al., 2002) and for a constraint graph by Clark and Lee (Clarke and Lee, 1997a). TA mutation operators include two main classes: timed and functional mutation operators. A complete list of timed operators can be found in Appendix A.

- *Restricting Timing Constraints (RTC)*: These timed operators focus on the timing constraints (i.e., clock guards) defined for each transition within a TA. RTCs narrow down the timing bounds by which they rejects inputs satisfying the clock guards of a transition in the specification. As a result, the number of mutated model states decreases compared with those of the specification. The functionality of these operators is dependent on the conjunction type of a timing constraint (i.e., boundary type: open or closed). Formally, let $TA = (L, l_0, C, A, E, I)$ be a timed automaton specification. RTC can be defined as a transformation function that takes a clock guard of the form $x \bowtie c$ for $x \in C, c \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$ and returns a mutated version of the guard.

$$\text{RTC (true)} = \{x \bowtie c\},$$

$$\text{RTC}(x \bowtie c) = \begin{cases} (x \bowtie^* c - \varepsilon_1) \wedge (x \bowtie^{**} \varepsilon_2), & \text{if } \bowtie \in \{<, \leq\} \\ (x \bowtie^* \varepsilon_3) \wedge (x \bowtie^{**} c + \varepsilon_4), & \text{if } \bowtie \in \{>, \geq\} \\ (x \bowtie c) & , \text{ if } \bowtie \in \{=\} \end{cases}$$

Where:

$$\varepsilon_1, \varepsilon_2 \in \mathbb{N}_{\leq c}, \varepsilon_3 \in \mathbb{N}_{> c}, \varepsilon_4 \in \mathbb{N}, \bowtie^* \in \{\leq, <\}, \bowtie^{**} \in \{>, \geq\}.$$

- *Widening Timing Constraints (WTC)*: These timed operators rely on increasing the timing bounds by which they accepts inputs which fail to satisfy the clock guards of a transition in the specification. Accordingly, the number of mutated model states increases compared with those of the specification. Formally, let $\text{TA} = (L, l_0, C, A, E, I)$ be a timed automaton specification. WTC can be defined as a transformation function that takes a guard of the form $x \bowtie c$ for $x \in C, c \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$ and returns a mutated version of the guard.

$$\text{WTC}(x \bowtie c) = \begin{cases} (x \bowtie^* c + \varepsilon_2), & \text{if } \bowtie \in \{<, \leq, =\} \\ (x \bowtie^{**} c - \varepsilon_1), & \text{if } \bowtie \in \{>, \geq, =\} \end{cases}$$

Where:

$$\varepsilon_1 \in \mathbb{N}_{\leq c}, \varepsilon_2 \in \mathbb{N}, \bowtie^* \in \{\leq, <\}, \bowtie^{**} \in \{>, \geq\}.$$

- *Shifting Timing constraints (STC)*: These timed operators shift the timing bounds either by increasing or decreasing their values. Formally, let $\text{TA} = (L, l_0, C, A, E, I)$ be a timed automaton specification. STC can be defined as a transformation function that takes a guard of the form $x \bowtie c$ for $x \in C, c \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$ and returns a mutated version of the guard.

$$\text{STC}(x \bowtie c) = \begin{cases} (x \bowtie^* c + \varepsilon_3) \wedge (x \bowtie^{**} \varepsilon_2), & \text{if } \bowtie \in \{<, \leq, =\} \\ (x \bowtie^* \varepsilon_2) \wedge (x \bowtie^{**} c - \varepsilon_1), & \text{if } \bowtie \in \{>, \geq, =\} \end{cases}$$

Where:

$$\varepsilon_1 \in \mathbb{N}_{\leq c}, \varepsilon_2 \in \mathbb{N}_{> 0}, \varepsilon_3 \in \mathbb{N}, \bowtie^* \in \{\leq, <\}, \bowtie^{**} \in \{>, \geq\}.$$

- *Resetting a Clock* (RC): This timed operator adds a clock reset to a transition to force that clock to reset once the transition is fired. This operator affects clock order and the number of states. Formally, let $TA = (L, l_0, C, A, E, I)$ be a timed automaton specification. RC can be defined as a transformation function that adds a reset function $r_x \notin r$ for a clock $x \in C$ to a transition linking two locations $l_1, l_2 \in L$ and fired by the application of an input action $a \in A$.

$$RC \left(l_1 \xrightarrow{a, g, r} l_2 \right) = \{ l_1 \xrightarrow{a, g, r \cup \{r_x\}} l_2 \mid x \in C, r_x \notin r \}$$

- *Not-Resetting a Clock* (NRC): This timed operator involves removing an existing clock reset from a transition. This operator affects clock order and the number of states. Formally, let $TA = (L, l_0, C, A, E, I)$ be a timed automaton specification. NRC can be defined as a transformation function that deletes a reset function $r_x \in r$ for a clock $x \in C$ from a transition that links two locations $l_1, l_2 \in L$.

$$NRC \left(l_1 \xrightarrow{a, g, r} l_2 \right) = \{ l_1 \xrightarrow{a, g, r \setminus \{r_x\}} l_2 \mid x \in C, r_x \in r \}$$

- *Exchanging Input Actions* (EIA): This functional operator exchanges a pre-defined input action over a transition with another existing input action. Formally, let $TA = (L, l_0, C, A, E, I)$ be a timed automaton specification. EIA can be defined as a transformation function that replaces an input action $a_1 \in A$ firing a transition that links two locations $l_1, l_2 \in L$ by another $a_2 \in A$.

$$EIA \left(l_1 \xrightarrow{a_1?} l_2 \right) = \{ l_1 \xrightarrow{a_2?} l_2 \mid a_2 \neq a_1 \}.$$

- *Exchanging Output Action* (EOA): This functional operator is similar to the EIA operator but it exchanges outputs instead of inputs. Formally, let

$TA = (L, l_0, C, A, E, I)$ be a timed automaton specification. EOA can be defined as a transformation function that replaces an output action $b_1 \in A$ firing a transition that links two locations $l_1, l_2 \in L$ by another $b_2 \in A$.

$$EOA \left(l_1 \xrightarrow{b_1!} l_2 \right) = \{ l_1 \xrightarrow{b_2!} l_2 \mid b_2 \neq b_1 \}.$$

- *Transferring Destination Locations* (TDL): This functional operator involves changing the destination location of a transition. The mutated transition will reach a location different from the location that the original transition reaches. Formally, let $TA = (L, l_0, C, A, E, I)$ be a timed automaton specification. TDL can be defined as a transformation function of the following form in which $l_1, l_2, l_3 \in L$ are defined locations and $a \in A$ is an input or output action:

$$TDL \left(l_1 \xrightarrow{a} l_2 \right) = \{ l_1 \xrightarrow{a} l_3 \mid l_3 \neq l_2 \}.$$

3.6.2 Mutation Execution

Let $TA = (L, l_0, C, A, E, I)$ be a timed automaton specification. Let $\mathbb{R}_{\geq 0}$ be the set of non-negative reals. Given $A = A? \cup A!$ as a finite set of input and output actions, let $TR = \{tr_1, \dots, tr_n\}$ where $tr_i \in (R_{\geq 0} \times A)^*$ be a test suite comprising n test traces represented by sequences of timed actions. We define $run(M, tr)$ to be the set of timed output sequences $OS \in (R_{\geq 0} \times A!)^*$ that can result from the application of a test trace tr on the specification model M . The process of mutation execution can be represented by $run(M', tr)$ where M' is the mutated specification of M . The computation of $run(M, tr)$ and $run(M', tr)$ are manually achieved. Comparing the output sequences resulting from executing the test suite on a particular mutant with those expected according to the original specification, we can state the following to calculate the adequacy score for the test suite according to Equation (3.6).

$$\text{Adequacy Score} = \frac{\text{no of mutants Killed by a test suite}}{\text{total no of generated mutants} - \text{no of equivalent mutants}} \quad (3.6)$$

- *Killed*: It is said that a mutant M' is killed by a test trace tr if $run(M, tr) \cap run(M', tr) = \phi$. In other words, there is no common allowed behaviour between the mutant M' and the original specification M .
- *Potentially Killed*: It is said that a mutant M' is potentially killed by a test trace tr if $run(M', tr) \not\subseteq run(M, tr)$. There is some behaviour of M' that is not allowed by M . Here, we may need many runs of the same test trace to actually observe a failure, since the outputs are not controllable by the tester.
- *Alive*: It is said that a mutant M' is alive if $\forall tr \in TR: run(M', tr) \subseteq run(M, tr)$. M' behaviour is a subset of M behaviour .

3.6.3 Mutation Analysis

In SMA for TA, Equation (3.6) is followed for calculating the adequacy score for the test suite. The score thus indicates the percentage of how many faults are detected by a test suite. Identifying acceptable scores is largely dependent on the application itself. Since we are conducting a comparison study, the most important information we are revealing is which approach scores better.

On the other hand, identifying the equivalent mutants remains the major problem we face. We identify three types of equivalent mutants in the TA-based case studies. First, an equivalent mutant might be produced by the application of the RTC operator on a clock guard defined for an output transition. The generated mutant will show equal behaviour to the specification since it must emit the outputs within the allowed time defined in the specification. For instance, the mutant generated by reducing the time interval ' $x < 5$ ' defined in the specification to ' $x < 3$ ' will force the mutant to emit the outputs within '3' time units which is still within '5' time units defined in the specification. Second, an equivalent

mutant might be produced by the application of the RC operator on an output transition followed by an unconstrained transition containing that clock reset. Third, an equivalent mutant might be produced when the application of the TDL operator leads to the same consecutive output transitions.

3.6.4 TA-based Testing Approaches

Many TA-based algorithms and methods for testing real-time systems have been proposed. They differ from each other in the specification variant models they adopt, the number of test cases they produce and their effectiveness in discovering logical as well as timing faults. However, to our knowledge no comparison study has been performed to validate their performance.

In this study, four well-known TA-based approaches were selected to be compared with our PA: Timed Testing approach based on a State Characterization Technique (SCT) (En-Nouaary et al., 2002; En-Nouaary et al., 1998), Scalable Method (SM) (En-Nouaary, 2008), Boundary Checking Technique (BCT) (En-Nouaary and Hamou-Lhadj, 2008) and timed testing approach based on UPPAAL Model Checker (COVER) (Hessel et al., 2008).

Selecting those methods was based on several criteria for a more fair comparison process. The specification formalisms followed are similar to ours. PA and COVER rely on a UTA as an input language where SCT, SM and BCT use TIOA. Although the formalisms appear different, they all are variants of TA and share its properties. Moreover, all testing approaches being compared depend on the deterministic completely observable class of TA. Similar to PA, SCT and SM use the concept of region automata to abstract the TA. COVER, on the other hand, is based on coverage criteria for selecting test cases similar to ours. The following present a concise summary of the testing approaches chosen for the comparison study.

- *Timed Testing approach based on a State Characterization Technique (SCT)*: This timed testing approach is based on TIOA. The proposed approach relies on reducing the TIOA state space according to a clock

valuation equivalence rule and creating the region graph. A testable automaton, called Grid Automaton (GA) was introduced by sampling the region graph with $(1/(n+2))$ maximum granularity, where 'n' represents the number of clocks. GA is then transformed to a Nondeterministic Finite State Machine (NTFSM) to enable the authors to adopt the Wp-method (Fujiwara et al., 1991) for generating test cases. They evaluated the testing approach according to the adopted timing fault model and argued that the generated test cases could discover the main (known) timing faults.

- *Scalable Method (SM)*: This method is a timed testing approach based on TIOA. The proposed method relies on sampling the TIOA specification according to a clock valuation equivalence rule to reduce the infinite state space. A GA was obtained with the coarsest possible granularity $(1/(n+1))$ as a result of the sampling process. It chooses each transition once or twice at two time points (earliest and latest possible occurrences). The GA is then traversed using a depth-first algorithm to derive test cases. The approach appeared to be scalable and produced a small number of test cases.
- *Boundary Checking Technique (BCT)*: Another timed testing method is proposed for testing real-time systems modelled as TIOA. This method allows testing every transition of the TIOA at three different times (soonest, latest and between two executions). To move the TIOA to the transition under test, a preamble and postamble should be used. The preamble is a set of timed inputs capable of moving the TIOA up to a particular transition under test as soon as possible. On the other hand, the postamble is a set of timed inputs capable of moving the TIOA back to the initial location as soon as possible. The approach generates very small number of test cases. However, the fault detection capability is questionable.
- *Timed Testing Approach based on UPPAAL Model Checker (COVER)*: Hessel and Pettersson proposed a timed testing method that extends the UPPAAL model checker with coverage criteria expressed by the Observer Automata formalism (Blom et al., 2005) such as edge, location, definition-

use pair, definition and affect-pair coverage. The testing approach is automated by an offline testing tool called CO \sqrt ER. CO \sqrt ER uses the UPPAAL model checking engine with a query language to generate test cases (Hessel et al., 2008).

3.6.5 Case Studies

Three different case studies that match the requirements of the selected testing approaches are chosen from the literature to enhance our validation study. They are all deterministic TA models from which TIOA and UTA based testing approaches can generate test cases (as TA properties are shared in TIOA and UTA). The case studies differ from each other in their size and the number of clocks used. Our selection of these case studies considers the manual generation of test cases and manual application and analysis of SMA. We believe that any shortcomings detected in a certain testing approach by relatively small case studies will persist for larger ones.

- *Lamp Controller*: Figure 3.11 shows a single-clock specification model (Hessel et al., 2008). It comprises nine locations including five committed ones, twelve transitions, one input, three outputs and one clock. The user controls the brightness of the light by interacting with a touchpad within certain time intervals. The light shows three levels: OFF, DIM and BRIGHT. The automaton enables the user to change between any two brightness levels by pressing the touch pad at a certain time. For instance, if the first press is within '2' time units the lamp will be turned on with dim brightness. Otherwise, the lamp will be turned on with high brightness.
- *Multimedia System*: Figure 3.12 shows the specification model of a simple multimedia system (En-Nouaary, 2008). It comprises four locations, four transitions, three inputs, one output and two clocks. The point of using this automaton is to show how the testing approaches used in this study deal with more than one clock. This automaton sends an acknowledgment signal if it successfully receives the image and sound signals, respectively

within their allowed timing constraints. The output signal should be produced no more than three time units after receiving the image signal and no more than two time units after receiving the sound signal. If the system satisfies the input/output timing constraints, it resets in order to wait for another image from the initial location.

- *Phone System:* Figure 3.13 shows the specification model of a simple phone system (Clarke and Lee, 1997a; En-Nouaary and Dssouli, 2003). It comprises eight locations, thirteen transitions, six inputs, two outputs and one clock. This automaton produces the dial tone and establishes connection if it receives all five digits at the correct times. The system will return to its initial state whenever the user ends the call. Moreover, each number should be dialed within ‘5’ time units or an error signal will be produced.

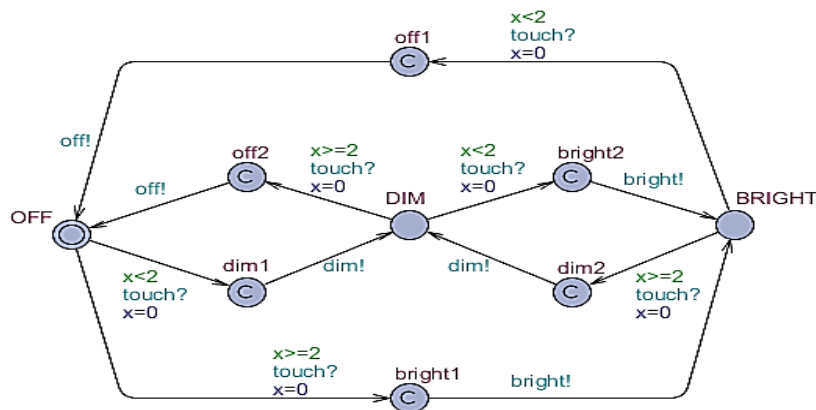


Figure 3.11: Lamp controller automaton

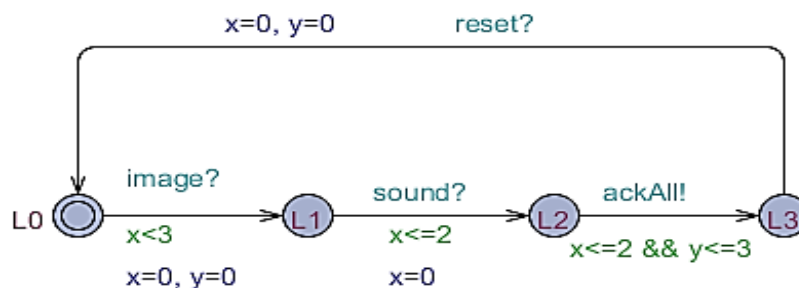


Figure 3.12: Multimedia automaton

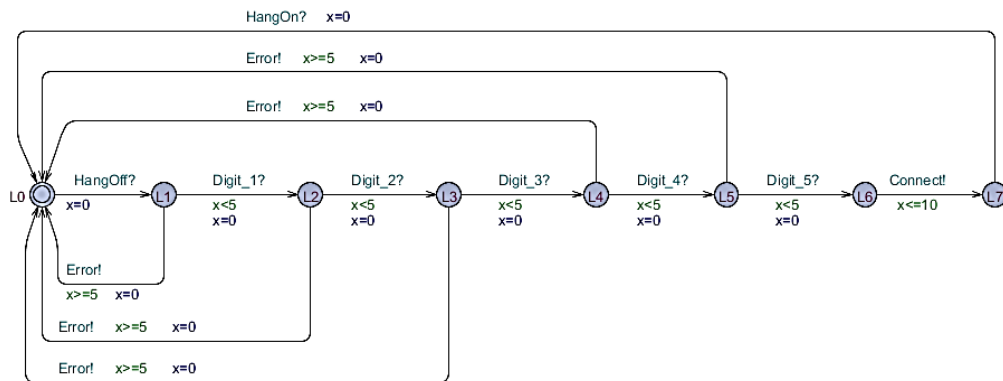


Figure 3.13: Phone automaton

3.6.6 Results and Discussion

Test cases were generated according to the chosen testing approaches for each case model. The test generation process was manually performed for PA, SCT, SM and BCT as they were not supported by tools. With respect to COVER, the COVER tool was used for generating test cases based on transition coverage as a test selection property (Hessel and Pettersson, 2007a). Transition coverage is considered more general than location coverage. Definition-use coverage criterion was not used as there was no data in the specification model.

Table 3.1 gives the number of test cases generated by each of the testing approaches used for each case study. We noticed that SCT suffered from a large number of tests and an enormous effort to manually generate those tests (2 days for each model). The number of tests rapidly increased when the size of specification models grew especially the number of locations, transitions and clocks. SM and BCT generated fewer tests due to the selection of fewer clock values to cover. Although SM shared SCT in their dependence on region automata, they did not cover all clock regions since they focused only on boundary clock regions. Moreover, the number of the generated tests from three case models was significantly affected by the size of specification models. PA and COVER showed a higher degree of stability with regard to the number of tests. Achieving as much coverage as possible in one single test case enabled PA to

generate relatively small number of tests. On the other hand, COVER generated few tests due to the un-timed coverage criterion being used. In other words, tests for checking SUT functional behaviour were only generated.

Approaches	Lamp Controller	Multimedia System	Phone System
PA	20	8	20
SCT	45	1295	3188
SM	37	4	63
BCT	10	7	29
COVER	4	1	1

Table 3.1: The count of generated test cases

The specification models were mutated according to the proposed operators. For each operator, all possible mutants were generated. Table 3.2, Table 3.3 and Table 3.4 depict the application of SMA for each of the testing methods on the ‘lamp controller’, ‘multimedia system’ and ‘phone system’, respectively. The number of mutants generated, equivalent, killed and potentially killed mutants, and mutation score were identified. The mutation score should consider the sum of ‘killed’ and ‘potentially killed’ mutant numbers. The following discusses the results of the application of SMA on each case study.

Applying the SMA, we noticed that some mutation operators (e.g., RC, EIA and EOA) were not applicable on some specification models due to the absence of the construct the mutation operator targets. To clarify, the application of RC involves adding a clock reset function to un-reset transitions. As a result, the application of RC on the ‘lamp controller’ was not possible since all clocks constraining the application of inputs had a reset function. The application of EIA was also not possible since there was just one input action defined in the model. With respect to the ‘multimedia system’, EOA was not applicable since there was just one output action.

Comparing the mutation scores the testing approaches achieved for each operator, we noticed the following.

COVER failed to achieve a high mutation score with respect to the timed operators for all case studies due to the usage of un-timed coverage criterion.

Approaches	Operators	Mutants	Equivalent	Killed	P-Killed	Score
PA	RTC	54	0	48	0	0.89
	WTC	29	0	0	29	1
	STC	31	0	19	12	1
	RC	N/A	-	-	-	-
	NRC	6	0	2	0	0.33
	EIA	N/A	-	-	-	-
	EOA	12	0	12	0	1
	TDL	42	6	24	0	0.67
SCT	RTC	54	0	48	0	0.89
	WTC	29	0	0	29	1
	STC	31	0	19	12	1
	RC	N/A	-	-	-	-
	NRC	6	0	3	0	0.5
	EIA	N/A	-	-	-	-
	EOA	12	0	12	0	1
	TDL	42	6	36	0	1
SM	RTC	54	0	47	0	0.87
	WTC	29	0	0	0	0
	STC	31	0	19	0	0.61
	RC	N/A	-	-	-	-
	NRC	6	0	4	0	0.67
	EIA	N/A	-	-	-	-
	EOA	12	0	12	0	1
	TDL	42	6	24	0	0.67
BCM	RTC	54	0	47	0	0.87
	WTC	29	0	0	0	0
	STC	31	0	19	0	0.61
	RC	N/A	-	-	-	-
	NRC	6	0	1	0	0.17
	EIA	N/A	-	-	-	-
	EOA	12	0	12	0	1
	TDL	42	6	24	0	0.67
COVER	RTC	54	0	16	0	0.3
	WTC	29	0	0	0	0
	STC	31	0	13	0	0.42
	RC	N/A	-	-	-	-
	NRC	6	0	1	0	0.17
	EIA	N/A	-	-	-	-
	EOA	12	0	6	0	0.5
	TDL	42	6	6	0	0.14

Table 3.2: SMA application on the lamp controller

PA achieved a full score with respect to RTC in the ‘multimedia system’. However, PA did not achieve a full score (but high score) in the ‘lamp controller’ and ‘phone system’ despite checking all boundary values of clock guards. Mutating unconstrained transitions by adding clock guards is the reason. Different from other testing approaches, PA is capable of checking the unconstrained transitions to some extent according to the total number of generated test cases. If

the added guard is far from the points PA checks, the fault is undetected. The other approaches (SCT, SM and BCT) provided with boundary checking facilities scored less than PA since they did not check unconstrained transitions.

Approaches	Operators	Mutants	Equivalent	Killed	P-Killed	Score
PA	RTC	33	18	15	0	1
	WTC	13	0	7	6	1
	STC	18	0	14	4	1
	RC	4	3	0	1	1
	NRC	5	1	3	1	1
	EIA	4	0	4	0	1
	EOA	N/A	-	-	-	-
	TDL	3	0	0	0	0
SCT	RTC	33	18	15	0	1
	WTC	13	0	7	6	1
	STC	18	0	14	4	1
	RC	4	3	0	1	1
	NRC	5	1	2	1	0.6
	EIA	4	0	4	0	1
	EOA	N/A	-	-	-	-
	TDL	3	0	0	0	0
SM	RTC	33	18	15	0	1
	WTC	13	0	0	0	0
	STC	18	0	14	2	0.89
	RC	4	3	0	0	0
	NRC	5	1	3	1	1
	EIA	4	0	4	0	1
	EOA	N/A	-	-	-	-
	TDL	3	0	0	0	0
BCM	RTC	33	18	15	0	1
	WTC	13	0	0	0	0
	STC	18	0	14	4	1
	RC	4	3	0	0	0
	NRC	5	1	3	1	1
	EIA	4	0	4	0	1
	EOA	N/A	-	-	-	-
	TDL	3	0	0	0	0
COVER	RTC	33	18	10	0	0.67
	WTC	13	0	0	0	0
	STC	18	0	14	4	1
	RC	4	3	0	0	0
	NRC	5	1	0	0	0
	EIA	4	0	4	0	1
	EOA	N/A	-	-	-	-
	TDL	3	0	0	0	0

Table 3.3: SMA application on the multimedia system

Approaches	Operators	Mutants	Equivalent	Killed	P-Killed	Score
PA	RTC	119	66	52	0	0.98
	WTC	38	0	20	18	1
	STC	54	0	30	24	1
	RC	1	1	0	0	-
	NRC	12	6	6	0	1
	EIA	36	0	36	0	1
	EOA	6	0	6	0	1
	TDL	42	0	0	0	0
SCT	RTC	119	66	44	0	0.83
	WTC	38	0	4	18	0.58
	STC	54	0	30	24	1
	RC	1	1	0	0	-
	NRC	12	6	0	0	0
	EIA	36	0	36	0	1
	EOA	6	0	6	0	1
	TDL	42	0	0	0	0
SM	RTC	119	66	51	0	0.96
	WTC	38	0	0	0	0
	STC	54	0	30	0	0.56
	RC	1	1	0	0	-
	NRC	12	6	5	0	0.83
	EIA	36	0	36	0	1
	EOA	6	0	6	0	1
	TDL	42	0	0	0	0
BCM	RTC	119	66	51	0	0.96
	WTC	38	0	0	0	0
	STC	54	0	30	0	0.56
	RC	1	1	0	0	-
	NRC	12	6	0	0	0
	EIA	36	0	36	0	1
	EOA	6	0	6	0	1
	TDL	42	0	0	0	0
COVER	RTC	119	66	40	0	0.75
	WTC	38	0	0	0	0
	STC	54	0	30	0	0.56
	RC	1	1	0	0	-
	NRC	12	6	0	0	0
	EIA	36	0	36	0	1
	EOA	6	0	1	0	0.17
	TDL	42	0	0	0	0

Table 3.4: SMA application on the phone system

In all case studies, PA achieved a full score with respect to WTC that involves expanding the clock guards in which unaccepted clock values become acceptable. PA considers checking the guards' out-boundary points. As a result, such faults can be detected. SCT showed the same ability to detect all WTC faults due to the large range of clock values that have been covered. However, the detection of such a fault was not consistent as SCT scores '0.58' in the 'phone system' case

study. SM and BCM, however, failed to detect any WTC faults. SM and BCM rely only on checking the boundary values of clock guards.

Moreover, PA capability of checking the boundary points as well as the out-boundary ones enables the detection of all possible shifting faults (i.e., STC). Again, SCT showed full detection capability of the STC faults. SM and BCM detected some but failed to detect others. The boundary checking that SM and BCT are based on increases the possibility of detecting such a fault, but does not guarantee full detection.

In the case of RC and NRC operators, PA showed more capability than the others in killing and potentially killing the generated mutants by scoring '1' in most cases. However, in the case of the 'lamp controller', SM scored the most '0.67' as it is the only method that considers all possible transition combinations when generating test cases.

With respect to the functional operators (EIA and EOA), all testing methods except COVER scored '1' as they covered all transitions while generating test cases. The COVER score was surprising as the generated test cases failed to cover any output transition. That might be due to failure in covering all transitions.

As expected with respect to the TDL operator, all testing methods failed to achieve a high score and sometimes achieved a '0' score. To kill such mutants, a testing approach should be equipped with a state identification capability; three testing approaches (PA, SM and BCT) do not have it. SCT, designed to detect state transfer faults, surprisingly failed to do so especially for the 'multimedia system' and 'phone system' models. In those case studies, we had a sequence of inputs with one or two outputs. Any mutant generated by altering a transition destination without leading to a different output or a different timing of an output failed to be killed. However, in the 'lamp controller' each input was followed by an output. As a result, all testing approaches had a score greater than 0 and SCT was able to score 1.

By applying the TA-based SMA, we increased our confidence about the performance of PA compared with the other approaches. Combining the mutation scores achieved by the testing approaches, PA achieved an almost full mutation score with respect to all timed and most functional mutation operators with relatively few tests (Figure 3.14). PA also showed a comparable result with respect to the TDL operator. When compared with SCT, equipped with state identification, PA cost and scored less but produced a smaller test suite. Further studies with larger models are still needed to confirm these outcomes.

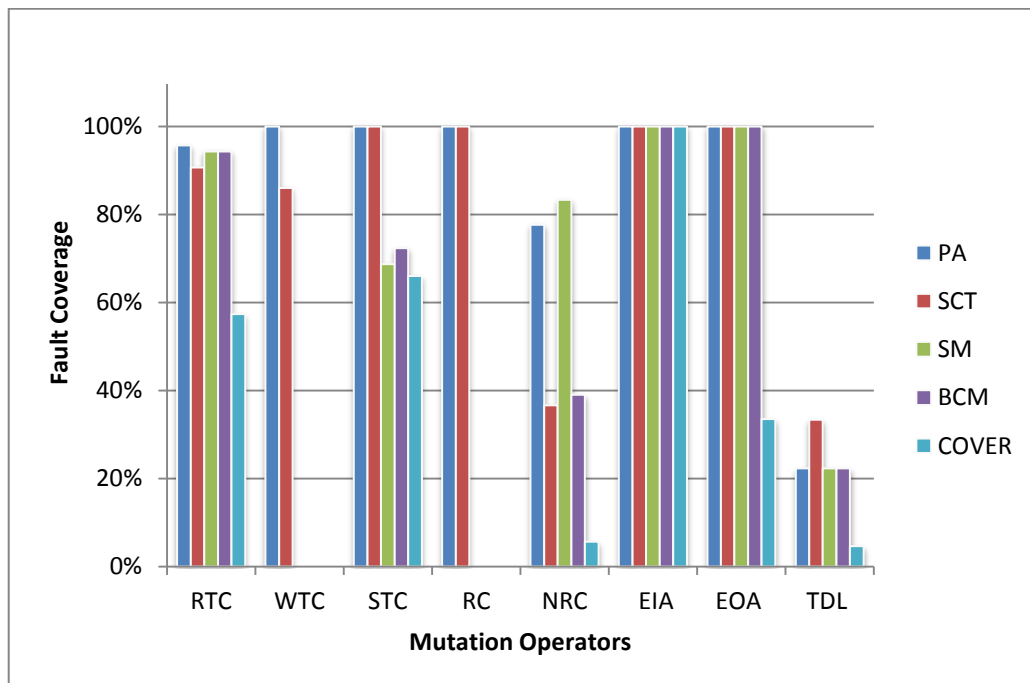


Figure 3.14: Fault detection ratio of the timed testing approaches with respect to mutation operators

In summary, Figure 3.15 depicts the overall fault detection capability of PA compared with other testing approaches. PA showed superiority in detecting timed as well as functional faults when compared with other approaches especially SCT that covered less with more cost.

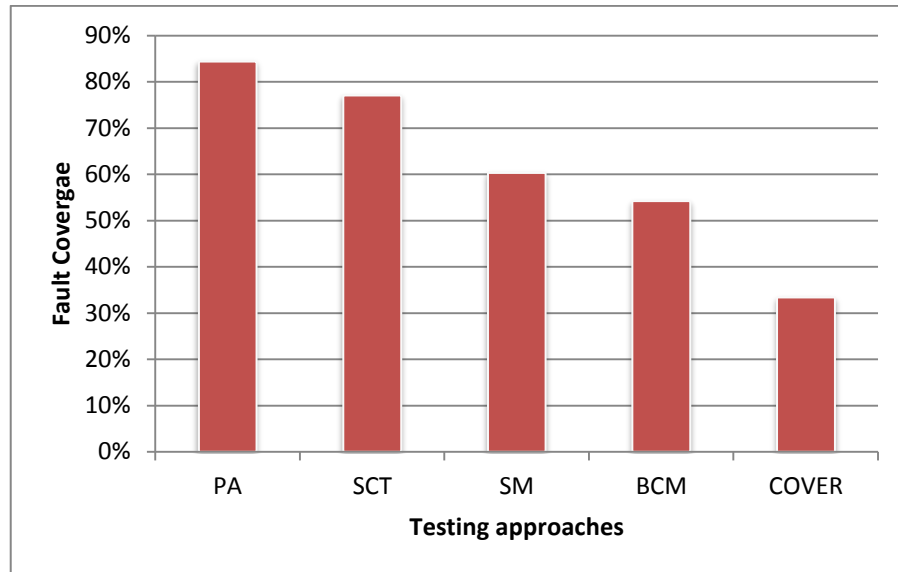


Figure 3.15: Overall fault coverage of the timed testing approaches

3.7 Summary

This chapter proposed the concept of clock region coverage CRC as a test selection criterion. CRC is based on the right selection of clock values that cover feasible clock regions (without losing or adding extra un-needed values that would increase the number of test cases). A set of mathematical equations that can help in efficiently calculating the number of regions was introduced. This chapter also presented an approach for generating timed test cases from a system specification modelled as UTA. This approach is based on CRC for generating test cases. The generated test cases are then divided into three sets of priorities (boundary, out-boundary, in-boundary). This enhances the flexibility of our approach by allowing the tester to choose the appropriate test set according to testing time and the criticality degree of the SUT. Complete algorithms that extract desired test cases according to our approach were then introduced.

In terms of validating the proposed PA, a TA-based Specification Mutation Analysis was introduced to compare our PA performance with some well-known testing approaches using three timed specification models. A set of timed and functional mutation operators was presented and discussed. We showed that our

PA out-performed other approaches if we combined the mutation score it obtained with the relatively few tests it generated. The validation revealed some interesting results such as the failure of SCT to detect all state transfer faults in spite of the state identification technique it is equipped with. Besides, COVER also failed in detecting all output or input faults in spite of the coverage criterion it follows.

Chapter 4: Automatic Test Case Generation and Execution using the Priority-Based Approach

4.1 Overview

In the previous chapter, we proposed a new component-based ‘priority-based’ approach (PA) for testing real-time systems modelled as UPPAAL Timed Automata (UTA). Test cases generated according to transition and clock region coverage criteria were divided into three sets of priorities, namely boundary, out-boundary and in-boundary, to reduce the number of required tests for a particular SUT. The selection of which test set is most appropriate for an SUT can be decided by the tester according to several factors such as the system type, testing time and testing budget.

This chapter extends the study by automating the generation and execution of test cases by developing a new timed testing tool, called GeTeX, and validating it using a TA-based prototype (specification model and code) (Aboutrab et al., 2011). GeTeX deploys the PA testing approach and *tioco* conformance theory and reduces the time and the cost required for the testing process. GeTeX can be considered as a complete offline testing tool that focuses on checking the correctness of SUT according to a timed selection criterion. In its current version, GeTeX supports Controller Area Network (CAN) applications.

The chapter also presents a set of code-based (timed and functional) mutation operators extracted from those proposed for TA-based SMA (see Section 3.6.1).

This enables the use of the Mutation Analysis Technique (MAT) for estimating fault coverage of a testing approach at the implementation level. Furthermore, the performance of our PA is assessed in comparison with some TA-based approaches, used in Chapter 3, but now at the implementation level, using a complete industrial-strength test bed (production-cell system). An assessment factor based on how many faults are detected and how many clock regions are covered in terms of the length of test cases generated by a testing approach is proposed. A set of lessons learned and the difficulties encountered, especially for testing the timing properties is highlighted (Aboutrab et al., 2012b).

The remainder of this chapter is organised as follows. The problem area this chapter tackles is highlighted in Section 4.2. Section 4.3 introduces preliminaries of *tioco* conformance theory and CAN principles. The proposal of GeTeX tool and its validation are presented in Section 4.4. Section 4.5 presents the assessment criteria, recalling the idea of clock regions as a timed testing coverage criterion, fault coverage supported by the use of mutation operators introduced for MAT, the mathematical representation of test case length and the assessment factor. Section 4.6 presents the production-cell test bed and the assessment results pointing to a set of lessons learned. Finally, Section 4.7 concludes the chapter.

4.2 Problem Area

Testing Real-Time Embedded Systems (RTESs) has become a popular research topic with significant recent attention given to model-based testing techniques. As a result, several TA-based testing algorithms have been proposed and differ from each other in the TA variant formalism they adopt, the effort expended in their use, the number of test cases they produce and their effectiveness in detecting logical as well as timing faults (Clarke and Lee, 1997b; En-Nouaary and Hamou-Lhadj, 2008; En-Nouaary, 2008; En-Nouaary and Dssouli, 2003). However, they suffer from the following problems which question their actual validity and complicate their actual use in real projects:

1. Most of the proposed approaches such as those in (Cardell-Oliver, 2000; En-Nouaary, 2008; En-Nouaary and Dssouli, 2003; En-Nouaary and Hamou-Lhadj, 2008; Springintveld et al., 2001) are theoretical frameworks for generating test cases. No automation support is provided. The application of such approaches requires a deep understanding of their mechanism and significant manual effort for generating and executing test cases.
2. Few proposed approaches are partially automated. Their tools are responsible for only automating the generation of test cases such as CO√ER (Hessel and Pettersson, 2007a), prototype RTCAT (Nielsen and Skou, 2001) and prototype tool TTG (Krichen and Tripakis, 2009). The execution of test cases generated by such approaches requires other sets of tools.
3. The software community still lacks serious and detailed industrial application of the proposed timed approaches. As an exception, CO√ER was applied using an industrial real-time test bed based on the WAP protocol (Hessel and Pettersson, 2007b). UPPAAL Tron has also been used in several industrial case studies such as the railway signalling case-study (Mitsching et al., 2009). However, CO√ER uses un-timed coverage criterion which does not guarantee coverage of timing behaviour of an SUT. UPPAAL Tron is an online testing tool where test case generation and execution take place at the same time. Timing behaviour of an SUT is not guaranteed to be covered as the choice of the next inputs to apply on an SUT is determined randomly, rather than following any selection criteria. The execution of a testing approach in a real-time context induces many problems (e.g., a time synchronisation issue) that need to be highlighted and tackled. More industrial test beds are thus necessary especially for validating the application of testing approaches concerning timing behaviour of an SUT.

4. To our knowledge, no detailed study that compares the performance of similar timed testing approaches on real applications based on well-identified assessment criteria exist. Such a study is essential to highlight the pros and cons of each approach to enrich the process of timed testing.

The *problem* tackled by this chapter is to address the above points by automating the generation and the execution of the proposed PA. The primary contributions of this chapter are:

- 1- The development of a new timed testing tool, called GeTeX. GeTeX automates the process of test case generation, execution and report based on PA and *tioco* conformance theory. In its current version, GeTeX is designed to support CAN applications as an example of RTEs.
- 2- The validation of GeTeX using a lamp controller prototype modelled as UTA and implemented as a CAN application.
- 3- The proposal of an assessment factor that combines fault coverage, structural coverage (i.e., clock region) and the length of test cases.
- 4- The application of Mutation Analysis Technique (MAT) at the implementation level as a means of measuring fault coverage of a testing approach. A set of mutation operators proposed in Chapter 3 have been mapped from the specification to the implementation level (C code) to enable the MAT application.
- 5- The application of PA on the implementation level using a complete industrial-strength test bed.
- 6- A comparison between the performance of PA and two similar testing approaches according to the proposed assessment criterion.

4.3 Preliminaries

This section introduces the mathematical definitions and properties of *tioco* conformance theory. A concise summary of CAN advantages is also presented.

4.3.1 Timed Input Output Conformance Theory (*tioco*)

tioco is a formal timed conformance relation inspired by un-timed *ioco* theory (Tretmans, 1996). Assuming that both the specification and SUT are modelled by the same formal language, both indicate that the SUT should behave according to the reference specification. SUT behaviour can be recorded by stimulating the SUT with a sequence of inputs and then observing its reactions. In the case of timed systems, SUT observed behaviour should not be limited to its observable outputs, but should also include their times since they are considered to be observable events. A *pass* or *fail* verdict will be given accordingly.

Definition 4.1 Conformance Relation *tioco*: Formally, *tioco* is defined as (Krichen and Tripakis, 2004):

$UTA_S \text{ } tioco \text{ } UTA_I$ iff

$$\forall \sigma \in ObsTTraces(UTA_S): out(UTA_I \text{ after } \sigma) \subseteq out(UTA_S \text{ after } \sigma)$$

Where:

- UTA_S and UTA_I represent the UTA specification and implementation models, respectively.
- *ObsTTraces* is a set that contains all possible sequences of observable timed actions.
- σ represents a sequence of observable timed actions.
- $out(UTAS \text{ after } \sigma)$ is a set of timed outputs after any behaviour σ .

tioco relation implies that for any observable behaviour of the specification, an implementation UTA_I conforms to the specification UTA_S if the set of SUT observable timed outputs is a subset of those of the specification at a certain matching point. If the implementation generally accepts inputs not included in the specification, a non-conformance or fail verdict will not arise since *tioco* is only related to the timed outputs. The main correctness properties that *tioco* pose are test suite *soundness* and *completeness*.

- A test suite TR is *sound* with respect to a UTA_S iff:

$$\forall UTA_I: UTA_I \text{ tioco } UTA_S \Rightarrow UTA_I \text{ passes } TR$$

- A test suite TR is *complete* with respect to a UTA_S iff:

$$\forall UTA_I: UTA_I \text{ passes } TR \Rightarrow UTA_I \text{ tioco } UTA_S$$

Soundness is a minimal correctness requirement. It is rather weak, since many tests can be sound (by always announcing *pass*). *Completeness* on the other hand, can be satisfied if, for every incorrect implementation, a test case can be generated that detects a non-conformance.

The rationale behind choosing *tioco* as a conformance relation to be adopted by our approach is its generality. *tioco* supports different types of specifications which range from non-deterministic partially observable with normal outputs to deterministic observable. Moreover, *tioco* allows the SUT to accept inputs undefined in the specification as long as they do not contradict with it. *tioco* also covers other timed relations such as Timed Trace Inclusion (TTI) and relativized *tioco* (*rtioco*). In other words, *tioco* can allow the comparison with other approaches that use different conformance relations.

4.3.2 Controller Area Network (CAN)

To initialize a strong serial communication, the CAN protocol was established by German Automotive systems in the mid-1980s. CAN is used in automobile industries because of its reliability, safety and efficiency. The popularity of CAN has widened to other markets of real-time embedded systems such as industrial automation, mobile devices and medical equipment (Tindell et al., 1995). As a result, it is chosen for the application of the PA approach in this Thesis. Since this chapter topic is not concentrated on CAN itself, the most important properties of the CAN protocol are only mentioned leaving the interested reader to follow (Pazul., 1999) for more details.

- Carrier-sense multiple access with collision detection.

- Message-based communication.
- Fast and robust communication including error detection capabilities.

4.4 GeTeX Tool Development

This section introduces the development process of a test Generating and Test eXecuting tool (GeTeX). The main components of GeTeX are presented. The outcomes of GeTeX are then validated using a lamp controller prototype.

4.4.1 GeTeX Design

GeTeX is developed to be a real-time test generation as well as a test execution tool. The requirements of GeTeX are based on PA algorithms for building its test generation engine and on a *tioco* conformance relation and the case study requirements for building its execution engine. This section gives an overview of GeTeX structure, as shown in Figure 4.1, to highlight its main features. GeTeX accepts a UTA specification model as an input. Using the UPPAAL model checker is thus necessary for creating UTA specification models and verifying them using temporal logic queries. The UTA models are compiled by UPPAAL into a file in an XML format recognizable by GeTeX. As UPPAAL supports the use of the network of timed automata, the produced XML file contains all the models of the UTA network.

The test generation engine of GeTeX applies PA algorithms to generate timed test cases from the XML file representing UTA specification models. Since PA is a component-based testing approach, the test generation engine allows the tester to choose a single UTA model to be the main source of generating timed test cases. For each single UTA model, GeTeX produces three sets of tests (boundary, out-boundary and in-boundary) according to PA. These sets thus add flexibility to the testing process by providing the tester with different choices. It is essential for any testing process to take a tester's opinion into account. Each testing process may

vary according to the testing environment, SUT type, testing time or testing budget, for an example. As a result, producing a single set of tests according to any chosen testing algorithms whatever the situation can be considered impractical especially for industrial applications.

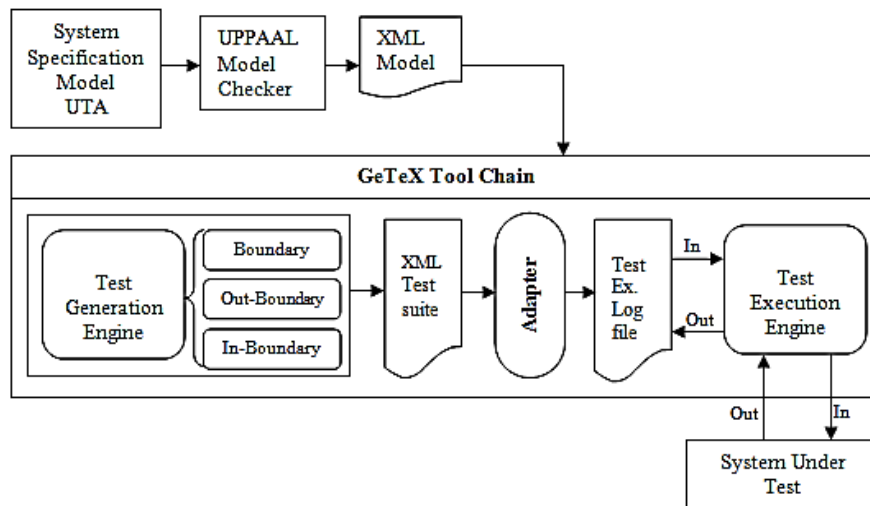


Figure 4.1: GeTeX chain structure

The test suite generated by GeTeX comprises a sequence of timed synchronised actions which need to be transformed to suit the SUT input domain. As a result, an XML data structure is chosen as a standard to represent the generated timed test suite to simplify the transformation process, whatever the SUT. In order to design the XML file of a test suite, the Document Object Model (DOM) defining a standard for accessing the XML file is built as shown in Figure 4.2. The test ‘priority’ sets form the basis of the test suite tree. Each ‘priority’ has an ‘id’, a ‘name’ and the ‘timed test traces’. Each timed test trace, recognizable by its ‘id’ comprises a sequence of timed actions (‘action’ at a certain ‘time’).

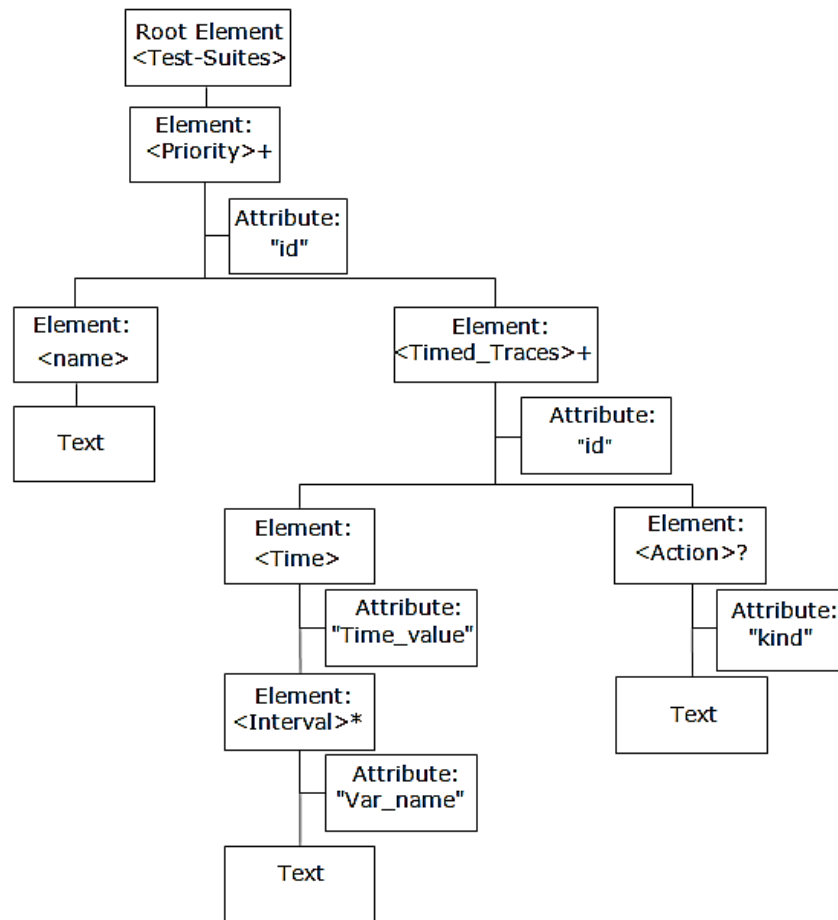


Figure 4.2: XML DOM tree of PA test suite

In UTA, actions can be inputs, outputs or internals according to the specification model. Action names are represented by a text and action types are represented by an attribute 'kind'. The time at which an action takes place is represented as an attribute to store a 'time_value' in the case of input actions and as an interval equation (e.g., $3 < x < 9$) in the case of output actions. In other words, the time at which inputs are sent to the SUT should be recorded whereas the time at which outputs are emitted from an SUT should be checked against specification timing intervals (i.e., timing constraints).

The adapter component of the GeTeX chain structure (Figure 4.1) is responsible for transforming the abstract XML test suite to real input data accepted by the

SUT. The adapter is a unique component that supports particular types of SUT. As a result, the adapter could be considered the most expensive part of the tool since different adapters need to be developed for different types of an SUT. In the current version of GeTeX, the adapter transforms the timed test suite to several sequences of CAN messages and stores them in the test suite log file. The log file is designed to enable the test execution engine to read inputs and write outputs easily into its predefined locations.

The test execution engine establishes the connection with an SUT using hardware adapters like Grid Connect USB/CAN adapter kit (Connect, 2010). It also monitors a CAN network from a personal computer using a USB port. The engine injects stored CAN messages into the CAN bus at specified time delays. The CAN bus is also being continuously monitored by the test execution engine to collect any messages transmitted from other CAN nodes. The received messages and their times are then stored into the log file. Time can be measured in different time units (e.g., seconds or micro-seconds) according to the chosen CAN bus baud rate. The test execution engine also establishes a *tioco* conformance relation by which timed output messages are compared with those expected; a *pass/fail* verdict is accordingly assigned to each timed test case. Finally, a test report is generated for the whole test suite.

4.4.2 GeTeX Implementation

GeTeX is a Java-based tool implemented using the NetBeans IDE 6.9.1 environment (NetBeans, 2010). It is a free environment which enables the user to easily debug, test and build a project. GeTeX is built under several packages presented in Figure 4.3. The test generation engine of GeTeX is implemented based on the PA test algorithms (Section 3.5.3).

The algorithms are implemented within two Java packages: ‘Test_Generation.Algorithm1’ and ‘Test_Generation.Algorithm2’. UTA

constructs and operations are created within the ‘Test_Generation.EntityClasses’ package; its classes and inheritance and association relationships are depicted in the class diagram. Three packages are dedicated for buffers. The ‘Buffer_Control.Specification_Level’ package is responsible for handling communication with the specification XML file. The ‘Buffer_Control’ package is responsible for converting generated timed test cases into XML format. The ‘Buffer_Control.Implementation_Level’ package is responsible for handling the communication with the SUT (i.e., adapter). The drivers of the USB/CAN adapter have been installed in the ‘peak.Can’ packages. The GeTeX execution engine and GUI are implemented in the ‘GeTeX’ package.

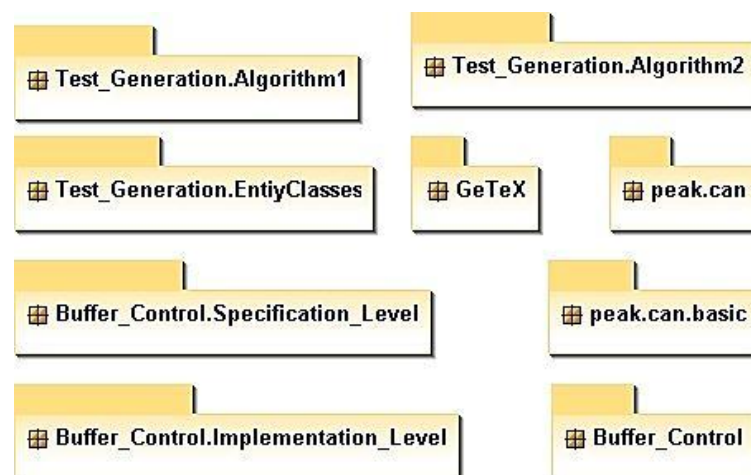


Figure 4.3: GeTeX packages

Figure 4.4 depicts the GUI of GeTeX based on the CAN adapter of Grid Connect. It comprises six panels. First, the ‘Test Generation/ CAN Configuration’ panel is responsible for configuring the generation of a timed test suite by choosing the specification model, test set, test suite XML file and test execution log file. Moreover, it configures CAN connection features such as ‘bus listen only’ mode. Second, the ‘New Connection’ panel is responsible for establishing a new

connection to the CAN bus by choosing the adapter channel type and the baud rate. Third, transmitting the CAN messages to the bus can be done via the ‘Write Messages’ panel. It gives the user two options - either to write and send a single CAN message from the GUI or to send a list of pre-defined CAN messages stored in a log file to the bus altogether.

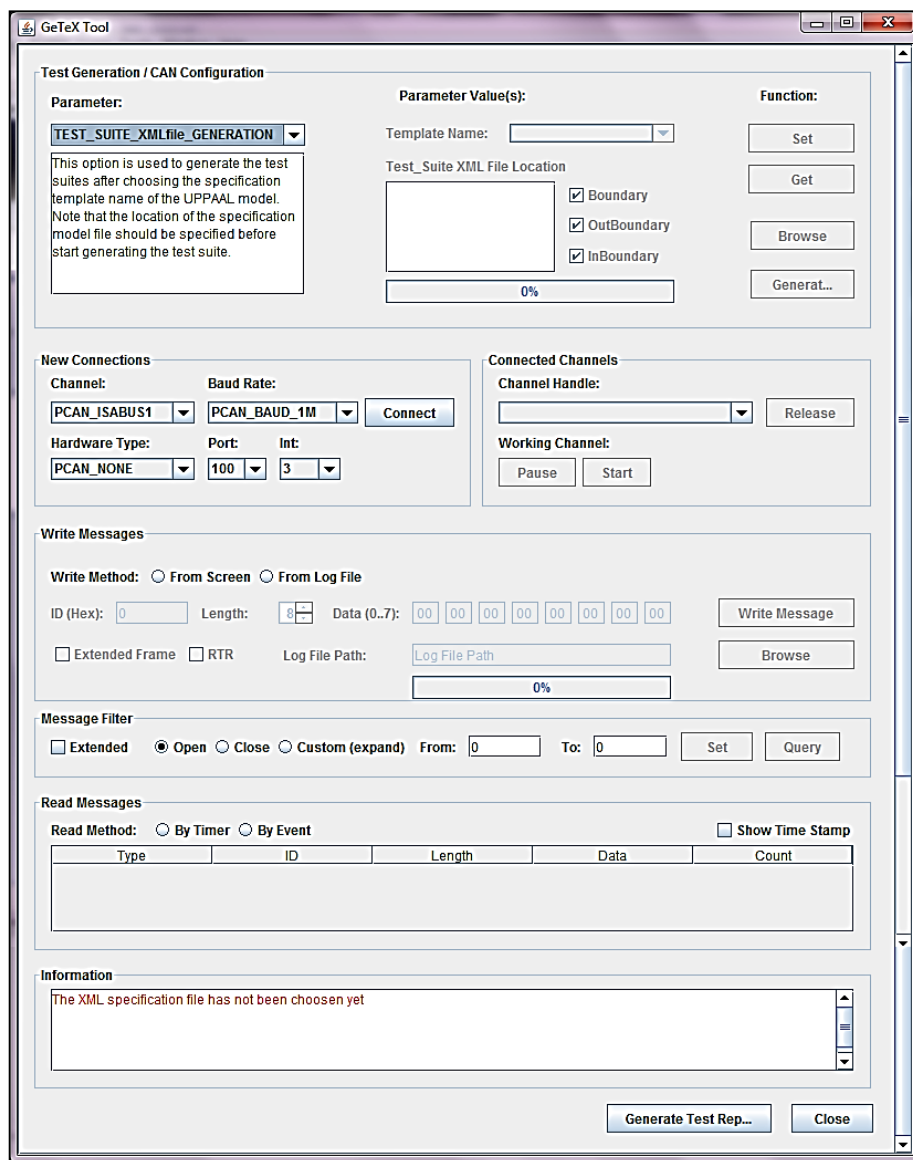


Figure 4.4: GeTeX GUI

Fourth, the ‘Message Filter’ panel is responsible for filtering received messages to view a group of them. Fifth, the ‘Read Messages’ panel views all received CAN messages from other nodes into the accompanied table. It gives the user two options - either to receive the messages on a certain time period or when they exist. CAN messages are identified according to their type, ID, length, data and receiving time. The CAN message type can be a standard frame format with 11 identifier bits or extended frame format with 29 identifier bits. The data carried in the message can range from 0 to 8 bytes in length and is represented by hexadecimal numbering system. The message count shows how many times a certain message has been received during a monitoring session. The time stamp of received message has been left as optional and can be added into the table by ticking the ‘show time stamp’ option. Finally, the ‘Information’ panel is for updating the user with the CAN bus status and the conditions of sending/receiving messages.

GeTeX was tested using JUnit test package. Test cases were designed to guarantee that all GeTeX methods run at least once. After executing test cases individually, an integrated test suite was performed to examine the tool performance.

4.4.3 GeTeX Trail

To demonstrate that GeTeX works correctly, we developed a lamp controller prototype based on the UTA model mentioned in Chapter 3 (Figure 3.1). An assumption that the controller is connected with the lamp via a CAN bus to form a two-node CAN network was made. The prototype was built using MCP2515DM-BM CAN Bus Monitor Demo Board (MicroshipDirect, 2010). The board kit contains two identical boards which can be connected together to create a simple two node CAN bus (i.e., one is implemented as the light and the other is implemented as the controller). Importing the XML file representing the UTA model of the lamp controller to GeTeX, the test generation engine produced the three-set timed test suite.

Figure 4.5 shows the three sets of generated test cases by setting the clock upper bound to 7. The empty brackets mean that the SUT was allowed to emit an action at any time. The total number of generated tests is manageable; 15 test cases in total. Note that the ‘out-boundary’ test set examines not allowed behaviour of the SUT. For *trace1* in ‘out-boundary’ priority as an instance, the correct SUT should not react with the output (*bright!*) after receiving the input action *press?* at ‘4.5’ time unit. In other words, the transition (LOW $\xrightarrow{\text{press?}, x \leq 4, x}$ bright) cannot be fired at a time point not satisfying its constraint ($x \leq 4$).

```

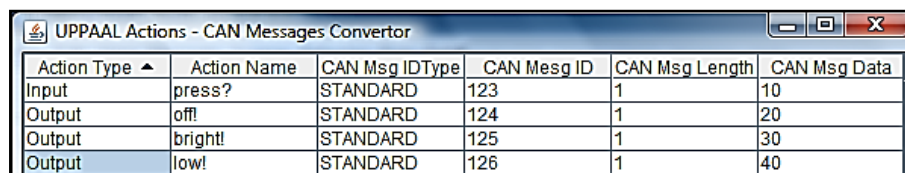
////////////////////////////////////Timed Traces////////////////////////////////////
-----
Boundary Priority
-----
(Trace1): (0).press? -> ().Low! -> (0).press? -> ().bright! -> (0).press? -> ().off!
(Trace2): (0.5).press? -> ().Low! -> (4).press? -> ().bright! -> (0.5).press? -> ().off!
(Trace3): (1).press? -> ().Low! -> (4.5).press? -> ().off!
-----
Out-Boundary Priority
-----
(Trace1): (1).press? -> ().Low! -> (4.5).press? -> ().bright! -> (0.5).press? -> ().off!
(Trace2): (1).press? -> ().Low! -> (4).press? -> ().off!
-----
In-Boundary Priority
-----
(Trace1): (1.5).press? -> ().Low! -> (0.5).press? -> ().bright! -> (1).press? -> ().off!
(Trace2): (2).press? -> ().Low! -> (1).press? -> ().bright! -> (1.5).press? -> ().off!
(Trace3): (2.5).press? -> ().Low! -> (1.5).press? -> ().bright! -> (2).press? -> ().off!
(Trace4): (3).press? -> ().Low! -> (2).press? -> ().bright! -> (2.5).press? -> ().off!
(Trace5): (3.5).press? -> ().Low! -> (2.5).press? -> ().bright! -> (3).press? -> ().off!
(Trace6): (4).press? -> ().Low! -> (3).press? -> ().bright! -> (3.5).press? -> ().off!
(Trace7): (4.5).press? -> ().Low! -> (3.5).press? -> ().bright! -> (4).press? -> ().off!
(Trace8): (5).press? -> ().Low! -> (5).press? -> ().off!
(Trace9): (5.5).press? -> ().Low! -> (5.5).press? -> ().off!
(Trace10): (6).press? -> ().Low! -> (6).press? -> ().off!
(Trace11): (6.5).press? -> ().Low! -> (6.5).press? -> ().off!
(Trace12): (7).press? -> ().Low! -> (7).press? -> ().off!

```

Figure 4.5: GeTeX test generation engine outcomes

Once timed test cases are generated, UTA abstract actions are transformed to a sequence of CAN messages according to the conversion table (see Figure 4.6). The conversion table is essential for GeTeX to configure CAN messages that need to be sent and received according to their counterpart actions. At the first step,

GeTeX sets all actions defined in the UTA specification model in the conversion table. The tester then needs to fill CAN message details according to the SUT design requirements. In our example, one input action (*press?*) and three output actions (*off!*, *low!*, *bright!*) were identified. CAN message details were assigned according to the lamp controller design where CAN messages ID and data were known. A ‘standard’ type (i.e., 11 bit) was chosen to represent the ID of CAN messages since the prototype consists of just two nodes. Their IDs and data were chosen to enable the controller and lamp nodes to understand each other’s messages. Once this table is ready, the abstract timed test suite was converted and stored in the log file allowing the test execution process to start.



Action Type ▲	Action Name	CAN Msg IDType	CAN Mesg ID	CAN Msg Length	CAN Msg Data
Input	press?	STANDARD	123	1	10
Output	off!	STANDARD	124	1	20
Output	bright!	STANDARD	125	1	30
Output	low!	STANDARD	126	1	40

Figure 4.6: Actions/CAN messages convertor

The test log file shown in Figure 4.7 is an Excel format consisting of 17 columns to store the sent/received CAN messages to/from the SUT. The ‘SetID’ column represents the corresponding ID of the testing sets: ‘1’ for boundary set, ‘2’ for out-boundary set and ‘3’ for in-boundary set. The ‘traceID’ column corresponds to the test trace identification within a test set. Since each test trace comprises a sequence of timed actions that have been converted to CAN messages, the ‘MsgType’, ‘MsgIDtype’, ‘MsgLength’ and ‘MsgData’ columns represent CAN message details. The ‘MsgX-time’ column stores time delays that determine at which time an action (i.e., a CAN message) should be sent to the bus or stores the timing interval at which an action can be received. When the UTA specification model uses more than one clock to represent its timing behaviour, each clock valuation is represented by separate columns named: ‘MsgX-time’, ‘MsgY-time’, ‘MsgZ-time’...etc.

The ‘status’ column identifies the communication status with the SUT. An ‘OK’ statement is used if a CAN message has successfully been sent or received. Otherwise, an ‘ERROR’ statement is used to identify that there was an error during the communication process with the CAN bus.

setID	tracelD	MsgType	MsgIDtype	MsgID	MsgLength	MsgData	MsgX-Time[s]	status	TimeVerdict	ActionVerdict	TestVerdict	MsgRcvIDtype	MsgRcvID	MsgRcvLe	MsgRcvData	RcvTime[s]	
1	1	1	Input	STANDARD	123	1	10	0	OK	Pass	Pass	Pass					
2	1	1	Output	STANDARD	126	1	40		OK	Pass	Pass	Pass	STANDARD	126	1	40	0.053
3	1	1	Input	STANDARD	123	1	10	0	OK	Pass	Pass	Pass					
4	1	1	Output	STANDARD	125	1	30		OK	Fail	Fail	Fail					
5	1	1	Input	STANDARD	123	1	10	0	OK	Pass	Pass	Pass					
6	1	1	Output	STANDARD	124	1	20		OK	Fail	Fail	Fail					
7	1	2	Input	STANDARD	123	1	10	0.5	OK	Pass	Pass	Pass					
8	1	2	Output	STANDARD	126	1	40		OK	Pass	Pass	Pass	STANDARD	126	1	40	0.062
9	1	2	Input	STANDARD	123	1	10	4	OK	Pass	Pass	Pass					
10	1	2	Output	STANDARD	125	1	30		OK	Pass	Pass	Pass	STANDARD	125	1	30	0.055
11	1	2	Input	STANDARD	123	1	10	0.5	OK	Pass	Pass	Pass					
12	1	2	Output	STANDARD	124	1	20		OK	Pass	Pass	Pass	STANDARD	124	1	20	0.045
13	1	3	Input	STANDARD	123	1	10	1	OK	Pass	Pass	Pass					
14	1	3	Output	STANDARD	126	1	40		OK	Pass	Pass	Pass	STANDARD	126	1	40	0.043
15	1	3	Input	STANDARD	123	1	10	4.5	OK	Pass	Pass	Pass					
16	1	3	Output	STANDARD	124	1	20		OK	Pass	Pass	Pass	STANDARD	124	1	20	0.04
17	2	1	Input	STANDARD	123	1	10	1.5	OK	Pass	Pass	Pass					
18	2	1	Output	STANDARD	126	1	40		OK	Pass	Pass	Pass	STANDARD	126	1	40	0.053
19	2	1	Input	STANDARD	123	1	10	4.5	OK	Pass	Pass	Pass					
20	2	2	Input	STANDARD	123	1	10	2	OK	Pass	Pass	Pass					
21	2	2	Output	STANDARD	126	1	40		OK	Pass	Pass	Pass	STANDARD	126	1	40	0.042
22	2	2	Input	STANDARD	123	1	10	4	OK	Pass	Pass	Pass					
23	3	1	Input	STANDARD	123	1	10	2	OK	Pass	Pass	Pass					
24	3	1	Output	STANDARD	126	1	40		OK	Pass	Pass	Pass	STANDARD	126	1	40	0.051
25	3	1	Input	STANDARD	123	1	10	0.5	OK	Pass	Pass	Pass					
26	3	1	Output	STANDARD	125	1	30		OK	Fail	Fail	Fail					
27	3	1	Input	STANDARD	123	1	10	0.5	OK	Pass	Pass	Pass					
28	3	1	Output	STANDARD	124	1	20		OK	Fail	Fail	Fail					
29	3	2	Input	STANDARD	123	1	10	2.5	OK	Pass	Pass	Pass					

Figure 4.7: A part of the test suite log file

Injecting the messages stored in the log file into the CAN bus, GeTeX monitors the bus in the case of any received messages which need to be stored in the ‘MsgRcvIDtype’, ‘MsgRcvLength’, ‘MsgRcvData’ and ‘RcvTime’ columns. The communication with the SUT may suffer from time delays due to (1) the time required for processing CAN messages by the CAN controller, (2) messages travelling time within the CAN bus and (3) the execution time of GeTeX code. Identifying this problem, GeTeX compensates for the time delay to a certain precision by measuring code execution time and calculating the propagation delay of the CAN controller and bus.

Having all input messages sent to and all output messages received, the test execution engine of GeTeX prepares the test report. According to *tioco* conformance relation, the test execution engine compares the received output messages and their times with those of the specification model. *Pass/fail* verdicts are accordingly assigned and stored in the following columns. The ‘TimeVerdict’ column assigns the *pass/fail* verdicts as a result of checking the time at which an output message is received with its timing guards. The ‘ActionVerdict’ column assigns the *pass/fail* verdicts as a result of checking the received output message with that expected according to the specification model. In the case of input messages, *pass/fail* verdicts are assigned according to the communication status with the SUT (i.e., whether the input message is successfully sent to the CAN bus). Finally, the ‘TestVerdict’ column determines the eventual verdict of a certain message by combining its verdicts stored in the ‘TimeVerdict’ and ‘ActionVerdict’ columns.

The first run of the experiment showed no faults. Every test set was correctly executed as the status column shows. Choosing a small application for the trial run enabled us to validate the tool. First, the test generation engine was validated by comparing the tests generated by the tool with those produced manually. Second, the test execution engine was validated by several runs of the experiment with different faults injected into the controller in different locations. For instance, the clock guard constraining the transition ($LOW \xrightarrow{press?, x \leq 4, x} bright$) was transformed to $2 \leq x \leq 4$ by which inputs satisfied the original guard should be rejected. Running *Trace 1* of the ‘boundary’ or ‘in-boundary’ test sets, the tool detected the injected fault by reporting this with a test trace *fail*.

GeTeX was capable of identifying the location of detected faults by referring to the action type, trace number and priority number. The grey box within Figure 4.7 shows examples. GeTeX was also capable of generating and executing timed test cases in short time. The validation process showed that the tool accurately represented PA.

4.5 Testing Assessment Criteria

This section introduces a set of assessment criteria; structural (clock region) coverage, timing fault coverage and test trace length. The assessment criteria by which the performance of TA-based testing approaches can be measured and compared is necessary.

Coverage criteria are often used in testing to assess the level of thoroughness of a test suite. Different types of coverage criteria are discussed and used in the literature such as structural and fault coverage. Fault coverage seeks tests capable of detecting potential faults in the SUT. Measuring fault coverage needs to be facilitated by:

- 1- A fault model identifying the possible faults that might be encountered.
- 2- The application of Mutation Analysis Technique (MAT) to control the process of fault coverage measurement.

The aim of structural coverage (e.g., transition coverage) is to measure to what extent test cases cover the specification model. Since any proposed fault model cannot guarantee specifying all faults, the use of structural coverage should not be ignored (Hessel et al., 2008; En-Nouaary et al., 1999).

On the other hand, achieving coverage criteria with a large number of test cases is not desirable. Measuring the length of the test suite generated by a testing approach is considered of paramount importance. The aim is thus for a testing approach which achieves high fault and structural coverage with fewer test cases.

4.5.1 Structural Coverage Assessment Criterion (CRC)

Recalling the idea and the equations of CRC discussed in Chapter 3, the clock regions coverage CRC achieved by a testing approach for the whole specification model can be calculated according to Equation (4.1).

$$CRC = \frac{\sum_{k=1}^Q \frac{CRC_k}{NCR'_k}}{Q} \quad (4.1)$$

Where:

- Q : The total number of input transitions in a specification model. Output transitions are excluded since the testing approaches used in this study equally cover the combined region of each output transition once it is fired.
- NCR'_k : The total number of feasible clock regions calculated for a transition k according to Equation (3.5).
- CRC_k : The actual number of clock regions that have been covered by all occurrences of transition k in the generated test cases.

In other words, CRC represents the average value of clock regions coverage calculated for all input transitions. If all timing constraints over transitions are similar in length, the average method in calculating the overall CRC for each model is reasonable. In the case of timing constraints with a large difference in length (e.g., $x < 50$, $x < 5$), weighted averages where different weights are assigned to CRC for each transition would be a preferable technique to use.

4.5.2 Fault Coverage Assessment Criterion (MAT)

Identifying how many faults can be detected by a test suite is known as *fault coverage*. Fault coverage should be supported with well identified faults that are defined in a fault model and which might be encountered in an implementation. The power of any test suite can be determined by its fault coverage; the higher the fault coverage, the more powerful the test suite (En-Nouaary and Hamou-Lhadj, 2008; En-Nouaary et al., 1999). The use of fault coverage as an assessment criterion can be more effective if it is used in a controlled way by the application of the Mutation Analysis Technique (MAT).

MAT was proposed to increase the confidence about SUT correctness. It is based on simulating real faults in an SUT to validate or identify adequate test data capable of revealing such faults. Mutants (i.e., faulty versions of an SUT) are produced by syntactically changing an SUT according to rules given by mutation operators. Each mutation operator is thus linked with the fault we need to reveal in an SUT. In the second stage, the generated mutants are executed using a given test suite. If a mutant shows different behaviour from the correct version of an SUT, the mutant is *killed* and the fault is identified. Otherwise, it is said that the mutant is *alive*. In other words, the test suite is not capable of killing the mutant due to the inadequacy of the test suite or the mutant being equivalent to the SUT. The equivalence relation implies that the SUT and the generated mutant should show same behaviour for the entire input domain. A mutation analysis oracle seeks to achieve a high mutation adequacy score (DeMillo et al., 1978).

To facilitate the application of fault coverage assessment using MAT, a set of mutation operators representing timed and functional faults that might be encountered in an SUT is introduced. Considering the similarity in structure between timing constraints defined in the specification model and clock conditions defined in the SUT C code, leads us to adopt a modified version of the TA-based mutation operators proposed in Chapter 3 (Section 3.6.1) to obtain C-based mutation operators. The mutation operators are divided into two main classes; timed and functional mutation operators. First, timed mutation operators include all operators relating to timing faults and comprises five types of operators.

- *Narrowing Clock Conditions* (NCC): This class of timed operators targets the conditions on clocks or timers defined within the SUT C code. They narrow down a condition bounds or change its relational operators ($\leq, <, =, >, \geq$) by which it rejects inputs originally accepted. For instance, this operator can be applied on the condition $a \leq x \leq b$ by changing either of its bounds ($a + \varepsilon_1 \leq x, x \leq b - \varepsilon_2$) or its relational operator ($a < x, x < b$); where $x \in C$ is a clock, $a, b \in \mathbb{N}$ are the bounds of the

condition and $\varepsilon_1, \varepsilon_2 \in \mathbb{N}_{>0}$ represent the syntactical changes applied to the condition.

- *Expanding Clock Conditions (ECC)*: This class of timed operators broadens the bounds of a clock condition or changes its relational operators by which it accepts inputs originally rejected. For instance, this operator can be applied on the condition $a < x < b$ by changing either of its bounds ($a - \varepsilon_1 < x, x < b + \varepsilon_2$) or its relational operator ($a \leq x, x \leq b$).
- *Shifting Clock Conditions (SCC)*: This class of timed operators depends on increasing/decreasing both bounds of a clock condition. For instance, this operator can be applied on the condition $a \leq x \leq b$ by increasing both of its bounds ($a + \varepsilon_1 \leq x \leq b + \varepsilon_2$) or decreasing them ($a - \varepsilon_1 \leq x \leq b - \varepsilon_2$).
- *Adding a new Starting Point of a clock (ASP)*: This timed operator involves adding a new starting position of the clock or timer controlling SUT timing behaviour.
- *Removing an existing Starting Point of a clock (RSP)*: This operator involves removing a starting position of an existing clock or timer controlling SUT timing behaviour.

Second, functional mutation operators include all operators related to functional faults and comprise two types of operators.

- *Exchanging Input Parameters of a method (EIP)*: This operator involves exchanging a predefined input parameter in a function or procedure with another one from the input set in the SUT.
- *Exchanging Output Parameters of a method (EOP)*: This operator involves exchanging a predefined output parameter in a function or procedure with another one from the output set in the SUT.

After obtaining the adequacy score for each operator, we can calculate fault coverage FC for a testing approach using Equation (4.2).

$$FC = \frac{\sum_{k=1}^w AS_k}{w} \quad (4.2)$$

Where:

- AS_k : The adequacy score calculated according to Equation (3.6) for each mutation operator k .
- w : The total number of mutation operators used. In our study $w = 7$.

In other words, Equation (4.2) gives the average number for all adequacy scores calculated for the mutation operators. Again, weighted averages can be used if the number of mutants differs largely from one mutation operator to another.

4.5.3 Test Traces Length Assessment Criterion (TTL)

Testing in general suffers from a high cost of test generation and execution. One of the most salient factors affecting the testing cost is the number of test cases (i.e., test traces). To clarify, more tests need more time to be generated and executed. Moreover, timed testing requires the generation and executing of test cases with different time delays. As a result, more tests require more time delays and, accordingly, cost more.

It is therefore desirable to find a small test suite that detects the most number of faults. In timed MBT, each test trace is generated as a sequence of timed actions covering a set of transitions at certain times. The same transition might then be a part of different test traces but with different clock delays. Different test traces might have different lengths. As a result, the total length of the generated test traces is calculated according to Equation (4.3). The lower the length of generated test traces, the more effective the testing approach is with respect to the cost:

$$\text{TTL} = \sum_{k=1}^n |\text{test trace}|_k \quad (4.3)$$

Where:

- n : The total number of test traces
- $|\text{test trace}|_k$: The count of (d, a) occurrence in the k_{th} test trace.
- $d \in \mathbb{R}_{\geq 0}$: A time delay.
- $a \in A$: An action.

4.5.4 Combined Assessment Factor (AF)

Any testing approach can be assessed according to each of the aforementioned assessment criteria. However, one testing approach can be effective according to one assessment criterion and not effective according to others. We thus introduce the Assessment Factor (AF) to combine all previous assessment criteria; CRC, FC and TTL. We are interested in identifying a testing approach that achieves the highest score with respect to all assessment criteria; high fault coverage, high clock region coverage and minimum length of generated test traces. AF can be represented mathematically according to Equation (4.4).

Since the CRC and FC range between (0, 1), AF will give a very small number. The AF result is thus scaled up 1000 times to be more recognisable. The experimental evaluation will be based on each individual assessment criterion (CRC, FC and TTL) as well as the combined criterion (AF).

$$\text{AF} = 1000 \times \frac{\text{CRC} \times \text{FC}}{\text{TTL}} \quad (4.4)$$

4.6 Empirical Assessment based on a Complete Test Bed

This section introduces the empirical validation of three TA-based testing approaches (including PA) based on the introduced assessment criteria using a complete test bed. Two out of the four introduced in Chapter 3 (Section 3.6.4) were chosen for this study (SM and BCT). The rationale for excluding the other two (SCT and COVER) is as follows. First, SCT generate a relatively large number of test cases compared with the others. Most importantly, SCT is not supported with an automation tool. The time needed for executing the large number of SCT test cases on the test bed manually is significant. The time needed to input the generated test cases into the GeTeX execution engine for automating the execution is also significant. Second, the results from the previous chapter suggest that COVER is not as good as other approaches due to the un-timed coverage criteria it uses for generating test cases. The test bed used for validating PA in comparison with SM and BCT, the specification models and the assessment results are presented and analysed in the following subsections.

4.6.1 Production-Cell Test Bed

We were given access to an industrial-strength production-cell lab in order to execute test cases generated by the testing approaches used in this study. All documents including the production-cell design and software design models were given. Different visits were also arranged to discuss the production-cell structures with the design engineers in the case of any missing piece of information.

A production-cell is a RTES consisting of two robots (robot-in and robot-out), a conveyor and a control panel. Figure 4.8 shows the physical layout of the cell.

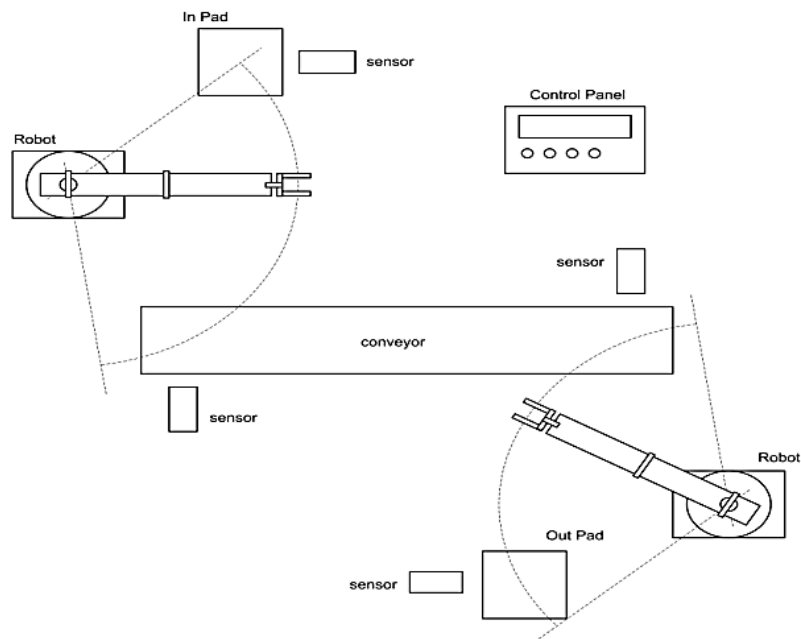


Figure 4.8: Production-cell physical layout

At its simplest level, the robot-in is responsible for picking up a Work Piece (WP) from the load platform (in-pad) and placing it on the conveyor. The item passes along the conveyor until it reaches the exit point to be ready to be picked off. The robot-out picks the item from the conveyor exit point and places it on an out-pad. The control panel allows an operator to supervise the system. There are a number of sensors positioned to detect items as they pass through the cell. The sensors are associated with the various components to form subsystems; each subsystem is managed by a micro-controller. The micro-controllers are connected by a CAN communication network to coordinate actions of the components and move items through the production-cell. Figure 4.9 gives a schematic overview of the system (Robson and Henderson, 2010).

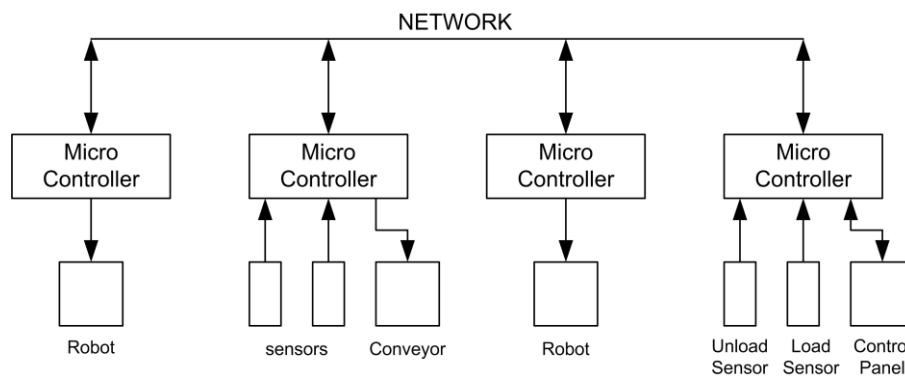


Figure 4.9: Production-cell schematic

4.6.2 Specification Models

A production-cell is a real-time distributed system consisting of four components communicating via a CAN bus. Figure 4.10, Figure 4.11, Figure 4.12 and Figure 4.13 represent the specification models of load, unload, conveyor load and conveyor unload sensors, respectively for identifying the position of a WP within the cell.

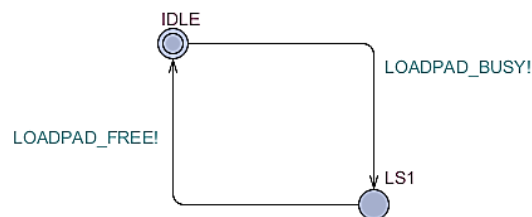


Figure 4.10: Load sensor automaton

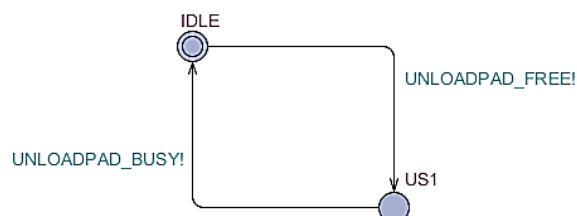


Figure 4.11: Unload sensor automaton

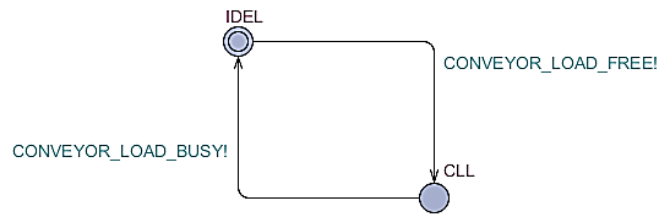


Figure 4.12: Conveyor load sensor automaton

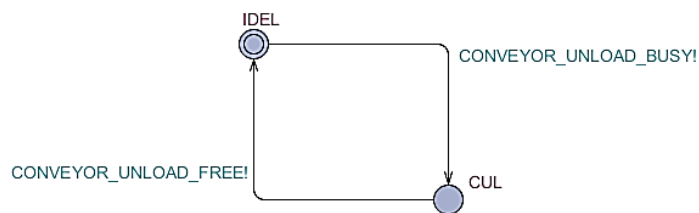


Figure 4.13: Conveyor unload sensor automaton

Load and unload sensors identify whether the WP is picked up from the in-pad (a place where a WP enters the cell) or deposited in the out-pad (a place where a WP leaves the cell). Conveyor load and conveyor unload sensors identify the location of the WP in the conveyor.

Figure 4.14 represents the specification model of the control panel. Receiving the signal from the load sensor, the control panel knows that the WP is loaded. It then informs the robot-in to pick up the WP from the in-pad. When receiving a signal from robot-in within 1-5 seconds querying whether it succeeds in picking up the WP, the control panel waits for a signal to be received from the sensor to be able to send the confirmation to the robot-in. Before depositing the WP into the out-pad, the robot-out should ask the control panel within 36-63 seconds to know if the out-pad is free. In turn, the control panel sends the confirmation to the robot-out once it receives a signal from the unload sensor stating that the out-pad is free. Another confirmation will be sent to the robot-out when it succeeds in depositing the WP in the out-pad within 12-15 seconds.

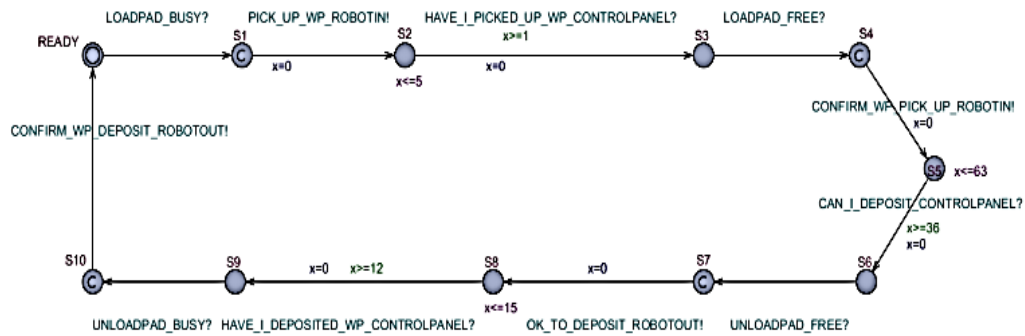


Figure 4.14: Control panel automaton

Figure 4.15 depicts the specification model of the conveyor. The conveyor allows the robot-in to deposit the WP if the robot-in asks to and the sensor does not detect another WP occupying its place.

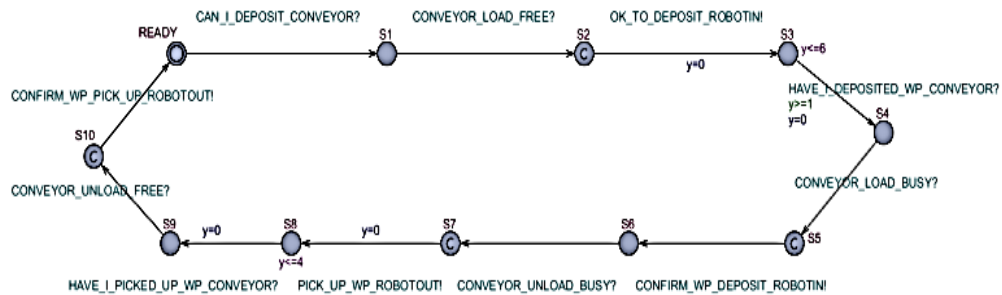


Figure 4.15: Conveyor automaton

The conveyor will send a confirmation signal if a signal is received from the robot-in within 1-6 seconds to indicate whether the WP has been deposited. The WP will move through the conveyor until reaching the end point when triggering a signal by the sensor to robot-out. The conveyor will broadcast a confirmation if the robot-out picks up the WP within 4 seconds.

Figure 4.16 represents the specification model of the robot-in component. Picking up the WP from the in-pad, the robot-in asks the control panel for a confirmation within 1-10 seconds. Once it obtains the pickup confirmation within 7 seconds, the robot-in will ask within 1-3 seconds if the conveyor is free to collect the WP.

A free-to-deposit confirmation should be received within 4 seconds for the robot-in to be able to ask the conveyor within 1-6 seconds if the WP is successfully deposited. The confirmation is then broadcasted.

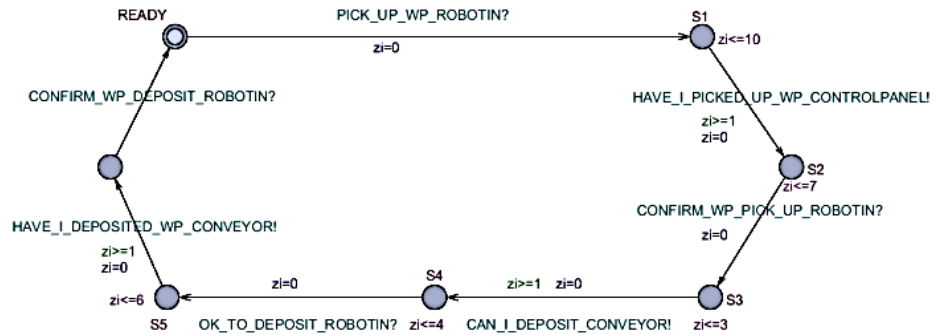


Figure 4.16: Robot-in automaton

Figure 4.17 depicts the specification model of the robot-out. Picking up the WP from the conveyor, the robot-out asks the conveyor for a confirmation within 15 seconds.

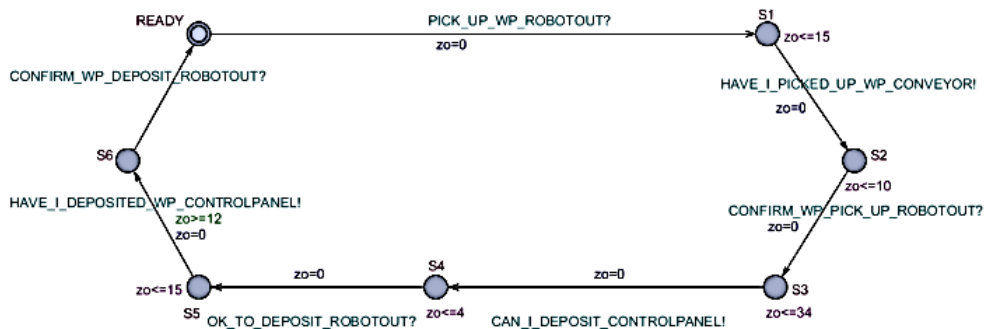


Figure 4.17: Robot-out automaton

Once getting pickup confirmation within 10 seconds, the robot-out should ask within 34 seconds if the out-pad is free to get the WP. A free-to-deposit confirmation should be received within 4 seconds for the robot-out to be able to ask the control panel within 12-15 seconds if the WP has been successfully deposited. The confirmation is then broadcasted.

4.6.3 Test Generation and Execution

Using the specification models, test cases according to PA, SM and BCT were generated for each component of the production-cell. The process of test generation according to PA was automated using the GeTeX tool. However, test cases according to SM and BCT were manually generated from the specification models; the restricted UTA models used without data are similar to those of TIOA. The length of test cases generated by each approach calculated according to Equation (4.3) is given in Table 4.5 in the TTL row. The CRC for each testing approach calculated using Equation (4.1) is presented in Table 4.5 in the CRC row. A detailed calculation of CRC for each approach is presented in Appendix B.

To run the generated test cases on production-cell components, a set of preparatory procedures were undertaken. First, using GeTeX, the generated test cases from three approaches were transformed into executable inputs interacting with SUT components. Second, production-cell components were disconnected since the intention was to perform component-based testing. Considering that production-cell components communicate via the CAN bus, the PC hosting GeTeX was connected to the CAN bus using a USB/CAN adapter to interface GeTeX with the production-cell. GeTeX replaced the communications required for each component to perform its jobs by injecting a suitable sequence of CAN messages according to a testing approach. Third, for calculating the FC assessment criterion, test cases from each testing approach were executed on each component of the production-cell; control panel, conveyor, robot-in and robot-out. GeTeX recorded the responses from each component to compare them with those of the specification. *Pass/fail* verdicts were then assigned if the components' response did or did not conform to the specification according to *tioco*, respectively.

The C code controlling each component was manually mutated according to the proposed operators for calculating fault coverage (FC) for each testing approach. For each operator, all possible mutants were generated. Studying the mutants' C

code of each operator, equivalent mutants were identified for NCC in the robot-in and robot-out components and for ASP and RSP in all production-cell components. Once a mutant was loaded into the micro-controller, all test cases were re-executed on the component under test. Table 4.1, Table 4.2, Table 4.3 and Table 4.4 depict, for each operator, the number of generated, equivalent, killed mutants and mutation score for control panel, conveyor, robot-in and robot-out, respectively.

Approaches	Operators	Mutants	Equivalent	Killed	Score
PA	NCC	73	0	64	0.88
	ECC	27	0	27	1
	SCC	36	0	36	1
	ASP	N/A	-	-	-
	RSP	6	3	3	1
	EIP	42	0	42	1
	EOP	12	0	12	1
SM	NCC	73	0	56	0.77
	ECC	27	0	0	0
	SCC	36	0	24	0.67
	ASP	N/A	-	-	-
	RSP	6	3	3	1
	EIP	42	0	42	1
	EOP	12	0	12	1
BCT	NCC	73	0	56	0.77
	ECC	27	0	0	0
	SCC	36	0	24	0.67
	ASP	N/A	-	-	-
	RSP	6	3	3	1
	EIP	42	0	42	1
	EOP	12	0	12	1

Table 4.1: MAT Application on the control panel

Approaches	Operators	Mutants	Equivalent	Killed	Score
PA	NCC	65	0	59	0.91
	ECC	12	0	12	1
	SCC	16	0	16	1
	ASP	N/A	-	-	-
	RSP	4	2	2	1
	EIP	42	0	42	1
	EOP	12	0	12	1
SM	NCC	65	0	49	0.75
	ECC	12	0	0	0
	SCC	16	0	12	0.75
	ASP	N/A	-	-	-
	RSP	4	2	2	1
	EIP	42	0	42	1
	EOP	12	0	12	1
BCT	NCC	65	0	49	0.75
	ECC	12	0	0	0
	SCC	16	0	12	0.75
	ASP	N/A	-	-	-
	RSP	4	2	2	1
	EIP	42	0	42	1
	EOP	12	0	12	1

Table 4.2: MAT Application on the conveyor

Approaches	Operators	Mutants	Equivalent	Killed	Score
PA	NCC	69	33	27	0.75
	ECC	33	0	33	1
	SCC	44	0	44	1
	ASP	N/A	-	-	-
	RSP	6	1	5	1
	EIP	12	0	12	1
	EOP	6	0	6	1
SM	NCC	69	33	25	0.69
	ECC	33	0	27	0.81
	SCC	44	0	44	1
	ASP	N/A	-	-	-
	RSP	6	1	3	0.6
	EIP	12	0	12	1
	EOP	6	0	6	1
BCT	NCC	69	33	25	0.69
	ECC	33	0	27	0.81
	SCC	44	0	44	1
	ASP	N/A	-	-	-
	RSP	6	1	3	0.6
	EIP	12	0	12	1
	EOP	6	0	6	1

Table 4.3: MAT Application on the robot-in

Approaches	Operators	Mutants	Equivalent	Killed	Score
PA	NCC	69	33	27	0.75
	ECC	21	0	21	1
	SCC	28	0	28	1
	ASP	N/A	-	-	-
	RSP	6	1	4	0.8
	EIP	12	0	12	1
	EOP	6	0	6	1
SM	NCC	69	33	25	0.69
	ECC	21	0	15	0.71
	SCC	28	0	28	1
	ASP	N/A	-	-	-
	RSP	6	1	3	0.6
	EIP	12	0	12	1
	EOP	6	0	6	1
BCT	NCC	69	33	25	0.69
	ECC	21	0	15	0.71
	SCC	28	0	28	1
	ASP	N/A	-	-	-
	RSP	6	1	3	0.6
	EIP	12	0	12	1
	EOP	6	0	6	1

Table 4.4: MAT Application on the robot-out

To calculate the FC for each component, the average mutation scores obtained for all operators per production-cell component was calculated according to Equation (4.2). Table 4.5 clarifies the fault coverage outcomes for each testing approach per component in the FC row. According to CRC and FC results, the assessment factor (AF) was calculated using Equation (4.4); AF is presented in Table 4.5.

Assessment Criteria	Testing Approaches	Control Panel	Conveyor	Robot-in	Robot-out
TTL	PA	732	168	136	184
	SM	96	48	32	32
	BCT	84	60	40	40
FC	PA	0.95	0.96	0.93	0.9
	SM	0.71	0.79	0.86	0.82
	BCT	0.71	0.79	0.86	0.82
CRC	PA	1	1	1	1
	SM	0.07	0.12	0.11	0.1
	BCT	0.1	0.14	0.16	0.13
AF	PA	1.3	5.71	6.84	4.89
	SM	0.52	1.98	2.96	2.56
	BCT	0.85	1.84	3.44	2.67

Table 4.5: Assessment results

4.6.4 Assessment Discussion

Comparing PA with SM and BCT according to fault coverage criterion (FC), we found that PA showed superiority and stability in detecting most faults injected into the production-cell components. FC of PA ranged from 90% in the robot-out to 96% in the conveyor. On the other hand, FC score of SM and BCT was less than that of PA; their FC ranged from 71% to 82% across all production-cell components.

To understand the high FC score achieved by PA in comparison with SM and BCT, the individual mutation score for each operator is discussed. Contrary to SM and BCT, PA maintained the full mutation score '1' for ECC and SCC, because of the selection of time points that can detect such faults. However, FC of PA was negatively affected by the mutation score of NCC. Selecting the boundary points of clock conditions was insufficient to detect the entire injected faults for several reasons. First, the TA model might end with an input transition such as the transition (s_6, s_0) in the robot-out automaton (Figure 4.17). All test cases generated by the approaches under study finished at the initial location. In other words, the input transition will be the last transition in a test trace. Since any injected faults require outputs to be detected, there is no possibility of detecting any faults injected into code representing such a transition. Second, the TA model might contain an unconstrained transition. The fault as a result of mutating the code with a new clock condition might be undetectable by the time points chosen by PA. For instance, the fault resulting from adding a time condition $(x < 50)$ to the code representing the transition (s_0, s_1) in the robot-in automaton (Figure 4.16) is undetectable by PA. Third, detecting some faults under the NCC category requires sending the SUT an input at an exact time point (e.g., replacing $x \leq 4$ with $x < 4$). However, the accuracy of the clock used in testing process and the uncontrolled delay through the communication with the SUT does not guarantee that the SUT will receive an input at the same time point as intended by the tester.

Detecting faults under the RSP category depends on the position of a clock reset, the consecutive transitions and the length of timing guards constraining them. PA empirically showed higher capabilities in detecting such faults than SM and BCT. The ASP operator was not considered as it only produced equivalent mutants.

With respect to the functional mutation operators, PA shared a full mutation score with SM and BCT; the full transition coverage achieved by all is considered to be sufficient to detect all functional faults injected into the implementation C code.

The high CRC of PA compared with SM and BCT relied on the full clock regions achieved by PA. The low CRC score of SM and BCT arise from restricting the selection of time points to cover only two regions in the case of SM and three regions in the case of BCT. Their target was to dramatically reduce the cost by minimising the number of generated test cases. That is clear from the low TTL in both cases. The few test cases generated by SM or BCT were capable of detecting 82% of the faults injected as a best result. However, selecting them for testing hard, real-time or safety critical systems is still questionable due to the shortage in structurally covering SUT behaviour.

The importance of timed structural coverage comes from the possibility of faults existing in the SUT un-categorised by the fault model. However, the high score in structural coverage usually correlates with a higher cost in terms of the number of generated test cases or TTL. As a result, any testing approach that can combine a high FC score and CRC score with a relatively small number of test cases (achieve high AF) is preferable. Figure 4.18 shows that PA performed much better than SM and BCT, in terms of AF, for all production-cell components although it produces relatively larger test cases than the other testing approaches. However, PA did not maintain the same AF score ranging from '1.3' for control panel to '6.84' for robot-in due to the differences in TTL generated for each of the components.

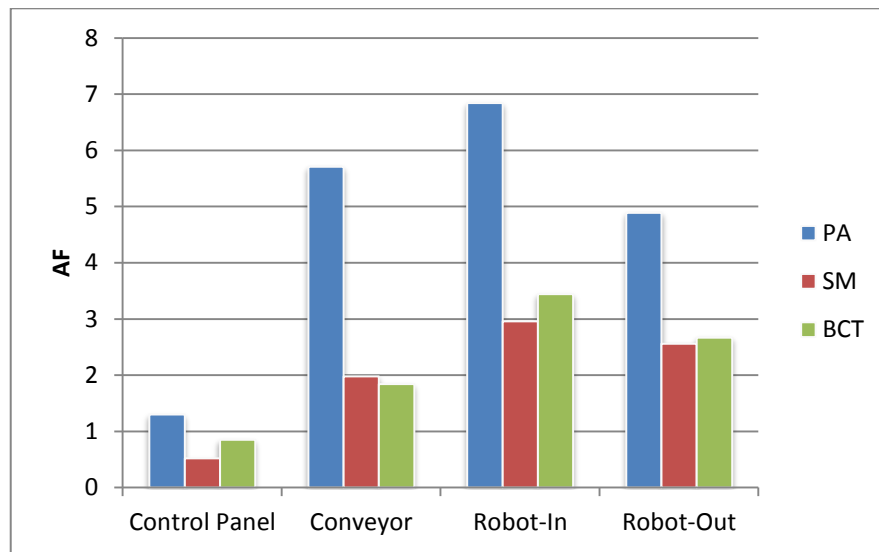


Figure 4.18: AF factor of each testing approach according to production-cell components

4.6.5 Lessons Learned and Problems Encountered

While running the production-cell test bed, several issues affecting real-time testing were noted. In this section, a summary of those issues is presented as a step towards facilitating more empirical, real-time model-based test beds.

To begin with, the specification models do not always represent the code. In this study, we had the opportunity to study the SUT code in order to mutate them. The code based on the real-time operating system kernel (Micro C) contains more functions than those represented in the specification models. The MBT approaches used did not guarantee testing all functions in the code. To avoid this problem, we assumed that the SUT was fully represented by the specification model. However, in reality, this problem is still an issue.

Moreover, synchronising clocks between GeTeX and the SUT was another issue. To clarify, transitions in TA models are instantaneous (i.e., the time of triggering a transition is 0). However, triggering transitions consumes time in actuality. The time delay that needs to be considered occurs at the implementation level (code

execution time of the GeTeX) and at the physical layer (CAN bus). The more accurate the time delay calculated, the more synchronised the clocks are in GeTeX and the SUT. In this study, theoretical and empirical methods were followed to estimate time delay. First, the CAN bus delay was calculated using the propagation delay equation (suggested by the controller data sheet) taking into account the bus length, the CAN controller, transceiver ports and its baud rate. Second, the CAN bus delay was measured by developing echo software between two nodes. The first node broadcasts a message at a specific time point. The second node replicates the message once it is received. When the first node receives the replicated messages, it records its time. The bus delay is calculated as half the time required for a message to be sent and received at the first node. The theoretical and empirical results were similar (10 ms). Moreover, the code execution time of GeTeX was measured using some Java libraries (i.e., Nano-time and calendar). The soft and physical time delays were compensated by GeTeX when sending inputs to the SUT at specific times. In spite of this compensation, it was not guaranteed that an input was received by the SUT at an exact testing time point. This would diminish the testing approaches capability of detecting the boundary faults as indicated by the relatively low NCC score of the testing approaches.

Lastly, the use of clocks either by the testing tool or SUT was another problem encountered in testing real-time systems. This issue is related to clock accuracy. In this study, the time units used were in seconds. The accuracy of timers to track time progress was found to be dependent on the hardware specification as well as the software. For instance, the Micro C operating system used in the micro-controllers cannot measure to less than 1 ms. The clock accuracy within the experiment was found to be ± 3 ms. When a timing constraint ($x \leq 3$ for instance) is tested at its boundary value by sending the SUT an input message at 3 seconds, the SUT could receive the input at 3.003 seconds which does not satisfy the clock condition. The transition is thus not triggered and an incorrect failure will be emitted. Another example of a clock accuracy issue was found when mutating a

timing constraint by changing its boundary type ($x \leq 3$ for instance becomes $x < 3$). This fault cannot be detected unless an input message is sent to the SUT at 3 seconds exactly. To remedy such problems, GeTeX was instructed to accept 3 ms allowance for each message sent or received. Although (3 ms) is very little amount compared with seconds, the testing approaches will be unable to detect timing faults occurring in this allowance interval. To minimize the clock accuracy issue, a more accurate hardware timer could be employed.

4.7 Summary

This chapter introduced GeTeX as a new timed testing tool for CAN applications. GeTeX can be considered as a complete tool that tests timing properties of an RTES in particular. GeTeX depends on PA for generating timed test cases from a system specification modelled as UTA. GeTeX also depends on *tioco* theory in executing the timed test cases and assigning *pass/fail* verdicts to them. The practicality of using GeTeX was shown by experimenting with a light controller prototype. The tool generated and executed the test cases in a short time without any compilation errors.

This chapter also introduced an empirical test bed using production-cell case study and assessment criteria to validate the PA testing approach in comparison with two TA-based testing approaches (SM and BCT). The first assessment criterion includes formulating timed structural coverage represented by clock region coverage (CRC). A set of timed and functional mutation operators was presented to facilitate the second assessment criterion (FC). An assessment factor (AF) that considered fault coverage and clock coverage with respect the length of generated test cases was also presented. The experiments confirm the results collected for Chapter 3. PA performed better than the others in terms of FC or CRC even though it produced relatively larger test cases than the other testing approaches; salient problems encountered during conducting the empirical study were highlighted.

Chapter 5: A Multi-Criteria Decision Making Approach for Prioritising the Test Sets of the Priority-Based Approach

5.1 Overview

In the previous chapters, the Priority-based Approach (PA) which tested logical and timing behaviour of an RTES modelled formally as UPPAAL Timed Automata (UTA) was introduced and automated by the GeTeX tool. PA generated three separate sets of test cases (i.e., boundary, out-boundary and in-boundary) to enable the tester to choose between the proposed test sets (or any combination thereof). However, selecting the ‘best-suited’ test set to be deployed for a certain application in a particular organisation lacks the rigour that a systematic decision-making framework might offer.

This chapter fills this gap by developing a novel Analytical Hierarchy Process (AHP) as decision-making framework for PA. The framework provides testers with a systematic approach by which they can prioritise the available test sets that best fulfil testing requirements. The AHP framework developed is based on the data collected heuristically from the production-cell test bed and those collected by interviewing testing experts. The framework is then applied on two different testing scenarios to prove its validity by comparing the decision prioritising outcomes with those of the testing experts (Aboutrab et al., 2012a).

The remainder of this chapter is organised as follows. The problem area this chapter tackles is highlighted in Section 5.2. Section 5.3 gives an overview of

decision-making methods including that of AHP. In Section 5.4, the proposed AHP decision model is presented and explained. The process of prioritising the PA test sets using the AHP framework is also discussed. The framework is then validated using two testing scenarios in Section 5.5. Finally, Section 5.6 concludes the chapter.

5.2 Problem Area

Research in MBT methods has gained increasing attention especially for testing RTEs. This is due to MBT's ability to reduce testing cost by capturing and validating system behaviour from an early stage of the development cycle and using tools to automate the process of test case generation, execution and evaluation (Grieskamp et al., 2011). Many MBT algorithms and methods for testing real-time systems have been proposed over the last two decades (Cardell-Oliver, 2000; Clarke and Lee, 1997a; En-Nouaary, 2008; En-Nouaary and Hamou-Lhadj, 2008; Hessel et al., 2008; Larsen et al., 2005a; Merayo et al., 2008; Nielsen and Skou, 2003; Krichen and Tripakis, 2009; Hierons et al., 2009). Most testing approaches that achieve high fault coverage suffer from high cost in terms of expended effort and the large number of generated test cases (Mitsching et al., 2009). Choosing which approach most suits a testing project can therefore be considered as a problem for the following reasons.

1. The selection of a candidate testing approach is totally dependent on a tester's intention and experience.
2. Each testing approach provides a single test solution in which a tester's preferences or environmental factors affecting the testing process (e.g., available test time or budget) cannot be considered. In other words, a tester cannot guarantee whether choosing a particular subset of a test suite due to the shortage of test time (for instance) will provide the best testing outcome.

3. The existence of many factors that contribute to the testing process in different ways increases the complication of making the right decision such as choosing a testing approach with the aim of achieving high fault coverage with low cost.

To address such problems, the proposed PA automated by the GeTeX tool divides the generated test cases into three separate sets (i.e., boundary, out-boundary and in-boundary). PA thus enables the tester to choose between the proposed test sets (or any combination thereof). According to that choice, PA establishes a trade-off between increasing confidence in SUT correctness and limited testing resources such as time, effort and cost. However, selecting the ‘best-suited’ test set to be deployed for a certain application in a particular organisation by relying only on a tester’s intension is risky due to different environmental factors influencing the decision process. A formal decision framework in which all testing requirements and factors (decision criteria) affecting the testing process are independently categorised, weighted and analysed becomes viable.

An Analytical Hierarchy Process (AHP) (Saaty, 1977; Saaty, 1980) is a multi-criteria decision-making approach based on dividing the decision criteria into several levels to enable their pair-wise ranking subject to field experts or empirical data. AHP potentially reduces the complexity of the decision problem and allows consistent outcomes to be generated.

The *problem* tackled by this chapter is to prioritise the PA test sets for a particular testing project using the AHP multi-criteria decision-making method. The primary contributions of this chapter are:

- 1- The development of the AHP decision model considering criteria that might affect a tester’s decision in prioritising the PA test set for a particular testing project.
- 2- The development of the AHP framework with its process using the test data set obtained from the production-cell test bed and a group of testing experts.

- 3- The validation of the AHP framework using two different testing scenarios by checking the degree of similarity between the AHP decision outcomes with those of testing experts.

5.3 Preliminaries

A formal decision-making procedure is an essential tool for modern organisations. Dealing with complex environments with technological cutting-edge requirements increases the risk implications of any decision yet to be made on the future of any organisation (Saaty, 2001). Formal decision-making methods provide a structural process by which decisions are clear, justified, consistent and repeatable. The process of decision-making involves choosing a solution from a set of available solutions according to some decision criteria. It is based on ranking the solutions according to each criterion to obtain a decision by combining all rankings. The ranking process might include a group of expert opinions. This section presents an overview of well-known decision-making methods including the AHP.

5.3.1 Decision Making Methods

Several approaches have been developed to standardise the process of making decisions. Choosing an appropriate decision-making method is dependent on the type of the decision problem, the attributes of the decision-making method and the objectives of decision makers. The use of optimisation techniques can also lead to a greater deployment of decision-making methods (Bhushan and Rai, 2004) and the chosen method should thus be justified and evaluated (Baker et al., 2001). In general, the ease of use and applicability remain an issue for some approaches due to the heavy dependence on theoretical underpinnings or the inability to solve complicated decision problems. For instance, the Ranking Approach (Buss, 1983), a non-linear programming model (Badria and Davisb, 2001; Santhanam and Kyparisis, 1996), the 0-1 goal programming model and the Analytical Network Process (ANP) (Lee and Kim, 2000) are reliant on complicated mathematical

models and are difficult to understand and use. On the other hand, some decision-making methods support small decision problems where only a few decision criteria and solutions exist such as Pros and Cons analysis (Baker et al., 2001). For partially complex applications, Kepner-Tregoe (K-T) decision analysis (Kepner and Tregoe, 1981) can be used.

Numerous multivariate methods ignore decision makers' preferences in the process of decision-making (e.g., the Simple Multi-Attribute Rating technique (SMAR) (Salmeron and Herrero, 2005; Dutta and Burgess, 2003) and Decision-making Units (DMU) (Salmeron and Herrero, 2005)). DMU involves assessing the performance of different units that might be different in nature such as a computer or a school. Performance is measured considering the amount of inputs involved and outputs generated. The measures of unit performances are then compared in the sense that one unit is more efficient than another if it gives more outputs for same quantity of inputs or the same amount of outputs for smaller set of inputs. This comparison can be represented mathematically by ratio of the sum of outputs over the sum of inputs. The Data Envelopment Analysis (DEA) approach (Salmeron and Herrero, 2005) extends DMU by assigning different weights to outputs and inputs. The weights are different values assigned to reflect the fact that one unit is more important than others. DMU and DEA are preferable when there is no need to consider the preferences of decision makers as the main intention is to compare unit performances.

On the other hand, there are several methods that consider the decision makers' preferences such as the Multi-Attribute Utility Theory (MAUT) (Edwards and Barron, 1994; Goodwin and Wright, 1999) and the Analytical Hierarchy Process (AHP) (Saaty, 2001; Saaty, 1990a; Saaty, 1990b; Saaty and Kearns, 1985; Saaty, 2008; Saaty and Vargas, 2000; Saaty and Vargas, 1984; Saaty and Vargas, 1991). Firstly, MAUT is a quantitative decision-making method that depends on optimising measures of costs, benefits and risks for decision alternatives. The measures are then combined along with the preferences of the decision makers in

a cumulative format. Secondly, the AHP depends upon making decisions on pairwise ranking of decision alternatives according to decision criteria; this is done on the basis that humans are more skilled at making relative decisions than complete ones. Some researchers might not support the use of the AHP due to the way it numerates and processes the ranking values (Dutta and Burgess, 2003; Goodwin and Wright, 2000). However, comparing the AHP with some of its counterparts, Table 5.1 demonstrates its advantageous features.

Comparison Factors	Decision Making Techniques				
	AHP	SMAR	DEA	RA	ANP
Incorporation of preference structure	✓	–	–	–	–
Synthesised analysis of diverse judgements	✓	–	–	–	–
Is an intuitive technique	–	–	–	✓	–
Optimises resource allocation for interaction of factors	✓	–	✓	–	✓
Limited attributes to carry out real world decisions	–	✓	✓	✓	✓
Captures individual knowledge and experience	✓	✓	–	–	–
Gives easy understanding of the problem situation	✓	–	–	–	✓
Time-consuming process	–	–	–	–	–
Non-linear representation	–	–	–	✓	–
Managing large amount of qualitative/quantitative data	✓	–	–	–	–
Applicability weakened by complex mathematical models	–	–	–	✓	✓
Easy understanding of the prioritisation process	✓	✓	–	✓	–
Quick insight into structure of information	✓	✓	–	–	–
Requires less skill and training	✓	✓	✓	✓	✓
Measures the performance efficiency of decision makers	–	✓	✓	–	–
Structures through symbolic and numeric representation	✓	✓	–	–	–
Supports different viewpoints through rich pictures	✓	–	–	–	–
Techniques inappropriate for all situations	✓	✓	✓	✓	✓
Too much focus on quantifiable calculations	–	✓	✓	✓	✓
Provides a step-wise guideline for prioritising the factors	✓	–	–	–	✓
Accessible data format	✓	–	✓	–	–
Graphical representation	✓	–	–	–	–
Resolves complex problems of choice and prioritisation	✓	–	✓	–	✓

Table 5.1: Comparisons of decision-making approaches (Kamal, 2008)

5.3.2 Analytical Hierarchy Process (AHP)

Using rigorous mathematical rules, the AHP analyses the decision problem and structures the experience, preference, intuition and heuristics of the decision makers (Huang et al., 2004). Due to its simplicity and organised structure, the AHP is suitable for a wide range of applications including alternative selection (Zeng et al., 2007), resource allocation (Ramanathan, 1995), forecasting (Ülengin, 1994; Jensen, 1982; Jensen and Spencer, 1986; Saaty, 1987), business process re-engineering (Ashayeri et al., 1998; Wei et al., 2005), quality function deployment (Karsak et al., 2003), balanced scorecard (Ravi et al., 2005), benchmarking (Lu et al., 1994), public policy decisions (Saaty, 2001), healthcare (Dolan, 1989), multimedia communication (Ghinea et al., 2005), software testing (McCaffrey, 2005) and many more. AHP results are always compatible with expectations regardless of the type of applications. As a result, the AHP is an accepted method (Saaty, 2008).

The AHP has several features and characteristics making it more preferable than other decision-making approaches. Firstly, the AHP qualitatively decomposes the decision problems to a set of sub-problems and unrelated factors organised in a hierarchical structure in which every set of factors is classified under a certain decision sub-problem. As a result, the assessment bias can be significantly reduced (Chin et al., 1999; Cheng and Li, 2002). The multi-criteria format enables the AHP to use a pair-wise comparison mechanism in ranking the decision factors quantitatively. The ranking process thus becomes more informative and accurate and represents the importance of decision factors with respect to others (Salmeron and Herrero, 2005; Saaty, 1980; Jackson, 2001). Secondly, the AHP is equipped with consistency assessments to minimise any inconsistency within the rating of decision makers (Salmeron and Herrero, 2005; Saaty, 1980; Jackson, 2001). Thirdly, the AHP uses an appropriate measurement scale making the judgements logical and comprehensive (Lai et al., 1999). Fourthly, the AHP outcomes are

determined by prioritising a set of decision alternatives according to the relative ranking of the decision criteria (Wei et al., 2005; Saaty, 1990b).

The AHP process comprises several steps (Saaty and Vargas, 2000):

Step 1 - Constructing the Hierarchy Model: In this step, the decision problem is defined and the decision factors are categorised into a hierarchical model comprising goal, criteria, sub-criteria and alternatives. The decision goal forms the root of the model where the decision alternatives form the leaf nodes. The root and the leaves are connected by various levels (criteria and sub-criteria) where the relationship between elements of one level with those of other levels are indicated and classified.

Step 2 - Ranking Decision Factors through Pair-Wise Comparisons: The importance of each decision factor is determined relative to all other factors. This is considered an easy and efficient way of obtaining actual priorities. The comparison process needs to be made for elements at a certain level within their own criterion. The ranks can be collected from heuristics, decision makers or field experts and then converted to numbers according to a nine-point scale introduced by Saaty (Saaty, 1977). Table 5.2 illustrates the scale and its meanings. The numerical rating is not dependent on a standard scale but represents the preference relationship established between the factors being compared.

Pair-wise comparisons can be done in different ways. Interviewing a group of field experts can be considered one of the most popular means of obtaining numerical rates. We denote W_{AB} as the preference of the factor A with respect to the factor B and $(1/W_{AB})$ as the preference of the factor B with respect to the factor A where A and B belong to the same decision criterion. This procedure helps to decrease the number of ratings to $n(n-1)/2$ where n represents the number of factors under a decision criterion (Salmeron and Herrero, 2005). Use of heuristics is another way for obtaining the rating. The absolute data collected for each factor needs to be mathematically normalised to a nine-point scale.

Numerical Rating	Verbal Judgments of Preferences
1	A is equally preferable to B
2	A is equally to moderately preferable to B
3	A is moderately preferable to B
4	A is moderately to strongly preferable to B
5	A is strongly preferable to B
6	A is strongly to very strongly preferable to B
7	A is very strongly preferable to B
8	A is very strongly to exceptionally preferable to B
9	A is exceptionally preferable to B

Table 5.2: Pairwise comparison scale for AHP preferences (Saaty, 1977)

Step 3 - Creating Comparison Matrices: The pair-wise rates for different decision criteria at a certain level in the hierarchical model are arranged in a square matrix ‘A’ as depicted in Equation (5.1). Each element a_{ij} in the matrix represents the preference of the factor in a row i to the factor in a column j . All diagonal elements are thus equal to 1. Moreover, all elements in the upper triangle of the square matrix represent the reciprocal of the elements in its lower triangle.

$$A = \begin{bmatrix} 1 & \dots & a_{ij} \\ \dots & 1 & \dots \\ 1/a_{ij} & \dots & 1 \end{bmatrix} \quad (5.1)$$

Step 4 - Calculating Eigenvectors: This step involves decomposing the comparison matrix containing the relative ranking values into a non-zero vector representing the absolute weights of decision criteria, sub-criteria or alternatives. The transformation of relative ranks (i.e., in pair-wise comparison matrices) to an absolute weights can be considered as an eigenvalue problem. As a result, calculating the largest positive eigenvalue for pair-wise matrices with associated eigenvector leads to a vector of weights. Since the point of using the AHP is to

prioritise a set of solutions, Saaty (2008) found that the calculation of eigenvectors can be approximated without largely affecting the results of that prioritization.

The process of calculating the approximate eigenvectors involves normalising the comparison matrix by dividing each element by the sum of its column. The sum of each row of the normalised matrix is then divided by the number of its elements to obtain the approximate eigenvector.

Step 5 - Calculating a Consistency Ratio: Ranking the decision factors using a group of experts being interviewed raises a consistency issue (i.e., whether all ranks are consistent with each other). The use of comparison matrices eliminates symmetric inconsistencies due to reciprocal elements with respect to the matrix diagonal. However, the transitive consistency property may not be satisfied. In other words, if A is more important than C and C is more important than B, it is not known if A is more important than B. As a result, the consistency ratio of the comparison matrix of order n needs to be calculated and evaluated. The closer the consistency ratio is to zero, the more consistent the matrix. The AHP tolerates inconsistency to a certain degree due to the amount of redundancy in the framework. To accept the pair-wise ranking, the value of consistency ratio should not exceed 10%. If it is found that the consistency ratio exceeds the 10% level, the judgments made are ineffective as they become too similar to random judgments. As a result, the rating process may need to be re-done since the decision makers are inconsistent in their ratings (Saaty, 2008). The consistency ratio CR can be calculated according to Equation (5.2).

$$CR = CI / RI \quad (5.2)$$

Where:

- CI represents the consistency index calculated according to Equation (5.3).

- *RI* represents the random matrix depicted in Table 5.3. The chosen value of *RI* should correspond to the order of a comparison matrix (e.g., *RI* = 0.58 for three-dimensional comparison matrix).

$$CI = (max-n) / (n-1) \tag{5.3}$$

Where:

- *max* represents the maximum eigenvalue of the pairwise matrix.
- *n* represents the order of a comparison matrix.

Order of the matrix <i>n</i>	1	2	3	4	5	6	7	8	9	10
Random Consistency Index – <i>RI</i>	0	0	0.58	0.89	1.11	1.25	1.35	1.40	1.45	1.49

Table 5.3: Random consistency indices (Saaty, 1990a)

Step 6 - Determining Normalised Weights: This step involves prioritising the decision alternatives according to the calculated weights. The global weight of a sub-criterion is calculated by multiplying the weight of the decision criteria it belongs to by its local weight. The weights of alternatives are calculated with respect to a sub-criterion by multiplying the weight of each alternative by the global weight of that sub-criterion. The alternative weights are then aggregated to obtain the final rating by which they are prioritised.

Step 7-Integrating Group Judgments: If the ranking process includes several experts to be interviewed or several experiments to be run, the results are integrated using the geometric mean approach since the ranks are represented by a geometric scale (Saaty, 2008).

5.4 The AHP Framework

Figure 5.1 depicts the hierarchical AHP model introduced to solve the decision problem under study. The root of the hierarchy is the definition of the decision problem (decision goal). The leaf nodes represent the decision alternatives (i.e., solutions) to be prioritised according to decision criteria and sub-criteria. In the following, the proposed AHP model is defined and explained.

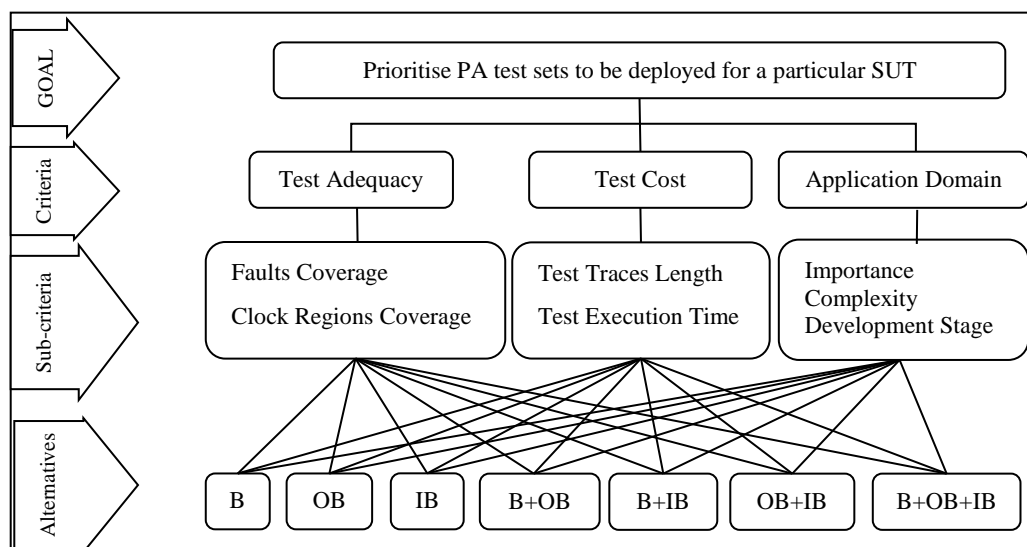


Figure 5.1: AHP hierarchal model

5.4.1 Decision Problem

The Priority-based Approach (PA) was proposed for testing logical and timing behaviour of an RTES modelled formally as UTA. The core concept of the PA is based on dividing the generated test cases into three sets (i.e., priorities) as the priority of choosing a particular test set differs according to several factors such as the testing environment specified by the criticality of SUT, the allowable time and the budget specified for the testing process (Aboutrab et al., 2010). Prioritising PA test sets to be deployed for a certain application in a particular organisation is thus a complex decision-making task facing a tester.

5.4.2 Decision Alternatives

The outcome of the decision framework is to prioritise the available test sets generated by PA. Each set is named and constructed according to the structure of timing constraints. Chapter 3 (Section 3.5.2) gives details about the PA test sets. To summarise, the Boundary set (B) contains test cases that achieve transition coverage by considering the boundary values of timing constraints defined for each transition it covers. The Out-Boundary set (OB) contains test cases that achieve transition coverage by considering the out-boundary values of timing constraints defined for each transition it covers. The In-Boundary set (IB) contains test cases that achieve transition coverage by considering the in-boundary values of timing constraints defined for each transition it covers. The B+OB set combines the Boundary set (B) and Out-Boundary set (OB). The B+IB set combines the Boundary set (B) and In-Boundary set (IB). The OB+IB set combines the Out-Boundary set (OB) and In-Boundary set (IB). Finally, the B+OB+IB set combines the Boundary set (B), Out-Boundary set (OB) and In-Boundary set (IB).

5.4.3 Decision Criteria

The factors and requirements affecting the decision process are classified into three criteria: the test adequacy, test cost and application domain. Each criterion is sub-categorised according to different sub-criteria.

5.4.3.1 Test Adequacy

Adequacy criteria are often used to rank the quality of a proposed test suite. Different types of adequacy criteria are discussed and used in the literature such as structural or fault coverage. The test adequacy considered in our decision model includes both fault and structural coverage (i.e., CRC), which have been discussed in previous chapters. The following present a concise summary of their concepts.

1. *Fault Coverage (FC)*: Identifying how many faults can be detected by a test suite is known as fault coverage. It is always desirable to seek tests capable of detecting most potential faults in an SUT. Accordingly, fault coverage is one of the parameters essential in determining the power of produced test cases in detecting faults in an implementation and hence plays an important role in the decision-making process. The measurement of fault coverage is calculated by the application of Mutation Analysis Technique (MAT) (Lipton, 1971). MAT involves injecting well-defined faults into the SUT to identify the fault detection capability of a test suite. A set of timed as well as functional mutation operators were proposed in Chapter 4 (Section 4.5.2) to represent the possible faults that might be encountered. PA test sets can be thus prioritised according to their FC scores calculated by Equation (5.4).

$$FC = \frac{\text{number of mutants Killed by a test set}}{\text{total number of generated mutants} - \text{number of equivalent mutants}} \quad (5.4)$$

2. *Clock Region Coverage (CRC)*: The aim of structural coverage is to measure to what extent test cases cover the specification model. Since any proposed fault model cannot guarantee to specify all faults, the use of structural coverage cannot be ignored. Chapter 3 (Section 3.4) introduced CRC as timed coverage criterion to select tests that are able to cover timing behaviour of an SUT. The CRC as a transition-based term is calculated for each test set with respect to each transition in the specification model according to Equation (5.5). To calculate the CRC for a test set with respect to the entire specification model, the transition-based CRC values calculated for all transitions within the model are averaged.

$$CRC = \frac{\text{number of clock regions covered by a chosen set with respect to a transition}}{\text{total number of clock regions of the transition in a specification model}} \quad (5.5)$$

5.4.3.2 Test Cost

Testing in general suffers from a high cost of test generation and executing process. Usually, the test cost can be determined by two factors, namely test length and test execution time.

1. *Test Traces Length (TTL)*: One of the most relevant factors affecting the test cost is the number or the length of test cases. It is desirable to find small test suites that detect many faults. As a result, the TTL is an essential factor in the decision-making framework. The length of test cases in each test set used for our AHP model is calculated according to Equation (4.3) mentioned in Chapter 4.
2. *Test Execution Time (TET)*: Test execution time determines how fast an SUT performs under a particular test set. Since the tester will always prefer a test set that needs the least time to execute and therefore least cost, calculating each set execution time for a particular SUT is important for making the right decision. PA was automated by the GeTeX tool providing a complete automation process for generating and executing real-time test-cases on the SUT. As a result, the execution time for each test set is measured by GeTeX.

5.4.3.3 Application Domain

The testing prioritisation process should take the application domain into account. In our decision model, we consider three different sub-criteria.

1. *Importance*: The more important the application, the more thorough testing it needs. For instance, a user might be slightly irritated if a coffee cup is delivered from a coffee machine in a longer time frame than expected. However, a user life could be under threat if a safety critical system shows faulty behaviour. It is thus essential to compare the test sets to find a more suitable one for more important applications.

2. *Complexity*: Some projects are simple, such as a light controller whereas some are more complex like air traffic control. Complexity is related to the technologies used, the number of lines of code or coupling between the classes or routines. As a result, it is essential to consider the application complexity in determining the most appropriate test set for more complex applications.
3. *Development Stage*: A project could be at different stages when a testing project starts. An early stage can be when only a general idea and a specification model exist, whereas a mature stage can be when the application is almost ready. The project development stage should affect the tester's choice as to which test set can be generated and implemented that mostly suit early-stage applications.

5.4.4 Data Collection

In order to rank the decision alternatives according to the criteria and sub-criteria and thus form the pair-wise comparison matrices, two methods were followed: a) heuristics by running the production-cell test bed and b) interviews.

5.4.4.1 Production-Cell Test Bed

Some decision sub-criteria (FC, CRC, TTL and TET) are quantifiable factors which cannot be ranked subjectively by humans without real data. As a result, executing the PA test sets on real-time systems is essential to enable collection of the data required for pair-wise comparing the PA test sets in terms of CRC, FC, TTL and TET. As a result, the production-cell test bed was used to collect the required data. To construct the pair-wise comparison matrices that rank the preference of the PA test sets according to FC, CRC, TTL and TET, PA test sets were generated and executed for each component of the production-cell (i.e., robot-in, robot-out, control panel and conveyor) using GeTeX.

Firstly, to pair-wise compare the PA test sets with respect to the FC sub-criterion, we produced all possible mutants by manually mutating the C code of each component of the production-cell according to the proposed mutation operators (Section 4.5.2). The test sets of PA were then executed against each mutant. A mutant is considered killed if the injected fault is detected by a test set. For each test set, we calculated the number of generated and killed mutants to obtain fault coverage (FC) for the control panel, conveyor, robot-in and robot-out, respectively according to Equation (5.4). The number of equivalent mutants has no effect on the data since they are the same for all test sets.

To calculate the final value of the FC for each test set, we averaged the FC values obtained for all production-cell components. The pair-wise comparison matrix of the PA test sets with respect to the FC was then constructed by transforming the obtained FC values of each test set according to the nine-point scale as depicted in Table 5.4. The comparison matrix implies that ‘B+OB+IB’ and ‘OB+IB’ sets are the most preferable sets in terms of fault detection capability.

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	3.04	0.71	0.39	0.71	0.34	0.34
OB	0.33	1	0.23	0.13	0.23	0.11	0.11
IB	1.41	4.3	1	0.55	1	0.48	0.48
B+OB	2.55	7.76	1.8	1	1.8	0.86	0.86
B+IB	1.41	4.3	1	0.55	1	0.48	0.48
OB+IB	2.96	9	2.09	1.16	2.09	1	1
B+OB+IB	2.96	9	2.09	1.16	2.09	1	1

Table 5.4: Pair-wise comparison matrix of alternatives with respect to FC

Secondly, to pair-wise compare the test sets with respect to the CRC sub-criterion, we averaged the CRC values calculated for each test set according to the robot-in, robot-out, control panel and conveyor using Equation (5.5). The pair-wise comparison matrix of the PA test sets with respect to the CRC was then constructed by transforming the obtained CRC values of each test set according to the nine-point scale as depicted in Table 5.5.

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1.73	0.23	0.82	0.2	0.22	0.19
OB	0.58	1	0.13	0.47	0.12	0.13	0.11
IB	4.34	7.52	1	3.56	0.87	0.95	0.84
B+OB	1.22	2.11	0.28	1	0.24	0.27	0.23
B+IB	4.98	8.63	1.15	4.09	1	1.09	0.96
OB+IB	4.55	7.89	1.05	3.73	0.91	1	0.88
B+OB+IB	5.2	9	1.2	4.26	1.04	1.14	1

Table 5.5: Pair-wise comparison matrix of alternatives with respect to CRC

The comparison matrix implies that ‘B+OB+IB’ set is the most preferable set in terms of covering most of the clock regions.

Thirdly, to pair-wise compare the test sets with respect to the TTL sub-criterion, TTL of each test set according to the robot-in, robot-out, control panel and conveyor was calculated using Equation (4.3). The TTL values for all production-cell components were averaged and transformed to nine-point scale to construct the pair-wise comparison of the PA test sets with respect to the TTL sub-criteria as depicted in Table 5.6. The comparison matrix shows that ‘B’ set is the most preferable set in terms of the TTL as it generates the least number of test cases and hence the least length.

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1.03	3.02	1.15	4.2	4.89	9
OB	0.97	1	2.92	1.11	4.06	4.73	8.7
IB	0.33	0.34	1	0.38	1.39	1.62	2.98
B+OB	0.87	0.9	2.63	1	3.67	4.27	7.86
B+IB	0.24	0.25	0.72	0.27	1	1.16	2.14
OB+IB	0.2	0.21	0.62	0.23	0.86	1	1.84
B+OB+IB	0.11	0.11	0.34	0.13	0.47	0.54	1

Table 5.6: Pair-wise comparison matrix of alternatives with respect to TTL

Lastly, to pair-wise compare the test sets with respect to the TET sub-criterion, the test cases of each test set were executed on a particular production-cell component and its execution times measured in seconds. The final TET of each test set was calculated by averaging the TET values calculated for all production-cell components. The TET values were then transformed to match the nine-point scale

to construct the corresponding pair-wise comparison matrix as depicted in Table 5.7. The comparison matrix shows that ‘B’ set is the most preferable set in terms of the TET as it has the shortest execution time compared with other sets.

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1.07	3.37	1.15	4.19	5.9	9
OB	0.93	1	3.13	1.07	3.9	5.49	8.38
IB	0.3	0.32	1	0.34	1.24	1.75	2.67
B+OB	0.87	0.94	2.94	1	3.65	5.15	7.85
B+IB	0.24	0.26	0.8	0.27	1	1.41	2.15
OB+IB	0.17	0.18	0.57	0.19	0.71	1	1.52
B+OB+IB	0.11	0.12	0.37	0.13	0.47	0.66	1

Table 5.7: Pair-wise comparison matrix of alternatives with respect to TET

5.4.4.2 Testing expert Interviews

Other decision sub-criteria (‘complexity’, ‘importance’ and ‘development stage’) are qualitative factors that can be subject to testers’ preferences and experiences. Interviews are considered the most valuable method in collecting data qualitatively (Denzin and Lincoln, 1998; Yin, 1994). Interviews can collect and interpret participants’ views, thoughts, ambitions and preferences about certain actions or events (Walsham, 1995). As a result, interviews were used in this study to pair-wise compare the preferences of the PA test sets with respect to the decision sub-criteria (‘complexity’, ‘importance’ and ‘development stage’). A panel of five testing experts (E1...E5) from the Department of Information Systems and Computing in Brunel University was chosen for the interviews. The selected experts had experience in managing industrial testing projects and/or an academic testing background. The interviewees were first given sufficient information including examples about how PA works. Three comparisons matrices for ranking the preference of the seven test sets with respect to ‘complexity’, ‘importance’ and ‘development stage’ were then structured and given to each expert. The verbal preferences were interpreted by those experts into numbers according to the nine-point scale. The interview sheet can be found in Appendix C.

Five comparison matrices for each ‘importance’, ‘complexity’ and ‘development stage’ sub-criteria were ranked by five experts. The expert ratings were similar and acceptable as the Consistency Ratio (CR) calculated for each produced comparison matrix was less than 10%. Due to space limitations and to avoid repetitions, several representative matrices only are shown. Please refer to Appendix D for a complete set of tables. Table 5.8 depicts the pair-wise comparison matrix of alternatives with respect to the ‘importance’ sub-criterion as a result of interviewing the testing expert E1. E1 believed that a complete test set (B+OB+IB) was the most preferable choice for testing more important applications as it is (9, 9, 9, 7, 7, 7) times preferable than the test sets (B, OB, IB, B+OB, B+IB, OB+IB), respectively. The CR calculated for this matrix according to Equation (5.2) was 5.8%. In other words, the ranks of interviewee E1 were acceptable since the CR was less than 10%.

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1	1	0.5	0.25	0.2	0.11
OB	1	1	1	0.5	0.25	0.2	0.11
IB	1	1	1	0.5	0.25	0.2	0.11
B+OB	2	2	2	1	0.33	0.25	0.14
B+IB	4	4	4	3	1	0.25	0.14
OB+IB	5	5	5	4	4	1	0.14
B+OB+IB	9	9	9	7	7	7	1

Table 5.8: Pair-wise comparison matrix of alternatives with respect to the ‘importance’ (E1)

Table 5.9 depicts the pair-wise comparison matrix of alternatives with respect to the ‘complexity’ sub-criterion as a result of interviewing testing expert E4. Choosing the ‘B+IB’ set is seven times preferable than ‘OB’ set for testing more complex applications according to E4. The CR calculated for this matrix was 3.68%, which implies that the E4 ranking was acceptable according to Saaty.

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	4	2	0.33	0.25	0.5	0.17
OB	0.25	1	0.33	0.17	0.14	0.2	0.11
IB	0.5	3	1	0.25	0.2	0.33	0.14
B+OB	3	6	4	1	0.5	2	0.25
B+IB	4	7	5	2	1	3	0.33
OB+IB	2	5	3	0.5	0.33	1	0.2
B+OB+IB	6	9	7	4	3	5	1

Table 5.9: Pair-wise comparison matrix of alternatives with respect to the ‘complexity’ (E4)

5.4.5 Raised Power Matrices

Obtaining all matrices that pair-wise compare the PA test sets according to all decision sub-criteria either by the test bed or the interviews, all obtained matrices were raised to a larger power to improve its accuracy according to (Saaty, 2008). Table 5.10 shows a matrix derived from the comparison matrix in Table 5.8 by squaring it twice.

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	411.06	411.06	411.06	254.2	149.31	87.49	31.89
OB	411.06	411.06	411.06	254.2	149.31	87.49	31.89
IB	411.06	411.06	411.06	254.2	149.31	87.49	31.89
B+OB	677.63	677.63	677.63	419.97	248.12	144.9	52.44
B+IB	1299.12	1299.12	1299.12	806.42	481.18	281.68	100.73
OB+IB	2301.51	2301.51	2301.51	1420.86	846.77	504.47	180.95
B+OB+IB	6362.09	6362.09	6362.09	3905.71	2286.46	1369.36	502.04

Table 5.10: Squared matrix of alternatives with respect to the ‘importance’ (E1)

5.4.6 Normalised Matrix and Eigenvector

All obtained raised power comparison matrices were then normalised to calculate their eigenvectors. A representative normalised matrix of the matrix in Table 5.10 and its eigenvector are depicted in Table 5.11. To normalise a matrix, each of its

elements was divided by the sum of its columns. For instance, the normalised value (0.03) in the cell (row: B, column: B) in Table 5.11 was obtained by dividing the value of the same cell (411.06) in the squared matrix (Table 5.10) by the sum of values of its column (11873.53). The eigenvector of the test sets can then be calculated by dividing the sum of each row of the normalised matrix by the number of its elements (i.e., calculating the average of each row values).

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigenvector
B	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
OB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
IB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
B+OB	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
B+IB	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
OB+IB	0.19	0.19	0.19	0.19	0.2	0.2	0.19	0.195
B+OB+IB	0.54	0.54	0.54	0.53	0.53	0.53	0.54	0.535

Table 5.11: Normalised matrix and eigenvector of alternatives with respect to the ‘importance’ (E1)

According to each expert, the eigenvector of alternatives (i.e., test sets) was calculated in order to transform a) the relative weights of alternatives with respect to each decision sub-criterion to b) absolute weights. As a result, we obtained five alternative eigenvectors (i.e., ranks) from five interviewees. The eigenvectors of the test sets with respect to the sub-criteria (FC, CRC, TTL and TET) were the same since their comparison matrices were constructed once using the test bed. On the other hand, the eigenvectors of the test sets with respect to sub-criteria (‘importance’, ‘complexity’ and ‘development stage’) were different since their comparison matrices were constructed five times according to the five experts. Using the geometric mean approach, the five ranking tables were integrated into one final table showing the weight of each test set according to each decision sub-criterion (Table 5.12).

Criteria	Sub Criteria	Test Sets						
		B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
Test Adeq.	FC	0.0793	0.0260	0.1119	0.202	0.1119	0.2344	0.2344
	CRC	0.0457	0.0264	0.1985	0.056	0.2279	0.2082	0.2376
Test Cost	TTL	0.2685	0.2595	0.0890	0.234	0.0639	0.0549	0.0298
	TET	0.2763	0.2571	0.0821	0.241	0.066	0.0468	0.0307
App. Domain	Imp.	0.0365	0.0588	0.0996	0.0819	0.2079	0.1781	0.3372
	Comp.	0.0365	0.0588	0.0996	0.0819	0.2079	0.1781	0.3372
	D.Stage	0.0365	0.0588	0.0996	0.0819	0.2079	0.1781	0.3372

Table 5.12: Integrated ranking of alternatives with respect to all sub-criteria (geometric mean)

The calculated weights of each test set as shown in Table 5.12 are independent from a testing project. In other words, these weights are the same for all testing projects and necessary for the next stage of the decision-making process for a particular testing project.

5.5 Testing scenarios

In this section, the use of the proposed AHP framework in two common real-time testing scenarios for validating the approach is described. Applying the proposed AHP framework on a particular testing project can assist the tester in choosing the best suited PA test set for it. Having the absolute weights (i.e., ranks) of the test sets with respect to all decision sub-criteria, the tester (decision-maker) has to pair-wise compare the preference of one decision criterion to another. Within each criterion, the sub-criteria also need to be pair-compared; this is to obtain the absolute weights (i.e., eigenvectors) for each decision criterion and sub-criterion with respect to the particular scenario using the same steps as previously mentioned. Calculating the weights of decision criteria and sub-criteria is dependent on the testing scenario. As a result, the tester should repeat the calculation of the decision criteria and sub-criteria weights for each testing project or scenario. To reduce the time the AHP calculations might take, the AHP process was automated using a tool (Alrouh, 2011). Obtaining the weights for decision alternatives, criteria and sub-criteria, we are able to obtain the decision outcomes.

5.5.1 Scenario 1: Control System

This section gives an overview of a control system that was used as a testing scenario. The application of the proposed AHP framework in prioritising the PA test sets to suit the testing scenario is presented next.

5.5.1.1 Scenario 1 Description

A software company is assigned to develop a real-time system to control and monitor the temperature of freezer rooms in an industrial plant. The controller deals with several inputs such as room air temperature and a defrost temperature. It delivers outputs controlling several relays, a display unit showing room air temperature and LEDs indicating for any alarm or error. The compressor must remain on for minimum time duration and can restart after certain time as well. An alarm sounds if the temperature increases above a specified limit. Timing constraints within the specification are in the range of minutes.

The testing activities start at a late stage of the system development. The budget is limited and it is required to deliver the system without any latency. The system at delivery should match all the requirements without any major deficiencies.

5.5.1.2 AHP Application on Scenario 1

To obtain the weights of decision criteria and sub-criteria, their comparison matrices were constructed and given to the experts. In a real application, the comparison matrices should be constructed by a tester (who tests the application). We chose the experts to construct the comparison matrices to assess the validity of the AHP framework. Due to space limitations, we randomly picked a representative matrix for a decision criterion and sub-criterion; the entire set of matrices can be found in Appendix D. For instance, Table 5.13 presents the pairwise comparison matrix of the main criteria with respect to the decision goal according to E2. E2 assumed that the ‘test cost’ should have the highest priority

(i.e., 4 and 6 times preferable than ‘test adequacy’ and ‘application domain’ respectively) since the budget is limited. ‘Test adequacy’ is marginally more important (3 times preferable) than ‘application domain’ since the company has to deliver the system without any major deficiencies. The CR calculated for this matrix was 4.76%, implying that the expert ranking is acceptable (less than 10%). The matrix was then raised to a higher power and normalised to calculate the eigenvector (weights).

	Test Adequacy	Test Cost	Application Domain	Weights
Test Adequacy	1	0.25	3	0.2176
Test Cost	4	1	6	0.6909
Application Domain	0.33	0.17	1	0.0915

Table 5.13: Pair-wise comparison matrix and eigenvector of the main criteria with respect to the decision goal (E2, Scenario 1)

Three pair-wise comparison metrics for comparing the decision sub-criteria with respect to the criteria they belong to were constructed according to each expert. First, FC and CRC sub-criteria were compared with respect to the ‘test adequacy’ criterion. Table 5.14 depicts the pair-wise comparison matrix of the sub-criteria with respect to the ‘test adequacy’ criterion according to E2. Since the company intends to deliver the application without any major deficiencies, E2 assumed that CRC was 5 times important than FC as CRC covers most of the application system. The consistency ratio for this matrix was 0 since it has only two dimensions. The matrix was then raised to a higher power and normalised to calculate the eigenvector (local weights).

	FC	CRC	Local Weights
FC	1	0.2	0.1667
CRC	5	1	0.8333

Table 5.14: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘test adequacy’ (E2, Scenario 1)

The term ‘local weights’ is used to refer to the weight of each sub-criterion with respect to its decision criterion, but without taking into account the criteria weights themselves.

Second, TTL and TET sub-criteria were pair-wise compared with respect to the ‘test cost’ criterion. Table 5.15 depicts the pair-wise comparison matrix of the sub-criteria with respect to the ‘test cost’ criterion according to E2. Since the company intends to deliver the system very soon without any latency, E2 assumed that TET is three times more important than TTL. The consistency ratio for this matrix was 0 since it has only two dimensions. The matrix was then raised to a higher power and normalised to calculate the eigenvector (local weights).

	TTL	TET	Local Weights
TTL	1	0.333	0.2499
TET	3	1	0.7501

Table 5.15: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘test cost’ (E2, Scenario 1)

Third, ‘importance’, ‘complexity’ and ‘development stage’ sub-criteria were pair-wise compared according to the ‘application domain’ criterion. Table 5.16 depicts the pair-wise comparison matrix of the sub-criteria with respect to the ‘application domain’ criterion according to E2.

	Importance	Complexity	Development Stage	Local Weights
Importance	1	1	4	0.4231
Complexity	1	1	6	0.4844
Development Stage	0.25	0.17	1	0.0925

Table 5.16: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘application domain’ (E2, Scenario 1)

Since the SUT is a control system with many parameters and connections involved in, E2 assumed that the tester should pay particular attention to the SUT ‘complexity’ (i.e., 6 times more preferable than ‘development stage’). Application ‘importance’ is also more preferable than ‘development stage’. The CR calculated

for this matrix was 1.63% implying that the expert ranking is acceptable (less than 10%). The matrix was then raised to a higher power and normalised to calculate the eigenvector (local weights).

To obtain the global weight of each sub-criterion, the local weight of a sub-criterion was multiplied by the weight of its criterion (Section 5.3.2, step 6). The calculation of sub-criteria local and global weights was repeated for each expert and then integrated using the geometric mean approach (Table 5.17).

Criteria	Weight	Sub-Criteria	Local Weight	Global Weight
Test Adequacy	0.271090	FC	0.4637	0.1257
		CRC	0.3309	0.0897
Test Cost	0.398287	TTL	0.2264	0.0902
		TET	0.7195	0.2866
Application Domain	0.154559	Importance	0.2918	0.0451
		Complexity	0.2353	0.0364
		Development Stage	0.2652	0.0410

Table 5.17: Integrated local and global weights for Scenario 1 (geometric mean)

After having the generic alternative weights (Table 5.12) and scenario-based sub-criteria global weights (Table 5.17), the final ranking results were synthesized by multiplying each alternative weight by the global weight of its sub-criterion. For instance, the weight of the test set ‘B’ according to the FC sub-criterion is ‘0.0793’ as in Table 5.12. The global weight of the FC is ‘0.1257’. As a result, the final weight of the set B with respect to the FC considering Scenario 1 is $(0.0793 \times 0.1257 = 0.01)$. The resulting weights were added for each alternative to obtain its final priority as in shown in Table 5.18.

Criteria	Sub Criteria	Test Sets						
		B	OB	IB	B+OB	B+IB	OB+I B	B+OB +IB
Test Adeq.	FC	0.0100	0.0033	0.0141	0.0254	0.0141	0.0295	0.0295
	CRC	0.0041	0.0024	0.0178	0.0050	0.0204	0.0187	0.0213
Test Cost	TTL	0.0242	0.0234	0.0080	0.0211	0.0058	0.0050	0.0027
	TET	0.0792	0.0737	0.0235	0.0691	0.0189	0.0134	0.0088
App. Domain	Import.	0.0020	0.0017	0.0018	0.0057	0.0080	0.0072	0.0164
	Comp.	0.0019	0.0018	0.0024	0.0032	0.0053	0.0051	0.0091
	D. Stage	0.0020	0.0018	0.0026	0.0042	0.0065	0.0060	0.0138
Total Priority		0.1234	0.1080	0.0702	0.1336	0.0790	0.0847	0.1015
Ranking		2	3	7	1	6	5	4

Table 5.18: Final ranking results (Scenario 1)

The use of the AHP framework (Table 5.18) suggests that the ‘B+OB’ set is the most preferable test set to use in testing the application defined in Scenario 1. Choosing this test set would cover the majority of test project requirements. For instance, this test set combines the fault detectability power of ‘B’ and ‘OB’ sets where it can be executed in small time as it is a relatively small set. In addition, the AHP framework prioritises the possible test sets for a particular testing scenario. In this scenario, ‘B+OB’, ‘B’ and ‘OB’ are at the top of the rankings and add flexibility to the tester’s choice.

Each expert was asked to rank the test sets (1 to 7) according to Scenario 1 where ‘1’ represents the most appropriate and ‘7’ the least appropriate. The expert ranks were then integrated into a final rank taking into account the most frequent rank for each test set. Average was not used to combine the rank values since the ranks are categorical. If two ranks had the same frequency values with respect to a particular test set, the frequency value that was closest to other ranks was chosen. For instance, with respect to the ‘OB’ test set, the frequency value of rank ‘3’ and rank ‘2’ was ‘2’. Since the remaining rank ‘6’ is much closer to rank ‘3’ than rank ‘2’, the rank ‘3’ to represent the ‘OB’ set was chosen. Table 5.19 illustrates the expert ranks, the integrated ranks as well as the AHP ranks with respect to Scenario 1.

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
E1	2	3	6	1	5	7	4
E2	5	2	3	1	7	6	4
E3	5	6	7	1	3	4	2
E4	1	2	6	3	5	7	4
E5	2	3	6	1	5	7	4
Integrated Ranks	2	3	6	1	5	7	4
AHP Ranks	2	3	7	1	6	5	4

Table 5.19: AHP ranking VS experts' ranking outcomes (Scenario 1)

Comparing the experts' integrated ranks with those of the AHP framework, we found that they achieved a high degree of similarity. Table 5.20 shows that Kendall's and Spearman's correlation coefficients were significant at the 1% level which would demonstrate the validity of the AHP framework.

Type	Variables	Expert Ranks	AHP Ranks
Kendall's tau_b	Expert Ranks	1.000	.810 ^{**}
	AHP Ranks	.810 ^{**}	1.000
Spearman's rho	Expert Ranks	1.000	.893 ^{**}
	AHP Rank	.893 ^{**}	1.000
** Correlation is significant at the 0.01 level.			

Table 5.20: Kendall's and Spearman's correlation coefficients between the experts' integrated ranks and AHP ranks (Scenario 1)

5.5.2 Scenario 2: Medical System

This section gives an overview of a medical system that was used as a testing scenario. The application of proposed AHP framework in prioritising the PA test sets to suit the testing scenario is presented next.

5.5.2.1 Scenario 2 Description

A software company is assigned to develop a medical system that provides real-time monitor to the heart rate, blood pressure and blood oxygen. The system

accepts symptoms and makes diagnosis of infections. The project design and development consumed the majority of the time assigned to the project which caused the testing activities to start late. The deadline is approaching, but with a possibility of an extension.

5.5.2.2 AHP Application on Scenario 2

The proposed AHP framework was applied on Scenario 2. Similar to that for Scenario 1, several matrices were constructed and given to the experts to pair-wise compare the preferences of the criteria and sub-criteria. We randomly picked a representative matrix for a decision criterion and sub-criterion to be shown; the remaining matrices can be found in Appendix D. For instance, Table 5.21 presents the pair-wise comparison matrix of the main criteria with respect to the decision goal according to E3. E3 assumed that the ‘test adequacy’ should have the highest priority since the application is safety-critical and should be thoroughly tested. In addition, ‘application domain’ is marginally more important than ‘test cost’ (i.e., 2 times preferable than ‘test cost’). The CR calculated for this matrix was 2.12% implying that the expert ranking is acceptable (less than 10%). The matrix was then raised to a higher power and normalised to calculate the eigenvector (weights).

	Test Adequacy	Test Cost	Application Domain	Weights
Test Adequacy	1	5	3	0.6833
Test Cost	0.2	1	6	0.1169
Application Domain	0.25	2	1	0.1998

Table 5.21: Pair-wise comparison matrix and eigenvector of the main criteria with respect to the decision goal (E3, Scenario 2)

Three pair-wise comparison metrics for comparing the decision sub-criteria with respect to the criteria they belong to were constructed according to each expert. First, FC and CRC sub-criteria were compared with respect to the ‘test adequacy’ criterion. Table 5.22 depicts the pair-wise comparison matrix of the sub-criteria

with respect to the ‘test adequacy’ criterion according to E3. E3 assumed that CRC was equally important to FC, since detecting faults and thoroughly testing the application were both necessary. The consistency ratio for this matrix was 0 since it has only two dimensions. The matrix was then raised to a higher power and normalised to calculate the eigenvector (local weights).

	FC	CRC	Local Weights
FC	1	1	0.5
CRC	1	1	0.5

Table 5.22: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘test adequacy’ (E3, Scenario 2)

Second, Table 5.23 depicts the pair-wise comparison matrix of the sub-criteria with respect to the ‘test cost’ criterion according to E3. E3 assumed that TTL and TET have almost similar effect on the decision process but TTL is slightly more preferable. The consistency ratio for this matrix was ‘0’ since it has only two dimensions. The matrix was then raised and normalised to calculate the local weights.

	TTL	TET	Local Weights
TTL	1	2	0.6667
TET	0.5	1	0.3333

Table 5.23: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘test cost’ (E3, Scenario 2)

Third, Table 5.24 depicts the pair-wise comparison matrix of the sub-criteria with respect to the ‘application domain’ criterion according to E3. E3 assumed that the tester should pay a particular attention to the ‘importance’ criterion since it is a safety-critical application. The application ‘importance’ is thus more preferable than the ‘development stage’. The CR calculated for this matrix was 1.55% implying that the expert ranking is acceptable (less than 10%). The matrix was then raised to a higher power and normalised to calculate the eigenvector (local weights).

	Importance	Complexity	Development Stage	Local Weights
Importance	1	4	3	0.6251
Complexity	0.25	1	0.5	0.1365
Development Stage	0.33	2	1	0.2384

Table 5.24: Pair-wise comparison matrix and eigenvector of the sub-criteria with respect to the ‘application domain’ (E3, Scenario 2)

The global weight of each sub-criterion was obtained by multiplying the local weight of a sub-criterion by the weight of its criterion. The calculation of sub-criteria local and global weights was repeated for each expert and then integrated using the geometric mean approach (Table 5.25).

Criteria	Weight	Sub-Criteria	Local Weight	Global Weight
Test Adequacy	0.423431	FC	0.4637	0.1257
		CRC	0.3309	0.0897
Test Cost	0.160584	TTL	0.2264	0.0902
		TET	0.7195	0.2866
Application Domain	0.240414	Importance	0.2918	0.0451
		Complexity	0.2353	0.0364
		Development Stage	0.2652	0.0410

Table 5.25: Integrated local and global weights for Scenario 2 (geometric mean)

After generating the generic alternative weights (Table 5.12) and scenario-based sub-criteria global weights (Table 5.25), the final ranking results were synthesized by multiplying each alternative weight by the global weight of its sub-criterion. The resulting weights were added for each alternative to obtain its final priority as shown in Table 5.26.

The use of AHP framework suggests that the ‘B+OB+IB’ set is the most preferable test set to use in testing the application defined in Scenario 2 since the system under test is safety-critical. Any fault or missed behaviour can have a disastrous effect on the patient life. The cost can be ignored with respect to the safety in such applications.

Criteria	Sub Criteria	Test Sets						
		B	OB	IB	B+OB	B+IB	OB+I B	B+OB +IB
Test Adeq.	FC	0.0183	0.0060	0.0259	0.0467	0.0259	0.0542	0.0542
	CRC	0.0072	0.0041	0.0311	0.0088	0.0357	0.0326	0.0372
Test Cost	TTL	0.0138	0.0134	0.0046	0.0121	0.0033	0.0028	0.0015
	TET	0.0254	0.0236	0.0075	0.0221	0.0061	0.0043	0.0028
App. Domain	Import.	0.0033	0.0028	0.0030	0.0095	0.0133	0.0119	0.0273
	Comp.	0.0032	0.0029	0.0040	0.0053	0.0086	0.0083	0.0149
	D. Stage	0.0024	0.0022	0.0032	0.0051	0.0080	0.0073	0.0169
Total Priority		0.0736	0.0550	0.0792	0.1095	0.1009	0.1216	0.1549
Ranking		6	7	5	3	4	2	1

Table 5.26: Final ranking results (Scenario 2)

Similar to Scenario 1, each expert was asked to rank the test sets (1 to 7) for Scenario 2. The expert ranks were then integrated into a final rank taking into account the most frequent rank occurring for each test set. Table 5.27 illustrates the expert ranks, the integrated ranks as well as the AHP ranks with respect to Scenario 2.

Test Sets	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
E1	7	6	5	4	3	2	1
E2	5	7	6	4	3	2	1
E3	5	6	7	2	3	4	1
E4	5	7	6	3	2	4	1
E5	5	7	6	4	3	2	1
Integrated Ranks	5	7	6	4	3	2	1
AHP Ranks	6	7	5	3	4	2	1

Table 5.27: AHP ranking VS experts' ranking outcomes (Scenario 2)

Comparing the experts' integrated ranks with those of the AHP framework, we found that they again achieved a high degree of match. Table 5.28 shows that Kendall's and Spearman's correlation coefficients were significant at the 1% level which would again demonstrate the applicability of the AHP framework with different testing scenarios.

Type	Variables	Expert Ranks	AHP Ranks
Kendall's tau_b	Expert Ranks	1.000	.810 ^{**}
	AHP Ranks	.810 ^{**}	1.000
Spearman's rho	Expert Ranks	1.000	.929 ^{**}
	AHP Rank	.929 ^{**}	1.000
** Correlation is significant at the 0.01 level.			

Table 5.28: Kendall’s and Spearman’s correlation coefficients between the experts’ integrated ranks and AHP ranks (Scenario 2)

5.6 Summary

PA is a real-time test generation method that generates three different test sets. A systematic decision-making framework might help an organisation to choose the best suited test set to be deployed for a certain application. This chapter presented a novel Analytical Hierarchy Process (AHP) as decision-making framework which provides testers with a systematic and manageable approach through which they can prioritise the available testing sets that best fulfil their testing requirements. The development of the AHP framework was based on the data collected by the production-cell test bed and interviews with a group of testing experts. Since this study can be considered the first which applies AHP in TA-based testing, the results cannot be validated through a comparison study. As a result, the AHP framework was validated using two different scenarios highlighting different real-time systems under test with different testing requirements. The framework decision outcomes match to a high degree with the expert panel ranking outcomes demonstrating that the AHP framework is sound and valid. The framework is also supported by a tool to automate all the calculations required. As a result, the tester needs only the pair-wise comparison matrices for decision criteria and sub-criteria and the tool will give the final ranking immediately.

Chapter 6: Conclusions

6.1 Topic Overview

Real-time Embedded Systems (RTESs) have an increasing role in controlling the IT that we use on a day-to-day basis. RTES behaviour is not based solely on the interactions it might have with its surrounding environment, but also on timing requirements it induces. As a result, ensuring that an RTES behaves correctly is non-trivial, especially after adding time as a new dimension to the complexity of the testing process. Testing an RTES implementation to ensure that it is as fault-free as possible before its deployment is therefore important. Model-Based Testing (MBT), based on comparing SUT behaviour with a reference specification model aims to minimise cost through early capture of system behaviour and the automation of test case generation, execution and evaluation. A Timed Automata (TA) formalism is one of the most frequently used language to model RTESs due to its ability to express its real-time behaviour. Testing from TA has received increased attention in recent research where several TA-based testing algorithms have been proposed. However, the effort expended, the number of test cases generated and the test adequacy criteria that the testing approaches are based on are still questionable, especially in the absence of empirical validation based on defined assessment criteria. As a result, it is important to develop a valid and flexible approach that can handle these issues.

In this chapter, Section 6.2 summarises the research findings of each chapter. Section 6.3 explains how the research objectives are achieved. A summary of the Thesis contributions is then presented in Section 6.4. Finally, Section 6.5 identifies the research limitations and points to future research ideas.

6.2 Research Summary

The aim of the research presented in this Thesis was to develop, validate and automate a flexible TA-based testing approach based on a timed selection criterion for testing real-time embedded systems.

Chapter 1 gave an overview of the area under research and highlighted the motivation of this research. That emphasised the need for developing a valid TA testing approach capable of testing RTEs based on a timed adequacy criterion. A set of research objectives were identified to fulfil the research aim.

Chapter 2 reviewed the related literature that addressed testing RTEs. The concept of testing was defined and explained by addressing some topics related to the selection criteria, testing types and the combination of formal methods. To test RTEs, the formal language to be used for building the specification models should be capable of capturing continuous as well as discrete behaviour of the SUT. As a result, TA has been adopted for testing RTEs. Several studies were reviewed in this chapter. The majority were based on un-timed selection criteria for generating timed test cases. In addition, only a few have been supported by tools and empirically studied.

Chapter 3 set the rules and mathematical equations of adopting the clock region concept as a timed adequacy criterion for selecting test cases. Clock region coverage was the basis for proposing PA as a new component-based offline test case generation method for RTEs modelled as UTA. PA was based on dividing the generated test cases into three sets of priorities (boundary, out-boundary, in-boundary) to enhance the flexibility of the approach by allowing the tester to choose the appropriate set according to the testing environment. To validate PA, the chapter proposed a set of timed and functional mutation operators to enable the use of SMA in TA context. The validation was based on comparing the mutation score achieved by PA on three TA case studies with four other timed testing approaches based on TA. Combining the mutation scores achieved by PA through the used case studies, we showed that our PA out-performed other

approaches by achieving a higher score with relatively few generated tests. The validation also revealed some interesting results especially for validating other approaches. For instance, the SCT failed to detect all state transfer faults in spite of the state identification technique equipped used. COVER failed to detect all output or input faults in spite of the coverage criterion it follows.

Chapter 4 automated the generation and the execution of test cases according to PA and *tioco* theory by developing the GeTeX tool. GeTeX is an offline tool that targets testing timing behaviour of RTEs according to a timed selection criterion. The validity of GeTeX was empirically demonstrated by a light controller prototype. The tool generated and executed the test cases in a short time without any compilation errors. To execute the PA tests, this chapter introduced an empirical test bed using a production-cell case study and assessment criteria to validate the PA testing approach compared with two TA-based testing approaches (SM and BCT). The testing approaches were assessed and compared based on the timed structural adequacy, fault adequacy, test length and a factor that combined them all. Structural coverage was based on CRC calculated using a proposed equation. FC was measured by calculating the mutation score of each approach according to MAT. To enable this, a set of timed and functional mutation operators on the implementation level was presented. An assessment factor (AF) which considered fault coverage and clock coverage with respect the length of generated test cases was also presented. The experiments confirmed the superiority of PA over the other tested approaches. The overall assessment factor showed that structural and fault coverage scores of PA with respect to the length of its tests were better than those of SM and BCT. Finally, problems encountered during conducting the empirical study were highlighted to direct future experiments.

Chapter 5 highlighted the necessity of a formal decision-making approach for prioritising the PA test sets to be deployed for a certain application. The chapter then developed a multi-criteria decision-making framework based on the Analytical Hierarchy Process (AHP). The development of the AHP framework

was based on the data collected by the production-cell test bed and interviews with a group of testing experts. The AHP framework was validated using two different testing scenarios addressing different real-time systems with different testing requirements. The framework decision outcomes supported with an automation tool showed promising results. The decision outcomes of the AHP framework were significantly correlated to those of testing experts which demonstrated the soundness and validity of the framework. Tool support increased the applicability of the AHP framework in which a tester needs only the pair-wise comparison matrices for decision criteria and sub-criteria. The decision outcomes could be then obtained directly.

6.3 Meeting the Research Objectives

The main aim of the Thesis was to provide software engineering community with a sound, valid and flexible testing approach for testing RTEs considering its environment. This section shows how this research successfully achieved its objectives.

Objective 1: *‘To introduce a timed adequacy criterion for selecting timed test cases’*. The first objective was achieved in Chapter 3 by adopting CRC as a timed adequacy criterion. The proposal of CRC was supported by all necessary equations and rules.

Objective 2: *‘To develop a timed testing approach based on the TA formalism and the proposed timed selection criterion for generating test cases divided into different test sets’*. This objective was achieved in Chapter 3 by developing PA as a TA-based testing approach. PA was based on dividing the generated test cases into three sets of priorities (boundary, out-boundary, in-boundary) to enhance the flexibility of the approach by allowing the tester to choose the appropriate set according to the testing environment.

Objective 3: *‘To develop a tool for automating the generation and execution of timed test cases’*. This objective was achieved in Chapter 4 by developing and

validating the GeTeX tool that deploys PA and the *tioco* conformance relation. GeTeX is an offline tool that targets testing timing behaviour of RTEs. The validity of GeTeX was empirically shown by a light controller prototype.

Objective 4: *‘To evaluate the proposed timed testing approach at the specification and implementation level compared with a set of similar testing approaches based on proposed assessment criteria’.* This objective was achieved in Chapter 3 by validating PA in comparison with four TA-based approaches in terms of fault coverage. To enable this, TA-based mutation operators were proposed. This objective was also met in Chapter 4 by executing the generated tests from three TA-based testing approaches including ours on an industrial-strength test bed. Test assessment criteria were introduced to be able to compare the performance of the testing approaches under study.

Objective 5: *‘To develop and validate a decision-making framework for the proposed timed testing approach to formalise the selection of the best test set suiting a testing project’.* This objective was achieved in Chapter 5 by developing and validating the AHP framework to enable the tester prioritising the available PA test sets. The AHP framework was validated using two different testing scenarios addressing different real-time systems with different testing requirements.

6.4 Summary of Research Contributions

The main research contributions are summarised in the following subsections.

6.4.1 Timed Adequacy Criterion (CRC)

Several testing approaches have been proposed for testing real-time systems from TA specifications. However, the tests were generated based on either a random selection or un-timed coverage criterion to avoid covering the entire infinite continuous SUT behaviour. Other research abstracted the continuous behaviour (time) by converting the timed specification to an un-timed one. Timing behaviour

of an SUT will not be accordingly covered. It is thus essential to consider a timed coverage criterion for testing real-time systems. The lack of a mature timed adequacy criterion directed our research to adopt one.

The concept of clock region was proposed to replace the infinite timed state space by a finite region automaton. As a result, we adopted the clock region as a timed adequacy criterion by setting rules and mathematical equations. Feasible clock regions were generated for each transition within the specification model considering its clock guards, invariants and type (i.e., input or output). The generated test suite should cover all clock regions.

The proposal of clock region as a timed adequacy criterion differs from other works that used the clock region concept as timed abstraction technique in several ways. First, the Region Automaton (RA) in the literature was created at the model level where infeasible regions were not identified. The number of clock regions calculated was enormous for a small model. Second, GA, the source of generating test cases, was formed by sampling the RA at a fixed rate. This leads to the selection of more than one clock value (i.e., time delay) to represent each clock region. As a result, the number of generated test cases was very large. In this study, a set of rules was proposed to create the smallest set of feasible clock regions and to enhance the use of clock regions as a timed adequacy criterion by which test cases can be selected.

6.4.2 Priority-based TA-based Testing Approach (PA)

PA was proposed for generating timed test cases and differs from other proposed TA-based testing approaches in several ways. First, PA is based on a timed selection criterion (CRC). Second, the compact nature of the PA search algorithm enables covering as many transitions as possible in one single test trace. Second, PA takes the testing environment and a tester's opinion into account by dividing the generated test cases into three sets. The test sets are called 'priorities' as the priority of choosing a particular test set or a combination of them is likely to be

different according to the testing environment specified by the criticality of an SUT, the allowable time and the budget specified for the testing process. Each test set (priority) is named and constructed according to the structure of timing constraints.

6.4.3 Specification Mutation Analysis

Any proposed testing approach has to be validated. Assessing a testing approach by measuring its fault coverage is considered one of the widely used methods. Fault coverage needs to be facilitated by a fault model identifying the possible faults that might be encountered. The use of fault coverage as an assessment criterion can be more effective if it is used in a controlled way by the application of Specification Mutation Analysis technique (SMA). Since no study has addressed the application of SMA on TA to our knowledge, we proposed well-suited mutation operators for TA. The proposed TA mutation operators include previously formalised fault models in the literature.

PA was validated in terms of SMA in comparison with four other well-known TA-based testing approaches. This study could be considered the first (to our knowledge) that compared the performance of different approaches. Due to the absence of tool support, test cases of each approach were manually generated from three TA specifications. Comparing fault coverage, PA performed better than others. The study was also able to focus on each approach and point to its pros and cons.

6.4.4 The application of TA-based Approaches on an industrial-strength Test Bed

Some proposed approaches in the literature lack automation tool support. Using such approaches requires a deep understanding of their mechanism and significant manual effort in generating and executing test cases. Others were partially automated. Their tools were responsible for only automating the generation of test

cases. In other words, the execution of test cases generated by such approaches requires other sets of tools.

This research attempted to consider such problems by developing an automating tool for PA called GeTeX. GeTeX automates the process of test cases generation and execution based on the ‘tioco’ conformance theory. In its current version, GeTeX was designed to support CAN applications.

To our knowledge, there has yet to be a study which compares the performance of similar approaches on real applications. This research used a production-cell as an industrial-strength test bed. Well-identified assessment criteria by which the performance of testing approaches can be compared were also presented. In summary, our aim was to identify a testing approach capable of detecting as many faults as possible and covering as many clock regions as possible with minimum length of test cases. The study at the implementation level confirmed results obtained at the specification level. PA outperformed other approaches.

6.4.5 A multi-Criteria Decision Making Framework

PA is a flexible testing approach that enables the tester to choose any set of generated test cases according to the testing environment. According to that choice, PA establishes a trade-off between increasing confidence in SUT correctness and limited testing resources such as time, effort and cost. However, the decision that the tester has to make depends on their intention. Different testers will make different decisions for the same testing environment. A formal decision framework in which all testing requirements and factors (i.e., decision criteria) affecting the testing process are independently categorised, weighted and analysed then becomes viable.

This research developed a decision framework based on AHP. The AHP decision model considered criteria that might affect a tester’s decision in selecting the best PA test set for a particular testing project. The applicability of the framework was viable for two reasons. First, the framework was provided with an automation tool

to speed up the decision-making process and ensure the tester avoided time-consuming calculations. Second, the AHP framework was validated using two testing scenarios. The decision outcomes were compared with those of testing experts. The results showed a significant correlation between the framework outcomes and those of the experts.

6.5 Research Limitations and Future Work

This section identifies a set of research limitations encountered and suggests a set of complementary future work to address them.

6.5.1 The Class of TA Specification Model

One limitation of this study is the use of a restricted class of TA (i.e., deterministic observable model without data) for generating timed test cases. Such a class limits expressiveness and complicates the modelling process. The choice of such a restricted class in this study was so as to prove the applicability of timed selection criterion by isolating other factors that might be encountered. Non-determinism includes internal actions which raise a problem when applying adequacy criteria. It is not known how a non-deterministic SUT would react to an input or which transition is selected by such a reaction. Such a problem has been addressed in the literature by either using online testing or to try and make the specification model deterministic. Moreover, using data in any specification model complicates the process of generating test cases. Combining data with time makes it even more difficult.

Future research trends would be necessary to address this limitation by answering the following question. Can CRC be used with a more general class of TA (i.e., non-deterministic, partially observable with data)? This question could be addressed taking into account the following points. To solve non-determinism, we aim to use other research findings such as those by (Krichen and Tripakis, 2009) in dealing with the problem and then applying CRC. Moreover, to solve the data

problem, we aim to divide the specification model into two - control and data parts. In the control part, we apply our approach proposed here to cover timing behaviour. Other approaches would be used to generate test cases that cover the data aspect. A strategy that can combine test cases from two parts should be accordingly proposed.

6.5.2 Timed Adequacy Criterion

The proposed timed adequacy criterion in this research is based on the concept of clock regions. Using clock regions as a timed adequacy criterion could be criticised due to its relation with the number of clocks and their upper bounds. In other words, in the case of any model using many clocks or clocks with high upper bounds, the number of regions rapidly increases. The research addressed a set of rules to control the rapid growth of the number of regions and avoided using many clocks.

Other partitioning criteria exist in the literature such as zones. The creation of zones is still affected by the number of clocks but only to a certain limit. For future work, it is thus advantageous to study the possibility of using a coarser partitioning relation as a source of timed adequacy criterion. The results should then be compared with those of CRC to determine if the fault detection capability is affected.

6.5.3 Case Studies

This research succeeded in comparing the performance of PA with other testing approaches based on specification case studies and an industrial-strength test bed. However, the relatively small size of case studies used can be considered a limitation. Choosing small specification models for the SMA application was justified due to the manual generation of test cases.

In future work, we aim to use more industrial case studies by which more timing faults can be found and categorised. Moreover, comparing the results of SMA

with those of MAT on the implementation level might guide development of a prediction model that estimates fault coverage of a testing approach at the implementation level by measuring it at the specification level. We plan to consider more recent testing approaches in our future comparison studies.

6.5.4 More insights for the Multi-Criteria Decision Making Approach

The AHP framework is subject to several improvements. We intend to study the possibility of making the AHP model more general by including other decision criteria or sub-criteria. In addition, we plan to increase the accuracy of the AHP framework by increasing the volume of the experimental data as well as the number of experts. Finally, the application of the AHP framework is not restricted to the PA approach. As a result, we intend to apply the AHP framework on various timed model-based testing approaches to choose that most suited one for a testing project.

Testing real-time embedded systems is a promising research topic and much still needs to be done. This study allowed me to learn from both academic and industrial worlds. Meeting academics has influenced my experience and helped me to organise and synthesise my ideas. Moreover, the most interesting part of any research is finding suitable solutions to upcoming problems which are not necessarily related to the research topic. Lastly, but not least, I would actively continue to research in this area, since it is both interesting and important.

References

- Aboutrab, M. S., Alrouh, B., Counsell, S., Hierons, R. and Ghinea, G. (2010) A Multi-criteria Decision Making Framework for Real Time Model-Based Testing. *Testing – Practice and Research Techniques*, London, UK: Springer Berlin / Heidelberg, pp. 194-197.
- Aboutrab, M. S., Alrouh, B., Counsell, S., Hierons, R. and Ghinea, G. (2012a) Prioritising Timed Automata Based Test Sets: A Multi-Criteria Decision Making Approach. *Submitted to Journal of Systems and Software*.
- Aboutrab, M. S., Brockway, M., Counsell, S. and Hierons, R. M. (2012b) Testing Real-time Embedded Systems using Timed Automata Based Approaches. *Journal of Systems and Software (under second review)*.
- Aboutrab, M. S. and Counsell, S. (2010) Fault Coverage Measurement of a Timed Test Case Generation Approach. *17th IEEE International Conference on the Engineering of Computer-Based Systems*, Oxford, UK, pp. 141-149.
- Aboutrab, M. S., Counsell, S. and Hierons, R. M. (2011) GeTeX: A Tool for Testing Real-Time Embedded Systems Using CAN Applications *18th IEEE International Conference on the Engineering of Computer-Based Systems*, Las Vega, USA, pp. 61-70.
- Aboutrab, M. S., Counsell, S. and Hierons, R. M. (2012c) Specification Mutation Analysis for Validating Timed Testing Approaches Based on Timed Automata. *IEEE Signature Conference on Computers, Software, and Applications (COMPSAC 2012)*, Izmir, Turkey (to appear),
- Abran, A., Sellami, A. and Suryan, W. Metrology, measurement and metrics in software engineering. *Ninth International Software Metrics Symposium*, Ecole de Technologie Superieure, Canada pp. 2-11.
- ABRIAL, J.-R. (1996) *The B-Book: Assigning Programs to Meanings*. Cambridge, U.K.: Cambridge University Press.
- Acree, A. T. (1980) *On Mutation*. PhD Thesis, Georgia Institute of Technology.
- Aho, A. V., Dahbura, A. T., Lee, D. and Uyar, M. U. (1991) An optimization technique for protocol conformance test generation based on UIO sequences and rural

- Chinese postman tours. *IEEE Transactions on Communications*, 39 (11), pp. 1604-1615.
- Aichernig, B. K. (2003) Mutation Testing in the Refinement Calculus. *Formal Aspects of Computing*, 15 (2), pp. 280-295.
- Aichernig, B. K., Brandl, H., Jöbstl, E. and Krenn, W. (2010) Model-based mutation testing of hybrid systems. *Proceedings of the 8th international conference on Formal methods for components and objects*, Eindhoven, The Netherlands: Springer-Verlag, pp. 228-249.
- Aichernig, B. K., Brandl, H. and Krenn, W. (2009) Qualitative Action Systems. *Proceedings of the 11th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering*, Rio de Janeiro, Brazil: Springer-Verlag, pp. 206-225.
- Alrouh, B. (2011) *Towards Secure Web Services: Performance Analysis, Decision Making and Steganography Approaches*. PhD thesis, Department of Information Systems and Computing, Brunel University.
- Alur, R. and Dill, D. L. (1994) A theory of timed automata. *Theoretical Computer Science*, 126 (2), pp. 183-235.
- Alur, R., Fix, L. and Henzinger, T. A. (1999) Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science* 211 (1-2), pp. 253-273.
- Ammann, P. and Offutt, J. (2008) *Introduction to Software Testing*. 1st ed. Cambridge: Cambridge University Press.
- Ammann, P. E., Black, P. E. and Majurski, W. (1998) Using Model Checking to Generate Tests from Specifications. *Proceedings of the Second IEEE International Conference on Formal Engineering Methods*, Brisbane, Australia: IEEE Computer Society, pp. 46-54.
- Andrews, J. H., Briand, L. C. and Labiche, Y. (2005) Is mutation an appropriate tool for testing experiments? *Proceedings of the 27th international conference on Software engineering*, St. Louis, MO, USA: ACM, pp. 402-411.
- Ashayeri, J., Keij, R. and Bröker, A. (1998) Global business process re-engineering: a system dynamics-based approach. *International Journal of Operations and Production Management*, 18 (9/10), pp. 817-831.

- Badria, M. A. and Davisb, D. (2001) A comprehensive 0-1 goal programming model for project selection. *International Journal of Project Management*, 19 (4), pp. 243-252.
- Baker, D., Bridges, D., Hunter, R., Johnson, G., Krupa, J., Murphy, J. and Sorenson, K. (2001) *Guidebook to Decision-Making Methods*. Washington DC, USA: Department of Energy.
- Baldwin, D. and Sayward, F. G. (1979) *Heuristics for Determining Equivalence of Program Mutations*. 276, New Haven: Yale University,
- Bath, S. S., Uyar, M. U., Yu, W. and Fecko, M. A. Multiple Fault Models for Timed FSMs. *Proceedings of the IEEE Instrumentation and Measurement Technology Conference. IMTC 2006*. pp. 936-941.
- Behrmann, G., David, A. and Larsen, K. G. (2004) A tutorial on uppaal. *Bernardo M, Corradini F (eds) Formal Methods for the Design of Real-Time Systems (SFM-RT 2004)*, volume 3185 of *Lecture Notes in Computer Science*, pp. 200-236
- Beizer, B. (1990) *Software Testing Techniques*. 2ed ed. London: International Thomson Computer Press
- Bengtsson, J. and Yi, W. (2004) *Timed Automata: Semantics, Algorithms and Tools*. Springer Berlin / Heidelberg.
- Bertrand, N., Jéron, T., Stainer, A. and Krichen, M. (2011a) Off-line test selection with test purposes for non-deterministic timed automata. *Proceedings of the 17th international conference on Tools and algorithms for the construction and analysis of systems: part of the joint European conferences on theory and practice of software*, Germany: Springer-Verlag, pp. 96-111.
- Bertrand, N., Stainer, A., Jéron, T. and Krichen, M. (2011b) A game approach to determinize timed automata. *Proceedings of the 14th international conference on Foundations of software science and computational structures: part of the joint European conferences on theory and practice of software*, Germany: Springer-Verlag, pp. 245-259.
- Bhushan, N. and Rai, K. (2004) *Strategic Decision Making*. Springer.

- Blom, J., Hessel, A., Jonsson, B. and Pettersson, P. (2005) Specifying and generating test cases using observer automata. *4th International Workshop on Formal Approaches to Testing of Software 2004 (FATES'04)*, volume 3395 of *Lecture Notes in Computer Science*, Linz, Austria, pp. 125-139.
- Boehm, B. W. (1981) *Software Engineering Economics*. NJ: Prentice Hall PTR.
- Bonifácio, A. L. and Moura, A. V. (2011) A new method for testing timed systems. *Software Testing, Verification and Reliability*, pp. n/a-n/a.
- Bouquet, F. and Legeard, B. (2003) Reification of executable test scripts in formal specification-based test generation: The java card transaction mechanism case study. *FME 2003*, volume 2805 of *Lecture Notes in Computer Science*: Springer Verlag, pp. 778–795.
- Bousquet, L. d., Ramangalahy, S., Simon, S., Viho, C., Belinfante, A. and Vries, R. G. d. (2000) Formal Test Automation: The Conference Protocol with TGV/TORX. *Proceedings of the IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems: Tools and Techniques*, Deventer, The Netherlands: Kluwer, B.V., pp. 221-228.
- Bouyer, P. (2009) *From Qualitative to Quantitative Analysis of Timed Systems*. Mémoire d'habilitation Université Paris Computer and Information Science, Université Paris
- Bowen, J. P., Bogdanov, K., Clark, J., Harman, M., Hierons, R. M. and Krause, P. FORTEST: Formal methods and testing. *Proceedings of the 26th IEEE Computer Software and Applications*, California. pp. 91-101.
- Bowser, J. H. (1988) Reference Manual for Ada Mutant Operators. *Georgia Institute of Technology*, Atlanta, Georgia Technique Report,
- Bradfield, J. and Strling, C. Modal logics and mu-calculi: An introduction. *Handbook of Process Algebra*, Amsterdam, The Netherlands. Elsevier Science, pp. 293-330.
- Brandl, H., Weiglhofer, M. and Aichernig, B. K. Automated Conformance Verification of Hybrid Systems. *10th International Conference on Quality Software (QSIC)*, Zhangjiajie pp. 3-12.
- Briones, L. B. (2007) *Theories for model-based testing: real-time and coverage*. Thesis, Centre for Telematics and Information Technology.

- Briones, L. B. and Röhl, M. (2005) Test Derivation from Timed Automata. *Model-Based Testing of Reactive Systems*, volume 3472 of Lecture Notes in Computer Science, Berlin Heidelberg: Springer,
- Broekman, B. and Notenboom, E. (2003) *Testing Embedded Software*. London, UK: Addison-Wesley.
- Budd, T. A., DeMillo, R. A., Lipton, R. J. and Sayward, F. G. (1978) The Design of a Prototype Mutation System for Program Testing. *Proceedings the AFIPS National Computer Conference*, Anaheim, New Jersey: ACM, pp. 623-627.
- Budd, T. A. and Gopal, A. S. (1985) Program testing by specification mutation. *Computer Languages*, 10 (1), pp. 63-73.
- Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L. and Hwang, L. J. (1992) Symbolic model checking: 1020 States and beyond. *Information and Computation*, 98 (2), pp. 142-170.
- Buss, M. D. J. (1983) How to rank computer projects. *Harvard Business Review*, 61 (1), pp. 118-125.
- Cardell-Oliver, R. (2000) Conformance Tests for Real-Time Systems with Timed Automata Specifications. *Formal Aspects of Computing*, 12 (5), pp. 350-371.
- Cardell-Oliver, R. and Glover, T. (1998) A Practical and Complete Algorithm for Testing Real-Time Systems. *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Lyngby, Denmark, pp. 251-261.
- Cheng, E. L. and Li, H. (2002) Construction partnering process and associated critical success factors: Quantitative investigation. *Journal of Management in Engineering*, 18 (4), pp. 194-202.
- Chin, K. S., Chiu, S. and Tummala, V. M. R. (1999) An evaluation of success factors using AHP to implement ISO 14001 based EMS. *International Journal of Quality & Reliability Management*, 16 (4), pp. 341-361.
- Chow, T. S. (1978) Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engineering*, 4 (3), pp. 178-187.

- Cimatti, A., Clarke, E. M., Giunchiglia, F. and Roveri, M. (1999) NUSMV: A New Symbolic Model Verifier. *Proceedings of the 11th International Conference on Computer Aided Verification*, London, UK Springer-Verlag, pp. 495-499.
- Clarke, D. and Lee, I. (1997a) Automatic generation of tests for timing constraints from requirements. *Proceedings of the third International Workshop on Object-Oriented Real-Time Dependable Systems*, Newport Beach, CA, USA, pp. 199-206.
- Clarke, D. and Lee, I. (1997b) Automatic test generation for the analysis of a real-time system: Case study. *Proceeding of Third IEEE Real-Time Technology and Applications Symposium*, Montreal, Que., Canada, pp. 112-124.
- Clarke, E. M., Emerson, E. A. and Sistla, A. P. (1986) Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8(2), pp. 244-263.
- Clarke, E. M., Grumberg, O. and Peled, D. A. (2000) *Model Checking*. Cambridge, USA: The MIT Press.
- Connect, G. (2010) *USB/CAN adapter* [Online]. Available: <http://www.gridconnect.com/2010>.
- Dasarathy, B. (1985) Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them. *IEEE Transactions on Software Engineering*, 11 (1), pp. 80-86.
- Daws, C. and Yovine, S. (1996) Reducing the number of clock variables of timed automata. *Proceedings of the 17th IEEE Real-Time Systems Symposium*, Los Alamitos, CA, USA, pp. 73-81.
- De-Nicola, R. and Hennessy, M. (1984) Testing Equivalence for Processes. *Theoretical Computer Science*, (34), pp. 83-133.
- DeMillo, R. A. (1980) Mutation analysis as a tool for software quality assurance. . *Proceeding of the 4th Annual International Computer Software and Applications Conference (COMPSAC)*, Chicago, USA,
- DeMillo, R. A., Lipton, R. J. and Perlis, A. J. (1979) Social processes and proofs of theorems and programs. *Communications of the ACM* 22 (5), pp. 271-280.

- DeMillo, R. A., Lipton, R. J. and Sayward, F. G. (1978) Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*, 11 (4), pp. 34-41.
- Denzin, N. Y. K. and Lincoln, Y. S. (1998) *Collecting and Interpreting Qualitative Materials*. Thousand Oaks, California, USA: SAGE Publications.
- Dijkstra, E. W. (1970) Notes on Structured Programming. 70- WSK03, Eindhoven: Technological University,
- Dolan, J. G. (1989) Medical Decision Making Using the Analytic Hierarchy Process: Choice of Initial Antimicrobial Therapy for Acute Pyelonephritis. *Medical Decision Making*, 9 (1), pp. 51-56.
- Dssouli, R., Saleh, K., Aboulhamid, E., En-Nouaary, A. and Bourhfir, C. (1999) Test development for communication protocols: towards automation. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 31 (17), pp. 1835-1872.
- Dutta, R. and Burgess, T. F. (2003) Prioritising information system projects in higher education. *Campus-Wide information system*, 20 (4), pp. 152-158.
- Edwards, W. and Barron, F. H. (1994) Smarts and smarter: Improved simple methods for multiattribute utility measurement. *Organizational Behavior and Human Decision Processes*, 60 (3), pp. 306-325.
- Emerson, E. A. Temporal and modal logic. *Handbook of Theoretical Computer Science*, Amsterdam, The Netherlands. Elsevier Science, pp. 995-1072.
- Emerson, E. A. and Clarke, E. M. (1982) Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2 (3), pp. 241-266.
- En-Nouaary, A. (2008) A scalable method for testing real-time systems. *Software Quality Control*, 16 (1), pp. 3-22.
- En-Nouaary, A. and Dssouli, R. (2003) A Guided Method for Testing Timed Input Output Automata. *TestCom2003*, France, pp. 211-225.

- En-Nouaary, A., Dssouli, R. and Khendek, F. (2002) Timed Wp-method: testing real-time systems. *IEEE Transactions on Software Engineering*, 28 (11), pp. 1023-1038.
- En-Nouaary, A., Dssouli, R., Khendek, F. and Elqortobi, A. (1998) Timed test cases generation based on state characterization technique. *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, pp. 220-229.
- En-Nouaary, A. and Hamou-Lhadj, A. (2008) A Boundary Checking Technique for Testing Real-Time Systems Modeled as Timed Input Output Automata. *The Eighth International Conference on Quality Software, QSIC '08.*, pp. 209-215.
- En-Nouaary, A., Khendek, F. and Dssouli, R. (1999) Fault coverage in testing real-time systems. *Sixth International Conference on Real-Time Computing Systems and Applications, RTCSA '99 Hong Kong, China*, pp. 150-157.
- En-Nouaary, A. and Liu, G. (2004) Timed Test Cases Generation Based on MSC-2000 Test Purposes. in *Workshop on Integrated-reliability with Telecommunications and UML Languages (WITUL'04)*, part of the *15th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, Rennes, France,
- Fabbri, S. C. P. F., Maldonado, J. C. and Delamaro, M. E. (1999a) Proteum/FSM: a tool to support finite state machine validation based on mutation testing. *XIX Proceedings of International Conference of the Chilean Computer Science Society*, pp. 96-104.
- Fabbri, S. C. P. F., Maldonado, J. C., Masiero, P. C., Delamaro, M. E. and Wong, E. (1996) Mutation Testing Applied to Validate Specifications Based on Petri Nets. *Proceedings of the IFIP TC6 Eighth International Conference on Formal Description Techniques VIII*, London, UK, pp. 329-337.
- Fabbri, S. C. P. F., Maldonado, J. C., Sugeta, T. and Masiero, P. C. (1999b) Mutation Testing Applied to Validate Specifications Based on Statecharts. *Proceedings of 10th International Symposium on Software Reliability Engineering*, Boca Raton, Florida, pp. 210-219.
- Ferrante, J., Ottenstein, K. J. and Warren, J. D. (1987) The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9 (3), pp. 319-349.
- Fitzgerald, J., Hayes, I. J., Tarlecki, A., Bohnenkamp, H. and Belinfante, A. (2005) *Timed Testing with TorX*. Springer Berlin / Heidelberg.

- Fouchal, H., Petitjean, E. and Salva, S. (2000) An user-oriented testing of real time systems. *Proceedings of the International Workshop on Real-Time Embedded Systems, RTES'01*, London, UK: IEEE Computer Society Press
- Fujiwara, S., v. Bochmann, G., Khendek, F., Amalou, M. and Ghedamsi, A. (1991) Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17 (6), pp. 591-603.
- Gaudel, M.-c. Testing can be formal too. *TAPSOFT'95: Theory and Practice of Software Development*, Aarhus, Denmark. Springer-Verlag, pp. 82-96.
- Ghinea, G., Magoulas, G. D. and Siamitros, C. (2005) Multicriteria decision making for enhanced perception-based multimedia communication. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 35 (6), pp. 855-866.
- Glover, T. and Cardell-Oliver, R. (1999) A modular tool for test generation for real-time systems. *IEE Seminar Digests*, 1999 (6), pp. 3.
- Godefroid, P. (1997) Model checking for programming languages using VeriSoft. *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, Paris, France: ACM, pp. 174-186.
- Gonenc, G. (1970) A Method for the Design of Fault Detection Experiments. *IEEE Transactions on Computers*, C-19 (6), pp. 551-558.
- Goodwin, P. and Wright, G. (1999) *Decision Analysis for Management Judgment*. Wiley.
- Goodwin, P. and Wright, G. (2000) *Decision Analysis or Management Judgement*. John Wiley and Sons.
- Gopal, A. S. and Budd, T. A. (1983) Program Testing by Specification Mutation. Tucson, Arizona, Technical Report, TR 83-17: University of Arizona,
- Grieskamp, W., Kicillof, N., Stobie, K. and Braberman, V. (2011) Model-based quality assurance of protocol documentation: tools and methodology. *Software Testing, Verification and Reliability*, 21 (1), pp. 55-71.

- Harel, D. and Gery, E. (1997) Executable Object Modeling with Statecharts. *IEEE Computer*, 30 (7), pp. 31-42.
- Harel, D. and Naamad, A. (1996) The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology* 5(4), pp. 293-333.
- Harel, D. and Pnueli, A. (1985) *On the development of reactive systems*. New York: Springer-Verlag.
- Harman, M., Hierons, R. and Danicic, S. (2001) The Relationship Between Program Dependence and Mutation Analysis. *Proceedings of the 1st Workshop on Mutation Analysis*, San Jose, California: published in book form, as Mutation Testing for the New Century, pp. 5-13.
- Hessel, A., Larsen, K. G., Mikucionis, M., Nielsen, B., Pettersson, P. and Skou, A. (2008) Testing Real-Time Systems Using UPPAAL. *Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS*, Berlin Heidelberg, pp. 77-117.
- Hessel, A., Larsen, K. G., Nielsen, B., Pettersson, P. and Skou, A. Time-optimal Real-Time Test Case Generation using UPPAAL. *3rd International Workshop on Formal Approaches to Testing of Software (FATES)* Montreal, Canada. Springer-Verlag, pp. 114-130.
- Hessel, A. and Pettersson, P. (2007a) Cover - A Real-Time Test Case Generation Tool. *19th IFIP International Conference on Testing of Communicating Systems and 7th International Workshop on Formal Approaches to Testing of Software*,
- Hessel, A. and Pettersson, P. (2007b) Model-Based Testing of a WAP Gateway: an Industrial Study. *Proceedings of the 11th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2006) vol. 4346, Springer, Heidelberg* pp. 116-131.
- Hierons, R. M. (2004) Testing from a nondeterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53 (10), pp. 1330-1342.
- Hierons, R. M., Bogdanov, K., Bowen, J. P., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Luttgen, G., Simons, A. J. H., Vilkomir, S., Woodward, M. R. and Zedan, H. (2009) Using formal specifications to support testing. *ACM Computing Surveys*, 41 (2), pp. 1-76.

- Hierons, R. M. and Harman, M. (2004) Testing conformance of a deterministic implementation against a non-deterministic stream X-machine. *Theoretical Computer Science*, 323 (1-3), pp. 191-233.
- Hierons, R. M. and Merayo, M. G. (2007) Mutation Testing from Probabilistic Finite State Machines. *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, Winsdor, UK: IEEE Computer Society, pp. 141-150.
- Hierons, R. M. and Merayo, M. G. (2009) Mutation testing from probabilistic and stochastic finite state machines. *Journal of Systems and Software* 82 (11), pp. 1804-1818.
- Hierons, R. M., Sadeghipour, S. and Singh, H. (2001) Testing a system specified using Statecharts and Z. *Information and Software Technology*, 43 (2), pp. 137-149.
- Higashino, T., Nakata, A., Taniguchi, K. and Cavalli, A. R. (1999) Generating Test Cases for a Timed I/O Automaton Model. *Proceedings of 12th International Workshop on Testing Communicating Systems: Method and Applications*, Budapest, Hungary: Kluwer, B.V., pp. 197 - 214
- Hoare, C. A. R. (1985) *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentice Hall International Series in Computer Science.
- Holzmann, G. J. (2003) *The Spin Model Checker: Primer and Reference Manual*. 1st ed.: Addison Wesley.
- Hong, H. S., Cha, S. D., Lee, I., Sokolsky, O. and Ural, H. (2003) Data flow testing as model checking. *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon: IEEE Computer Society, pp. 232-242.
- Hong, H. S., Lee, I., Sokolsky, O. and Ural, H. (2002) A Temporal Logic Based Theory of Test Coverage and Generation. *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*: Springer-Verlag, pp. 327-341.
- Hong, H. S., Lee, I., Sokolsy, O. and Cha, S. D. Automatic test generation from Statecharts using model checking. *Formal Approaches To Testing Of Software*, Aarhus, Denmark. pp. 15-30.

- Huang, S. M., Chang, I.-C., Li, S. H. and Lin, M. T. (2004) Assessing risk in ERP projects: identify and prioritise the factors. *Industrial Management and Data Systems*, 104 (8), pp. 681-688.
- Hussain, S. (2008) *Mutation Clustering*. Masters Thesis, King's College London.
- ISO. (1989) ISO 8807:1989 Information Processing Systems, Open Systems Interconnection—LOTOS—A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Geneva, Switzerland: ISO.,
- ITU-T. (1997) Recommendation Z.500 Framework on Formal Methods in Conformance Testing. Geneva, Switzerland: International Telecommunications Union,
- Jackson, J. (2001) Prioritising customers and other stakeholders using AHP. *European Journal of Marketing*, 35 (7/8), pp. 858-871.
- Jensen, R. E. (1982) Reporting of management forecasts: An eigenvector model for elicitation and review of forecasts. *Decision Sciences*, 13 (1), pp. 15-37.
- Jensen, R. E. and Spencer, R. W. (1986) Matrix scaling of subjective probabilities of economic forecasts. *Economics Letters*, 20 (3), pp. 221-225.
- Jia, Y. and Harman, M. (2010) An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering*, PP (99), pp. 1-31.
- Kalaji, A. (2010) *Search-Based Software Engineering: A Search- Based Approach for Testing from Extended Finite State Machine (EFSM) Models*. PhD Thesis, Information Systems and Computing, Brunel University.
- Kamal, M. M. (2008) *Investigation Enterprise Applications Integration (EAI) Adoption in the Local Government Authorities (LGAs)*. PhD Thesis, Information Systems and Computing, Brunel University.
- Karsak, E. E., Sozer, S. and Alptekin, S. E. (2003) Product planning in quality function deployment using a combined analytic network process and goal programming approach. *Computers and Industrial Engineering*, 44 (1), pp. 171-190.
- Kemmerer, R. A. (1985) Testing Formal Specifications to Detect Design Errors. *IEEE Transactions on Software Engineering*, 11 (1), pp. 32-43.

- Kepner, H. and Tregoe, B. B. (1981) *The New Rational Manager*. Princeton Research Press.
- Khoumsi, A. (2002) A Method for Testing the Conformance of Real Time Systems. *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*: Springer-Verlag, pp. 331 - 354
- Khoumsi, A., En-Nouaary, A., Dssouli, R. and Akalay, M. (2000) A new method for testing real time systems. *Proceedings of Seventh International Conference on Real-Time Computing Systems and Applications*, pp. 441-450.
- Kohavi, Z. (1978) *Switching and finite automata theory*. New York: McGraw- Hill.
- Kov, G., Pap, Z., Viet, D. L., Wu-Hen-Chang, A. and Csopaki, G. (2003) Applying mutation analysis to SDL specifications. *Proceedings of the 11th international conference on System design*, Stuttgart, Germany: Springer-Verlag, pp. 269-284.
- Krichen, M. and Tripakis, S. (2004) *Real-Time Testing with Timed Automata Testers and Coverage Criteria*. Springer Berlin / Heidelberg.
- Krichen, M. and Tripakis, S. (2005) An Expressive and Implementable Formal Framework for Testing Real-Time Systems. *17th International Conference on Testing of Communicating Systems, TestCom'05*, Montreal, Canada, pp. 209-225.
- Krichen, M. and Tripakis, S. (2009) Conformance testing for real-time systems. *Formal Methods in System Design*, 34 (3), pp. 238-304.
- Lai, V. S., Trueblood, R. P. and Wong, B. K. (1999) Software selection: A case study of the application of the analytical hierarchy process to the selection of a multimedia authoring system. *Information and Management*, 36 pp. 221-232.
- Larsen, K. G., Mikucionis, M. and Nielsen, B. (2005a) Online Testing of Real-time Systems Using Uppaal. *Formal Approaches to Software Testing*: Springer Berlin / Heidelberg, pp. 79-94.
- Larsen, K. G., Mikucionis, M., Nielsen, B. and Skou, A. (2005b) Testing real-time embedded software using UPPAAL-TRON: an industrial case study. *Proceedings of the 5th ACM international conference on Embedded software*, Jersey City, NJ, USA: ACM, pp. 299 - 306.

- Larsen, K. G. and Wang, Y. (1997) Time-abstracted bisimulation: implicit specifications and decidability. *Information and Computation*, 134 (2), pp. 75-101.
- Larsen, K. G. and Yi, W. (1993) Time abstracted bisimulation: implicit specification and decidability. In *Proceedings mathematical foundations of programming semantics (MFPS 9)*, New Orleans, USA, pp. 160–176.
- Lee, D. and Yannakakis, M. (1996) Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84 (8), pp. 1090-1123.
- Lee, J. W. and Kim, S. H. (2000) Using analytic network process and goal programming for interdependent information system project selection. *Computers and Operations Research*, 27 pp. 367-382.
- Lipton, R. (1971) Fault Diagnosis of Computer Programs. Student Report, Carnegie Mellon University,
- Lu, M. H., Madu, C. N., Kuei, C. and Winokur, D. (1994) Integrating QFD, AHP and Benchmarking in Strategic Marketing. *Journal of Business and Industrial Marketing*, 82 (2), pp. 250-259.
- Lynch, N. A. and Attiya, H. (1992) Using mappings to prove timing properties. *Distributed Computing* 6(2), pp. 121-139.
- Ma, Y.-S., Kwon, Y.-R. and Offutt, J. (2002) Inter-Class Mutation Operators for Java. *Proceedings of the 13th International Symposium on Software Reliability Engineering*, Annapolis, Maryland: IEEE Computer Society, pp. 352.
- Ma, Y.-S., Offutt, J. and Kwon, Y. R. (2005) MuJava: an automated class mutation system: Research Articles. *Software Testing, Verification and Reliability*, 15 (2), pp. 97-133.
- Mandrioli, D., Morasca, S. and Morzenti, A. (1995) Generating test cases for real-time systems from logic specifications. *ACM Transactions on Computer Systems*, 13 (4), pp. 365-398.
- Mathur, A. P. Performance, effectiveness, and reliability issues in software testing. *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, Tokyo , Japan pp. 604-605.

- McCaffrey, J. (2005) Test Run: The Analytic Hierarchy Process. *MSDN Magazine*, June: Microsoft Corporation,
- McMillan, K. L. (1993) *Symbolic Model Checking*. Dordrecht, The Netherlands: Kluwer Academic Publishers.
- Merayo, M. G., Nunez, M. and Rodriguez, I. (2008) Formal testing from timed finite state machines. *Computer Networks: The International Journal of Computer and Telecommunications Networking* 52 (2), pp. 432-460.
- MicroshipDirect (2010) *MCP2515DM-BM CAN Bus Monitor Demo Board* [Online]. Available: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en537141.
- Milner, R. (1989) *Communication and Concurrency*. Englewood Cliffs, NJ: Prentice Hall International Series in Computer Science.
- Mitsching, R., Weise, C., Kolbe, A., Bohnenkamp, H. and Berzen, N. (2009) Towards an Industrial Strength Process for Timed Testing. *International Conference on Software Testing, Verification and Validation Workshops*, Denver, Colorado, pp. 29-38.
- NetBeans (2010). Available: <http://www.netbeans.org>.
- Nicolescu, G. and Mosterman, P. J. (2009) *Modeling, Verification, and Testing Using Timed and Hybrid Automata*. CRC Press
- Nicollin, X., Sifakis, J. and Yovine, S. (1992) Compiling Real-Time Specifications into Extended Automata. *IEEE transactions on Software Engineering*, 18 (9), pp. 794-804.
- Nielsen, B. and Skou, A. (1998) Automated Test Generation from Timed Automata. In *5th international symposium on formal techniques in real-time and fault tolerant systems FTRTFT'98*, Lyngby, Denmark, pp. 59-77.
- Nielsen, B. and Skou, A. Test generation for time critical systems: Tool and case study. *13th Euromicro Conference on Real-Time Systems*, Delft , Netherlands. pp. 155-162.

- Nielsen, B. and Skou, A. (2003) Automated test generation from timed automata. *International Journal on Software Tools for Technology Transfer*, 5 pp. 59–77.
- Offutt, A. J. and Jie, P. (1996) Detecting equivalent mutants and the feasible path problem. *Proceedings of the Eleventh Annual Conference on Computer Assurance*, Gaithersburg, MD , USA pp. 224-236.
- Offutt, A. J. and King, K. N. (1987) A Fortran 77 interpreter for mutation analysis. *Papers of the Symposium on Interpreters and interpretive techniques*, St. Paul, Minnesota, United States: ACM, pp. 177-188.
- Offutt, A. J. and Xu, W. (1996) Mutation Operators for Ada. *George Mason University Fairfax, Virginia, USA: Technique Report*,
- Ouedraogo, L., Koumsi, A. and Nourelfath, M. (2010) SetExp: a method of transformation of timed automata into finite state automata. *Real-Time Syst.*, 46 (2), pp. 189-250.
- Pazul., K. (1999) Controller Area Network (CAN) Basics. *Microchip Technology Inc. Preliminary DS00713A*, pp. 1-7.
- Pinto Ferraz Fabbri, S. C., Delamaro, M. E., Maldonado, J. C. and Masiero, P. C. (1994) Mutation analysis testing for finite state machines. *5th International Symposium on Software Reliability Engineering*, , pp. 220-229.
- Pnueli, A. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science*, Providence, RI, USA pp. 46-57.
- Pressman, R. S. (2010) *Software Engineering: A Practitioner's Approach*. 7th ed. New York: McGraw-Hill.
- Probert, R. L. and Guo, F. (1991) Mutation testing of protocols: Principles and preliminary experimental results. *Proceedings of the IFIP TC6 Third International Workshop on Protocol Teste Systems*, North Holand, pp. 57-76.
- Queille, J.-P. and Sifakis, J. (1982) Specification and verification of concurrent systems in CESAR. *Proceedings of the 5th Colloquium on International Symposium on Programming*, London, UK: Springer-Verlag, pp. 337-351.

- Ramanathan, R. (1995) Using AHP for resource allocation problems. *European Journal of Operational Research*, 80 (2), pp. 410-417.
- Rapps, S. and Weyuker, E. J. (1985) Selecting Software Test Data Using Data Flow Information. *IEEE Transactions on Software Engineering*, 11 (4), pp. 367-375.
- Ravi, V., Shankar, R. and Tiwari, M. K. (2005) Analyzing alternatives in reverse logistics for end-of-life computers: ANP and balanced scorecard approach. *Computers and Industrial Engineering*, 48 (2), pp. 327-356.
- Rivest, R. L. and Schapire, R. E. (1989) Inference of finite automata using homing sequences. *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, Seattle, Washington, United States: ACM, pp. 411-420.
- Robson, A. and Henderson, W. (2010) The Production Cell A Real-Time Case Study Version 1.0. Northumbria University,
- Rollet, A. (2003) Testing robustness of real-time embedded systems. In *Proceedings of Workshop On Testing Real-Time and Embedded Systems (WTRTES), Satellite Workshop of FM 2003 Symposium*, Pisa, Italy,
- Rütz, C. and Schmaltz, J. (2011) An Experience Report on an Industrial Case-Study about Timed Model-Based Testing with UPPAAL-TRON. *Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, Berlin IEEE Computer Society, pp. 39-46.
- Saaty, T. L. (1977) A scaling method for priorities in hierarchical structures. *Journal of Mathematical Psychology*, 15 (3), pp. 234-281.
- Saaty, T. L. (1980) *The Analytic Hierarchy Process: Planning, Priority Setting: Resource Allocation*. McGraw-Hill, New York.
- Saaty, T. L. (1987) A new macroeconomic forecasting and policy evaluation method using the analytic hierarchy process. *Mathematical Modelling*, 9 (3/5), pp. 219-231.
- Saaty, T. L. (1990a) How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, 48 (1), pp. 9-26.

- Saaty, T. L. (1990b) *Multicriteria Decision Making: The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. Pittsburgh: RWS Publications.
- Saaty, T. L. (2001) *Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World, New Edition 2001* RWS Publications
- Saaty, T. L. (2008) Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, 1 (1), pp. 83-98.
- Saaty, T. L. and Kearns, K. P. (1985) *Analytical Planning*. Pergamon Press.
- Saaty, T. L. and Vargas, L. G. (1984) The legitimacy of rank reversal. *Omega*, 12 (5), pp. 513-516.
- Saaty, T. L. and Vargas, L. G. (1991) *Prediction, Projection, and Forecasting: Applications of the Analytic Hierarchy Process in Economics, Finance, Politics, Games, and Sports*. Kluwer Academic Publication.
- Saaty, T. L. and Vargas, L. G. (2000) *Models, Methods, Concepts and Applications of the Analytic Hierarchy Process*. Springer.
- Salmeron, J. L. and Herrero, I. (2005) An AHP-based methodology to rank critical success factors of executive information systems. *Computer Standards and Interfaces*, 28 pp. 1-12.
- Santhanam, R. and Kyparisis, G. J. (1996) A decision model for interdependent information system project selection. *European Journal of Operational Research*, 89 pp. 380-399.
- Sarikaya, B. and Bochmann, G. (1984) Synchronization and Specification Issues in Protocol Testing. *IEEE Transactions on Communications*, 32 (4), pp. 389-395.
- Souza, S. D. R. S. D., Maldonado, C., Fabbri, S. C. P. F. and Souza, W. L. D. (1999) Mutation Testing Applied to Estelle Specifications. *Software Quality Control*, 8 (4), pp. 285-301.
- Spivey, J. M. (1992) *The Z Notation: A Reference Manual*. 2nd ed. Englewood Cliffs: Prentice Hall.

- Springintveld, J., Vaandrager, F. and D'Argenio, P. R. (2001) Testing timed automata. *Theoretical Computer Science*, 254 (1-2), pp. 225-257.
- Springintveld, J. and Vaandrager, F. W. (1996) Minimizable Timed Automata. *Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Uppsala, Sweden: Springer-Verlag,
- Sugeta, T., Maldonado, J. and Wong, W. (2004) *Mutation Testing Applied to Validate SDL Specifications*. Springer Berlin / Heidelberg.
- Tahat, L. H., Vaysburg, B., Korel, B. and Bader, A. J. Requirement-based automated black-box test generation. *25th Annual International Computer Software and Applications Conference*, Chicago, IL , USA pp. 489-495.
- Taylor, B. (1980) Introducing Real-time Constraints into Requirements and High Level Design of Operating Systems. *In Proc. of the National Telecommunications Conference*, Houston, TX, pp. 18.5.1–18.5.5.
- Tindell, K., Burns, A. and Wellings, A. J. (1995) Calculating controller area network (can) message response times. *Control Engineering Practice*, 3 (8), pp. 1163-1169.
- Trakhtenbrot, M. (2007) New Mutations for Evaluation of Specification and Implementation Levels of Adequacy in Testing of Statecharts Models. *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, Winsdor, UK: IEEE Computer Society, pp. 151-160.
- Tretmans, J. (1996) Test Generation with Inputs, Outputs, and Quiescence. *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*: Springer-Verlag, pp. 127-146
- Ülengin, F. (1994) Forecasting foreign exchange rates: A comparative evaluation of AHP. *Omega*, 22 (5), pp. 505-519.
- Untch, R. H., Offutt, A. J. and Harrold, M. J. (1993) Mutation analysis using mutant schemata. *Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis*, Cambridge, Massachusetts, United States: ACM, pp. 139-148.

- Ural, H. and Yang, B. (1991) A test sequence selection method for protocol testing. *IEEE Transactions on Communications*, 39 (4), pp. 514-523.
- Utting, M. and Legeard, B. (2007) *Practical model-based testing: a tools approach* San Francisco: Elsevier.
- Uyar, M. Ü., Wang, Y., Batth, S. S., Wise, A. and Fecko, M. A. (2005) Timing Fault Models for Systems with Multiple Timers. *Testing of Communicating Systems: Springer Berlin / Heidelberg*, pp. 192-208.
- Vadim Okun , Y. Y. (2004) Specification mutation for test generation and analysis. *University of Maryland at Baltimore County, Catonsville, MD*, pp. 1-72.
- Vilela, P., Machado, M. and Wong, W. E. (2002) Testing for Security Vulnerabilities in Software. *Software Engineering and Applications*,
- Walsham, G. (1995) Interpretive Case Studies in IS Research: Nature and Method. *European Journal of Information Systems*, 4 pp. 74-81.
- Wang, Y., Uyar, M. Ü., Batth, S. S. and Fecko, M. A. (2009) Fault masking by multiple timing faults in timed EFSM models. *Computer Networks*, 53 (5), pp. 596-612.
- Wei, C. C., Chien, C. F. and Wang, M. J. (2005) An AHP-based approach to ERP systems selection. *International Journal for Production Economics*, 96 (1), pp. 47-62.
- Wolper, P. (1981) Temporal logic can be more expressive. *22nd Annual Symposium on Foundations of Computer Science, SFCS '81*, Nashville, TN, USA, pp. 340-348.
- Woodward, M. R. (1992) OBJTEST: an experimental testing tool for algebraic specifications. *IEE Colloquium on Automating Formal Methods for Computer Assisted Prototyping* pp. 2.
- Woodward, M. R. (1993) Errors in algebraic specifications and an experimental mutation testing tool. *Software Engineering Journal*, 8 (4), pp. 211-224.
- Woodward, M. R. and Halewood, K. (1988) From weak to strong, dead or alive? an analysis of some mutation testing issues. *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis*, Banff Albert, Canda, pp. 152-158.

- Yin, R. K. (1994) *Case Study Research Design and Methods*. London, UK: Sage.
- Yoon, H., Choi, B. and Jeon, J.-O. (1998) Mutation-Based Inter-Class Testing. *Proceedings of the Fifth Asia Pacific Software Engineering Conference*, Taipei, Taiwan: IEEE Computer Society, pp. 174.
- Zeng, G., Jiang, R., Huang, G., Xu, M. and Li, J. (2007) Optimization of wastewater treatment alternative selection by hierarchy grey relational analysis. *Journal of environmental management*, 82 (2), pp. 250-259.
- Zheng, M., Alagar, V. and Ormandjieva, O. (2008) Automated generation of test suites from formal specifications of real-time reactive systems. *Journal of Systems and Software* 81 (2), pp. 286-304.

Appendix A

Timed Specification Mutation Operators

This Appendix presents detailed timed mutation operators (RTC, WTC and STC) that have been used for SMA application on TA specification models.

Restricting Timing Constraint (RTC)	
Clock Guard	Mutated Clock Guards
$x < a$	$x < a - \epsilon$
	$x \leq a - \epsilon$
	$\epsilon < x < a$
	$\epsilon \leq x < a$
	$\epsilon 1 < x < a - \epsilon 2$
	$\epsilon 1 \leq x < a - \epsilon 2$
	$\epsilon 1 < x \leq a - \epsilon 2$
	$\epsilon 1 \leq x \leq a - \epsilon 2$
$x \leq a$	$x < a$
	$x \leq a - \epsilon$
	$x < a - \epsilon$
	$\epsilon \leq x \leq a$
	$\epsilon < x \leq a$
	$\epsilon \leq x < a$
	$\epsilon < x < a$
	$\epsilon 1 \leq x \leq a - \epsilon 2$
	$\epsilon 1 < x \leq a - \epsilon 2$
	$\epsilon 1 \leq x < a - \epsilon 2$
	$\epsilon 1 < x < a - \epsilon 2$
$x > a$	$x > a + \epsilon$
	$x \geq a + \epsilon$
	$a < x < \epsilon$
	$a < x \leq \epsilon$
	$a + \epsilon 1 < x < \epsilon 2$
	$a + \epsilon 1 < x \leq \epsilon 2$
	$a + \epsilon 1 \leq x < \epsilon 2$
	$a + \epsilon 1 \leq x \leq \epsilon 2$

Restricting Timing Constraint (RTC)	
Clock Guard	Mutated Clock Guards
$x \geq a$	$x > a$
	$x \geq a + \epsilon$
	$x > a + \epsilon$
	$a \leq x < \epsilon$
	$a \leq x \leq \epsilon$
	$a < x < \epsilon$
	$a < x \leq \epsilon$
	$a + \epsilon 1 \leq x \leq \epsilon 2$
	$a + \epsilon 1 < x \leq \epsilon 2$
	$a + \epsilon 1 \leq x < \epsilon 2$
	$a + \epsilon 1 < x < \epsilon 2$
	$a < x < b$
$a + \epsilon \leq x < b$	
$a < x < b - \epsilon$	
$a < x \leq b - \epsilon$	
$a + \epsilon < x < b - \epsilon$	
$a + \epsilon \leq x < b - \epsilon$	
$a + \epsilon < x \leq b - \epsilon$	
$a + \epsilon \leq x \leq b - \epsilon$	
$a \leq x < b$	$a < x < b$
	$a + \epsilon < x < b$
	$a + \epsilon \leq x < b$
	$a \leq x < b - \epsilon$
	$a \leq x \leq b - \epsilon$
	$a < x < b - \epsilon$
	$a < x \leq b - \epsilon$
	$a + \epsilon < x < b - \epsilon$
	$a + \epsilon \leq x < b - \epsilon$
	$a + \epsilon < x \leq b - \epsilon$
$a + \epsilon \leq x \leq b - \epsilon$	
$a < x \leq b$	$a < x < b$
	$a + \epsilon < x < b$
	$a + \epsilon \leq x < b$
	$a < x < b - \epsilon$
	$a < x \leq b - \epsilon$
	$a + \epsilon < x < b - \epsilon$
	$a + \epsilon \leq x < b - \epsilon$
	$a + \epsilon < x \leq b - \epsilon$
	$a + \epsilon \leq x \leq b - \epsilon$
	$a + \epsilon < x \leq b - \epsilon$
$a + \epsilon \leq x \leq b - \epsilon$	

Restricting Timing Constraint (RTC)	
Clock Guard	Mutated Clock Guards
$a \leq x \leq b$	$a < x \leq b$
	$a \leq x < b$
	$a < x < b$
	$a + \epsilon < x \leq b$
	$a + \epsilon \leq x \leq b$
	$a + \epsilon < x < b$
	$a + \epsilon \leq x < b$
	$a \leq x < b - \epsilon$
	$a \leq x \leq b - \epsilon$
	$a < x < b - \epsilon$
	$a < x \leq b - \epsilon$
	$a + \epsilon < x < b - \epsilon$
	$a + \epsilon \leq x < b - \epsilon$
	$a + \epsilon < x \leq b - \epsilon$
$a + \epsilon \leq x \leq b - \epsilon$	
True	$x > \epsilon$
	$x \geq \epsilon$
	$x < \epsilon$
	$x \leq \epsilon$
	$\epsilon 1 < x < \epsilon 2$
	$\epsilon 1 \leq x < \epsilon 2$
	$\epsilon 1 < x \leq \epsilon 2$
	$\epsilon 1 \leq x \leq \epsilon 2$

Widening Timing Constraint (WTC)	
Clock Guard	Mutated Clock Guards
$x < a$	$x \leq a$
	$x < a + \epsilon$
	$x \leq a + \epsilon$
	True
$x \leq a$	$x \leq a + \epsilon$
	$x < a + \epsilon$
	True
$x > a$	$x \geq a$
	$x > a - \epsilon$
	$x \geq a - \epsilon$
	True
$x \geq a$	$x \geq a - \epsilon$
	$x > a - \epsilon$
	True
$a < x < b$	$a \leq x < b$
	$a < x \leq b$
	$a \leq x \leq b$
	$a - \epsilon < x < b$
	$a - \epsilon \leq x < b$
	$a - \epsilon < x \leq b$
	$a - \epsilon \leq x \leq b$
	$a < x < b + \epsilon$
	$a < x \leq b + \epsilon$
	$a \leq x < b + \epsilon$
	$a \leq x \leq b + \epsilon$
	$a - \epsilon < x < b + \epsilon$
	$a - \epsilon \leq x < b + \epsilon$
	$a - \epsilon < x \leq b + \epsilon$
	$a - \epsilon \leq x \leq b + \epsilon$
True	
$a \leq x < b$	$a \leq x \leq b$
	$a - \epsilon < x < b$
	$a - \epsilon \leq x < b$
	$a - \epsilon < x \leq b$
	$a - \epsilon \leq x \leq b$
	$a \leq x < b + \epsilon$
	$a \leq x \leq b + \epsilon$
	$a - \epsilon < x < b + \epsilon$
	$a - \epsilon \leq x < b + \epsilon$
	$a - \epsilon < x \leq b + \epsilon$
	$a - \epsilon \leq x \leq b + \epsilon$
	True

Widening Timing Constraint (WTC)	
Clock Guard	Mutated Clock Guards
$a < x \leq b$	$a \leq x \leq b$
	$a - \epsilon < x \leq b$
	$a - \epsilon \leq x \leq b$
	$a < x < b + \epsilon$
	$a < x \leq b + \epsilon$
	$a \leq x < b + \epsilon$
	$a \leq x \leq b + \epsilon$
	$a - \epsilon < x < b + \epsilon$
	$a - \epsilon \leq x < b + \epsilon$
	$a - \epsilon < x \leq b + \epsilon$
	$a - \epsilon \leq x \leq b + \epsilon$
True	
$a \leq x \leq b$	$a - \epsilon < x \leq b$
	$a - \epsilon \leq x \leq b$
	$a \leq x < b + \epsilon$
	$a \leq x \leq b + \epsilon$
	$a - \epsilon < x < b + \epsilon$
	$a - \epsilon \leq x < b + \epsilon$
	$a - \epsilon < x \leq b + \epsilon$
	$a - \epsilon \leq x \leq b + \epsilon$
	True

Shifting Timing Constraint (STC)	
Clock Guard	Mutated Clock Guards
$x < a$	$\epsilon \leq x \leq a$
	$\epsilon < x \leq a$
	$\epsilon \leq x < a + \epsilon$
	$\epsilon < x < a + \epsilon$
	$\epsilon \leq x \leq a + \epsilon$
	$\epsilon < x \leq a + \epsilon$
$x \leq a$	$\epsilon \leq x \leq a + \epsilon$
	$\epsilon < x \leq a + \epsilon$
	$\epsilon \leq x < a + \epsilon$
	$\epsilon < x < a + \epsilon$
$x > a$	$a \leq x < \epsilon$
	$a \leq x \leq \epsilon$
	$a - \epsilon 1 < x < \epsilon 2$
	$a - \epsilon 1 < x \leq \epsilon 2$
	$a - \epsilon 1 \leq x < \epsilon 2$
	$a - \epsilon 1 \leq x \leq \epsilon 2$
$x \geq a$	$a - \epsilon 1 \leq x < \epsilon 2$
	$a - \epsilon 1 \leq x \leq \epsilon 2$
	$a - \epsilon 1 < x < \epsilon 2$
	$a - \epsilon 1 < x \leq \epsilon 2$
$a < x < b$	$a + \epsilon < x \leq b$
	$a + \epsilon \leq x \leq b$
	$a + \epsilon < x < b + \epsilon$
	$a + \epsilon \leq x < b + \epsilon$
	$a + \epsilon < x \leq b + \epsilon$
	$a + \epsilon \leq x \leq b + \epsilon$
	$a \leq x < b - \epsilon$
	$a \leq x \leq b - \epsilon$
	$a - \epsilon < x < b - \epsilon$
	$a - \epsilon \leq x < b - \epsilon$
	$a - \epsilon < x \leq b - \epsilon$
	$a - \epsilon \leq x \leq b - \epsilon$
$a \leq x < b$	$a < x \leq b$
	$a + \epsilon < x < b + \epsilon$
	$a + \epsilon \leq x < b + \epsilon$
	$a + \epsilon < x \leq b + \epsilon$
	$a + \epsilon \leq x \leq b + \epsilon$
	$a - \epsilon < x < b - \epsilon$
	$a - \epsilon \leq x < b - \epsilon$
	$a - \epsilon < x \leq b - \epsilon$
$a - \epsilon \leq x \leq b - \epsilon$	

Shifting Timing Constraint (STC)	
Clock Guard	Mutated Clock Guards
$a < x \leq b$	$a + \epsilon < x < b + \epsilon$
	$a + \epsilon \leq x < b + \epsilon$
	$a + \epsilon < x \leq b + \epsilon$
	$a + \epsilon \leq x \leq b + \epsilon$
	$a \leq x < b$
	$a - \epsilon < x < b - \epsilon$
	$a - \epsilon \leq x < b - \epsilon$
	$a - \epsilon < x \leq b - \epsilon$
$a \leq x \leq b$	$a < x \leq b + \epsilon$
	$a \leq x < b + \epsilon$
	$a + \epsilon < x < b + \epsilon$
	$a + \epsilon \leq x < b + \epsilon$
	$a + \epsilon < x \leq b + \epsilon$
	$a + \epsilon \leq x \leq b + \epsilon$
	$a - \epsilon \leq x < b$
	$a - \epsilon < x < b$
	$a - \epsilon < x < b - \epsilon$
	$a - \epsilon \leq x < b - \epsilon$
	$a - \epsilon < x \leq b - \epsilon$
	$a - \epsilon \leq x \leq b - \epsilon$

Appendix B

CRC Calculation

This appendix presents clock region coverage achieved by each testing approach (SM, BCT and PA) that generate timed test cases from specification models of the ‘production-cell’ test bed (Control Panel, Conveyor, Robot-In and Robot-Out). The number of effective clock regions was calculated for each input transition using Equation (3.5). The number of clock regions covered by each testing approach was then observed to calculate the CRC ratio for each input transition. The Final CRC was calculated for the whole specification model using Equation (4.1).

CRC Ratio for input transitions in the Control Panel				
Transitions	NCR	SM	BCT	PA
S0-S1	Total No.	55		
	Covered	1	1	55
	Ratio	0.018	0.018	1
S2-S3	Total No.	11		
	Covered	2	3	11
	Ratio	0.181	0.272	1
S3-S4	Total No.	55		
	Covered	1	1	55
	Ratio	0.018	0.018	1
S5-S6	Total No.	57		
	Covered	2	3	57
	Ratio	0.035	0.055	1
S6-S7	Total No.	55		
	Covered	1	1	55
	Ratio	00.18	0.018	1
S8-S9	Total No.	9		
	Covered	2	3	9
	Ratio	0.222	0.333	1
S9-S10	Total No.	55		
	Covered	1	1	55
	Ratio	0.018	0.018	1

CRC Ratio for input transitions in the Conveyor				
Transitions	NCR	SM	BCT	PA
S0-S1	Total No.	11		
	Covered	1	1	11
	Ratio	0.091	0.091	1
S1-S2	Total No.	11		
	Covered	1	1	11
	Ratio	0.091	0.091	1
S3-S4	Total No.	13		
	Covered	2	3	13
	Ratio	0.154	0.231	1
S4-S5	Total No.	11		
	Covered	1	1	11
	Ratio	0.091	0.091	1
S6-S7	Total No.	11		
	Covered	1	1	11
	Ratio	0.091	0.091	1
S8-S9	Total No.	10		
	Covered	2	3	10
	Ratio	0.2	0.3	1
S9-S10	Total No.	11		
	Covered	1	1	11
	Ratio	0.091	0.091	1

CRC Ratio for input transitions in the Robot IN				
Transitions	Number of Regions	SM	BCT	PA
S0-S1	Total No.	15		
	Covered	1	1	15
	Ratio	0.067	0.067	1
S2-S3	Total No.	16		
	Covered	2	3	16
	Ratio	0.125	0.188	1
S4-S5	Total No.	10		
	Covered	2	3	10
	Ratio	0.2	0.3	1
S6-S0	Total No.	15		
	Covered	1	1	15
	Ratio	0.067	0.067	1

CRC Ratio for input transitions in the Robot Out				
Transitions	NCR	SM	BCT	PA
S0-S1	Total No.	21		
	Covered	1	1	21
	Ratio	0.048	0.048	1
S2-S3	Total No.	22		
	Covered	2	3	22
	Ratio	0.091	0.136	1
S4-S5	Total No.	10		
	Covered	2	3	10
	Ratio	0.2	0.3	1
S6-S0	Total No.	21		
	Covered	1	1	21
	Ratio	0.048	0.048	1

Appendix C

Interview Sheet

Having that all required information was given to the interviewee, the interview questions are structured as follows:

Q1- Can you please rank the pair-wise comparison preference of the seven testing sets with respect to application importance, complexity and development stage using the following scale of ranking?

Pairwise Comparison scale for AHP Preferences	
Numerical Rating	Verbal Judgements of Preferences
1	C is equally preferable to D
2	C is equally to moderately preferable to D
3	C is moderately preferable to D
4	C is moderately to strongly preferable to D
5	C is strongly preferable to D
6	C is strongly to very strongly preferable to D
7	C is very strongly preferable to D
8	C is very strongly to exceptionally preferable to D
9	C is exceptionally preferable to D

Importance	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1						
OB		1					
IB			1				
B+OB				1			
B+IB					1		
OB+IB						1	
B+OB+IB							1

Complexity	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1						
OB		1					
IB			1				
B+OB				1			
B+IB					1		
OB+IB						1	
B+OB+IB							1

Development Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1						
OB		1					
IB			1				
B+OB				1			
B+IB					1		
OB+IB						1	
B+OB+IB							1

Q2- Can you please have a look at testing Scenario One and Two, and rank the pair-wise comparison preference of the criteria and sub-criteria using the same scale of ranking? The criteria definitions are provided.

	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1		
Test Cost		1	
Application Domain			1

	Fault Coverage	Clock Region Coverage
Fault Coverage	1	
Clock region Coverage		1

	Test Traces Length	Test Execution Time
Test Traces Length	1	
Test Execution Time		1

	Importance	Complexity	Development Stage
Importance	1		
Complexity		1	
Development stage			1

Q3- Can you please order (1-7) the test sets according to the best suitability to the Scenario One?

B OB IB B+OB B+IB OB+IB B+OB+IB

Appendix D

AHP Matrices

The complete set of AHP pair-wise comparison matrices that have been filled by five testing experts (E1...E5) is presented in this Appendix with their normalised form and consistency ratio.

Pair-wise Comparison Matrix of alternatives with respect to Importance (E1)							
Importance	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1	1	0.5	0.25	0.2	0.11
OB	1	1	1	0.5	0.25	0.2	0.11
IB	1	1	1	0.5	0.25	0.2	0.11
B+OB	2	2	2	1	0.33	0.25	0.14
B+IB	4	4	4	3	1	0.25	0.14
OB+IB	5	5	5	4	4	1	0.14
B+OB+IB	9	9	9	7	7	7	1

Consistency Ratio =0.0582

Normalized Matrix of alternatives with respect to Importance (E1)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.034515
OB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.034515
IB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.034515
B+OB	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.056999
B+IB	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.109731
OB+IB	0.19	0.19	0.19	0.19	0.2	0.2	0.19	0.194743
B+OB+IB	0.54	0.54	0.54	0.53	0.53	0.53	0.54	0.534981

Pair-wise Comparison Matrix of alternatives with respect to Complexity (E1)							
Complexity	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1	1	0.5	0.25	0.2	0.11
OB	1	1	1	0.5	0.25	0.2	0.11
IB	1	1	1	0.5	0.25	0.2	0.11
B+OB	2	2	2	1	0.33	0.25	0.14
B+IB	4	4	4	3	1	0.25	0.14
OB+IB	5	5	5	4	4	1	0.14
B+OB+IB	9	9	9	7	7	7	1

Consistency Ratio =0.0582

Normalized Matrix of alternatives with respect to Complexity (E1)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.034515
OB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.034515
IB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.034515
B+OB	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.056999
B+IB	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.109731
OB+IB	0.19	0.19	0.19	0.19	0.2	0.2	0.19	0.194743
B+OB+IB	0.54	0.54	0.54	0.53	0.53	0.53	0.54	0.534981

Pair-wise Comparison Matrix of alternatives with respect to Development Stage (E1)							
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1	1	0.5	0.25	0.2	0.11
OB	1	1	1	0.5	0.25	0.2	0.11
IB	1	1	1	0.5	0.25	0.2	0.11
B+OB	2	2	2	1	0.33	0.25	0.14
B+IB	4	4	4	3	1	0.25	0.14
OB+IB	5	5	5	4	4	1	0.14
B+OB+IB	9	9	9	7	7	7	1

Consistency Ratio =0.0582

Normalized Matrix of alternatives with respect to Development Stage								
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.034515
OB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.034515
IB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.034515
B+OB	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.056999
B+IB	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.109731
OB+IB	0.19	0.19	0.19	0.19	0.2	0.2	0.19	0.194743
B+OB+IB	0.54	0.54	0.54	0.53	0.53	0.53	0.54	0.534981

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E1, Scenario 1)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	0.33	5
Test Cost	3	1	9
Development Stage	0.2	0.11	1

Consistency Ratio =0.0245

Normalized Matrix of the main criteria with respect to Goal (E1, Scenario 1)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.27	0.27	0.27	0.265381
Test Cost	0.67	0.67	0.67	0.671695
Application Domain	0.06	0.06	0.06	0.062924

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Adequacy (E1, Scenario1)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	9
Clock Region Coverage	0.11	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy Scenario1			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.9	0.9	0.900045
Clock Region Coverage	0.1	0.1	0.099955

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E1, Scenario1)		
	Test Traces Length	Test Execution time
Test Traces Length	1	0.11
Test Execution time	9	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost Scenario1			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.1	0.1	0.099955
Test Execution time	0.9	0.9	0.900045

Pair-wise Comparison Matrix of the sub-criteria with respect to Application domain (E1, Scenario 1)			
	Importance	Complexity	Development Stage
Importance	1	0.33	0.11
Complexity	3	1	0.17
Development Stage	9	6	1

Consistency Ratio =0.0461

Normalized Matrix of the sub-criteria with respect to Application domain (E1, Scenario 1)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.07	0.07	0.07	0.067879
Complexity	0.16	0.16	0.16	0.161814
Development Stage	0.77	0.77	0.77	0.770307

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E1, Scenario 2)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	8	9
Test Cost	0.12	1	1
Development Stage	0.11	1	1

Consistency Ratio =0.0011

Normalized Matrix of the main criteria with respect to Goal (Scenario 2)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.81	0.81	0.81	0.80925
Test Cost	0.1	0.1	0.1	0.097263
Application Domain	0.09	0.09	0.09	0.093488

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Adequacy (E1, Scenario2)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	7
Clock Region Coverage	0.14	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy Scenario2			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.87	0.87	0.874945
Clock Region Coverage	0.13	0.13	0.125055

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E1, Scenario2)		
	Test Traces Length	Test Execution time
Test Traces Length	1	1
Test Execution time	1	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost (E1, Scenario2)			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.5	0.5	0.5
Test Execution time	0.5	0.5	0.5

Pair-wise Comparison Matrix of the sub-criteria with respect to Application domain (E1, Scenario 2)			
	Importance	Complexity	Development Stage
Importance	1	9	1
Complexity	0.11	1	0.14
Development Stage	1	7	1

Consistency Ratio =0.0061

Normalized Matrix of the sub-criteria with respect to Application domain (E1, Scenario 2)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.49	0.49	0.49	0.490084
Complexity	0.06	0.06	0.06	0.059215
Development Stage	0.45	0.45	0.45	0.4507

Pair-wise Comparison Matrix of alternatives with respect to Importance (E2)							
Importance	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	0.25	1	0.17	0.17	0.17	0.11
OB	4	1	1	0.11	0.11	0.11	0.11
IB	1	1	1	0.11	0.11	0.11	0.11
B+OB	6	9	9	1	1	1	1
B+IB	6	9	9	1	1	1	1
OB+IB	6	9	9	1	1	1	1
B+OB+IB	9	9	9	1	1	1	1

Consistency Ratio =0.0413

Normalized Matrix of alternatives with respect to Importance (E2)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.027478
OB	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.037394
IB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.02617
B+OB	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.224433
B+IB	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.224433
OB+IB	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.224433
B+OB+IB	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.235657

Pair-wise Comparison Matrix of alternatives with respect to Complexity (E2)							
Complexity	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1	1	0.17	0.14	0.12	0.11
OB	1	1	1	0.17	0.14	0.12	0.11
IB	1	1	1	0.17	0.14	0.12	0.11
B+OB	6	6	6	1	0.5	1	0.2
B+IB	7	7	7	2	1	1	0.2
OB+IB	8	8	8	1	1	1	0.2
B+OB+IB	9	9	9	5	5	5	1

Consistency Ratio =0.0494

Normalized Matrix of alternatives with respect to Complexity (E2)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.026559
OB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.026559
IB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.026559
B+OB	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.127301
B+IB	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.166505
OB+IB	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.160092
B+OB+IB	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.466425

Pair-wise Comparison Matrix of alternatives with respect to Development Stage (E2)							
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1	1	0.17	0.14	0.12	0.11
OB	1	1	1	0.17	0.14	0.12	0.11
IB	1	1	1	0.17	0.14	0.12	0.11
B+OB	6	6	6	1	0.5	1	0.2
B+IB	7	7	7	2	1	1	0.2
OB+IB	8	8	8	1	1	1	0.2
B+OB+IB	9	9	9	5	5	5	1

Consistency Ratio =0.0494

Normalized Matrix of alternatives with respect to Development Stage (E2)								
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.026559
OB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.026559
IB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.026559
B+OB	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.127301
B+IB	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.166505
OB+IB	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.160092
B+OB+IB	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.466425

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E2, Scenario1)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	0.25	3
Test Cost	4	1	6
Development Stage	0.33	0.17	1

Consistency Ratio =0.0467

Normalized Matrix of the main criteria with respect to Goal (E2, Scenario1)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.22	0.22	0.22	0.217641
Test Cost	0.69	0.69	0.69	0.690909
Application Domain	0.09	0.09	0.09	0.09145

Pair-wise Comparison Matrix of the sub-criteria with respect to Test adequacy (E2, Scenario1)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	0.2
Clock Region Coverage	5	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy Scenario1			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.17	0.17	0.166667
Clock Region Coverage	0.83	0.83	0.833333

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E2, Scenario1)		
	Test Traces Length	Test Execution time
Test Traces Length	1	0.333
Test Execution time	3	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost Scenario1			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.25	0.25	0.249906
Test Execution time	0.75	0.75	0.750094

Pair-wise Comparison Matrix of the sub-criteria with respect to Application domain (E2, Scenario 1)			
	Importance	Complexity	Development Stage
Importance	1	1	4
Complexity	1	1	6
Development Stage	0.25	0.17	1

Consistency Ratio =0.0163

Normalized Matrix of the sub-criteria with respect to Application domain (E2, Scenario 1)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.42	0.42	0.42	0.423137
Complexity	0.48	0.48	0.48	0.484396
Development Stage	0.09	0.09	0.09	0.092467

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E2, Scenario 2)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	1	0.25
Test Cost	1	1	0.33
Development Stage	4	3	1

Consistency Ratio =0.0076

Normalized Matrix of the main criteria with respect to Goal (E2, Scenario 2)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.17	0.17	0.17	0.174381
Test Cost	0.19	0.19	0.19	0.191863
Application Domain	0.63	0.63	0.63	0.633756

Pair-wise Comparison Matrix of the sub-criteria with respect to Test adequacy (E2, Scenario2)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	0.5
Clock Region Coverage	2	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy (E2, Scenario2)			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.33	0.33	0.333333
Clock Region Coverage	0.67	0.67	0.666667

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E2, Scenario2)		
	Test Traces Length	Test Execution time
Test Traces Length	1	0.17
Test Execution time	6	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost (E2, Scenario2)			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.14	0.14	0.14298
Test Execution time	0.86	0.86	0.85702

Pair-wise Comparison Matrix of the sub-criteria with respect to Application domain (E2, Scenario 2)			
	Importance	Complexity	Development Stage
Importance	1	0.11	0.5
Complexity	9	1	7
Development Stage	2	0.14	1

Consistency Ratio =0.0186

Normalized Matrix of the sub-criteria with respect to Application domain (E2, Scenario 2)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.08	0.08	0.08	0.075989
Complexity	0.79	0.79	0.79	0.792759
Development Stage	0.13	0.13	0.13	0.131252

Pair-wise Comparison Matrix of alternatives with respect to Importance (E3)							
Importance	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	3	4	0.33	0.33	0.33	0.2
OB	0.33	1	2	0.25	0.25	0.25	0.17
IB	0.25	0.5	1	0.14	0.14	0.14	0.11
B+OB	3	4	7	1	2	2	0.5
B+IB	3	4	7	0.5	1	2	0.25
OB+IB	3	4	7	0.5	0.5	1	0.33
B+OB+IB	5	6	9	2	4	3	1

Consistency Ratio =0.0395

Normalized Matrix of alternatives with respect to Importance (E3)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.072293
OB	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.040749
IB	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02379
B+OB	0.21	0.21	0.21	0.21	0.21	0.21	0.21	0.208153
B+IB	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.159781
OB+IB	0.13	0.13	0.13	0.13	0.13	0.14	0.13	0.134572
B+OB+IB	0.36	0.36	0.36	0.36	0.36	0.36	0.36	0.360663

Pair-wise Comparison Matrix of alternatives with respect to Complexity (E3)							
Complexity	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1	0.5	3	3	3	7
OB	1	1	0.5	3	3	3	7
IB	2	2	1	4	4	4	8
B+OB	0.33	0.33	0.25	1	0.5	0.5	3
B+IB	0.33	0.33	0.25	2	1	0.5	3
OB+IB	0.33	0.33	0.25	2	2	1	4
B+OB+IB	0.14	0.14	0.12	0.33	0.33	0.25	1

Consistency Ratio =0.0226

Normalized Matrix of alternatives with respect to Complexity (E3)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.21	0.21	0.21	0.21	0.21	0.21	0.21	0.20597
OB	0.21	0.21	0.21	0.21	0.21	0.21	0.21	0.20597
IB	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.322725
B+OB	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.062752
B+IB	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.076837
OB+IB	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.098205
B+OB+IB	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.027541

Pair-wise Comparison Matrix of alternatives with respect to Development Stage (E3)							
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	1	0.5	1	1	1	1
OB	1	1	0.5	1	1	1	1
IB	2	2	1	2	2	2	2
B+OB	1	1	0.5	1	1	1	1
B+IB	1	1	0.5	1	1	1	1
OB+IB	1	1	0.5	1	1	1	1
B+OB+IB	1	1	0.5	1	1	1	1

Consistency Ratio =0.00

Normalized Matrix of alternatives with respect to Development Stage (E3)								
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.125
OB	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.125
IB	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
B+OB	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.125
B+IB	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.125
OB+IB	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.125
B+OB+IB	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.125

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E3, Scenario1)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	0.33	0.5
Test Cost	3	1	2
Development Stage	2	0.5	1

Consistency Ratio =0.0076

Normalized Matrix of the main criteria with respect to Goal (E3, Scenario 1)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.16	0.16	0.16	0.163374
Test Cost	0.54	0.54	0.54	0.539651
Application Domain	0.3	0.3	0.3	0.296975

Pair-wise Comparison Matrix of the sub-criteria with respect to Test adequacy (E3, Scenario1)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	0.5
Clock Region Coverage	2	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy (E3, Scenario1)			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.33	0.33	0.333333
Clock Region Coverage	0.67	0.67	0.666667

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E3, Scenario1)		
	Test Traces Length	Test Execution time
Test Traces Length	1	0.5
Test Execution time	2	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost (E3, Scenario1)			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.33	0.33	0.333333
Test Execution time	0.67	0.67	0.666667

Pair-wise Comparison Matrix of the sub-criteria with respect to Application Domain (E3, Scenario 1)			
	Importance	Complexity	Development Stage
Importance	1	4	2
Complexity	0.25	1	0.5
Development Stage	0.5	2	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Application domain (E3, Scenario 1)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.57	0.57	0.57	0.571429
Complexity	0.14	0.14	0.14	0.142857
Development Stage	0.29	0.29	0.29	0.285714

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E3, Scenario 2)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	5	4
Test Cost	0.2	1	0.5
Development Stage	0.25	2	1

Consistency Ratio =0.0212

Normalized Matrix of the main criteria with respect to Goal (E3, Scenario 2)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.68	0.68	0.68	0.68334
Test Cost	0.12	0.12	0.12	0.11685
Application Domain	0.2	0.2	0.2	0.19981

Pair-wise Comparison Matrix of the sub-criteria with respect to Test adequacy (E3, Scenario2)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	1
Clock Region Coverage	1	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy (E3, Scenario2)			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.5	0.5	0.5
Clock Region Coverage	0.5	0.5	0.5

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E3, Scenario2)		
	Test Traces Length	Test Execution time
Test Traces Length	1	2
Test Execution time	0.5	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost (E3, Scenario2)			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.67	0.67	0.66667
Test Execution time	0.33	0.33	0.33333

Pair-wise Comparison Matrix of the sub-criteria with respect to Application domain (E3, Scenario 2)			
	Importance	Complexity	Development Stage
Importance	1	4	3
Complexity	0.25	1	0.5
Development Stage	0.33	2	1

Consistency Ratio =0.0155

Normalized Matrix of the sub-criteria with respect to Application domain (E3, Scenario 2)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.63	0.63	0.63	0.625052
Complexity	0.14	0.14	0.14	0.136512
Development Stage	0.24	0.24	0.24	0.238437

Pair-wise Comparison Matrix of alternatives with respect to Importance (E4)							
Importance	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	4	2	0.33	0.25	0.5	0.17
OB	0.25	1	0.33	0.17	0.14	0.2	0.11
IB	0.5	3	1	0.25	0.2	0.33	0.14
B+OB	3	6	4	1	0.5	2	0.25
B+IB	4	7	5	2	1	3	0.33
OB+IB	2	5	3	0.5	0.33	1	0.2
B+OB+IB	6	9	7	4	3	5	1

Consistency Ratio =0.0368

Normalized Matrix of alternatives with respect to Importance (E4)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.063724
OB	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.023876
IB	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.043459
B+OB	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.144631
B+IB	0.21	0.21	0.21	0.22	0.22	0.22	0.22	0.214967
OB+IB	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.095876
B+OB+IB	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.413467

Pair-wise Comparison Matrix of alternatives with respect to Complexity (E4)							
Complexity	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	4	2	0.33	0.25	0.5	0.17
OB	0.25	1	0.33	0.17	0.14	0.2	0.11
IB	0.5	3	1	0.25	0.2	0.33	0.14
B+OB	3	6	4	1	0.5	2	0.25
B+IB	4	7	5	2	1	3	0.33
OB+IB	2	5	3	0.5	0.33	1	0.2
B+OB+IB	6	9	7	4	3	5	1

Consistency Ratio =0.0368

Normalized Matrix of alternatives with respect to Complexity (E4)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.063724
OB	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.023876
IB	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.043459
B+OB	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.144631
B+IB	0.21	0.21	0.21	0.22	0.22	0.22	0.22	0.214967
OB+IB	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.095876
B+OB+IB	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.413467

Pair-wise Comparison Matrix of alternatives with respect to Development Stage (E4)							
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	4	2	0.33	0.25	0.5	0.17
OB	0.25	1	0.33	0.17	0.14	0.2	0.11
IB	0.5	3	1	0.25	0.2	0.33	0.14
B+OB	3	6	4	1	0.5	2	0.25
B+IB	4	7	5	2	1	3	0.33
OB+IB	2	5	3	0.5	0.33	1	0.2
B+OB+IB	6	9	7	4	3	5	1

Consistency Ratio =0.0368

Normalized Matrix of alternatives with respect to Development Stage (E4)								
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.063724
OB	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.023876
IB	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.043459
B+OB	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.144631
B+IB	0.21	0.21	0.21	0.22	0.22	0.22	0.22	0.214967
OB+IB	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.095876
B+OB+IB	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.413467

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E4, Scenario 1)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	1	0.33
Test Cost	1	1	0.5
Development Stage	3	2	1

Consistency Ratio =0.0155

Normalized Matrix of the main criteria with respect to Goal (E4, Scenario 1)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.21	0.21	0.21	0.209797
Test Cost	0.24	0.24	0.24	0.240229
Application Domain	0.55	0.55	0.55	0.549974

Pair-wise Comparison Matrix of the sub-criteria with respect to Test adequacy (E4, Scenario1)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	1
Clock Region Coverage	1	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy (E4, Scenario1)			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.5	0.5	0.5
Clock Region Coverage	0.5	0.5	0.5

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E4, Scenario1)		
	Test Traces Length	Test Execution time
Test Traces Length	1	1
Test Execution time	1	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost (E4, Scenario1)			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.5	0.5	0.5
Test Execution time	0.5	0.5	0.5

Pair-wise Comparison Matrix of the sub-criteria with respect to Application domain (E4, Scenario 1)			
	Importance	Complexity	Development Stage
Importance	1	1	0.33
Complexity	1	1	0.5
Development Stage	3	2	1

Consistency Ratio =0.0155

Normalized Matrix of the sub-criteria with respect to Application domain (E4, Scenario 1)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.21	0.21	0.21	0.209797
Complexity	0.24	0.24	0.24	0.240229
Development Stage	0.55	0.55	0.55	0.549974

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E4, Scenario 2)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	0.5	1
Test Cost	2	1	3
Development Stage	1	0.33	1

Consistency Ratio =0.0155

Normalized Matrix of the main criteria with respect to Goal (E4, Scenario 2)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.24	0.24	0.24	0.240229
Test Cost	0.55	0.55	0.55	0.549974
Application Domain	0.21	0.21	0.21	0.209797

Pair-wise Comparison Matrix of the sub-criteria with respect to Test adequacy (E4, Scenario2)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	1
Clock Region Coverage	1	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy (E4, Scenario2)			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.5	0.5	0.5
Clock Region Coverage	0.5	0.5	0.5

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E4, Scenario2)		
	Test Traces Length	Test Execution time
Test Traces Length	1	1
Test Execution time	1	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost (E4, Scenario2)			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.5	0.5	0.5
Test Execution time	0.5	0.5	0.5

Pair-wise Comparison Matrix of the sub-criteria with respect to Application domain (E4, Scenario 2)			
	Importance	Complexity	Development Stage
Importance	1	0.33	1
Complexity	3	1	2
Development Stage	1	0.5	1

Consistency Ratio =0.0155

Normalized Matrix of the sub-criteria with respect to Application domain (E4, Scenario 2)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.21	0.21	0.21	0.209797
Complexity	0.55	0.55	0.55	0.549974
Development Stage	0.24	0.24	0.24	0.240229

Pair-wise Comparison Matrix of alternatives with respect to Importance (E5)							
Importance	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	0.5	0.33	0.33	0.25	0.2	0.17
OB	2	1	0.5	0.5	0.33	0.33	0.25
IB	3	2	1	2	0.5	0.33	0.25
B+OB	3	2	0.5	1	0.5	0.33	0.25
B+IB	4	3	2	2	1	3	0.33
OB+IB	5	3	3	3	0.33	1	0.5
B+OB+IB	6	4	4	4	3	2	1

Consistency Ratio =0.0501

Normalized Matrix of alternatives with respect to Importance (E5)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.036491
OB	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.058787
IB	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.099639
B+OB	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.081867
B+IB	0.21	0.21	0.21	0.21	0.21	0.21	0.21	0.207896
OB+IB	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.178103
B+OB+IB	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.337216

Pair-wise Comparison Matrix of alternatives with respect to Complexity (E5)							
Complexity	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	0.5	0.33	0.33	0.25	0.2	0.17
OB	2	1	0.5	0.5	0.33	0.33	0.25
IB	3	2	1	2	0.5	0.33	0.25
B+OB	3	2	0.5	1	0.5	0.33	0.25
B+IB	4	3	2	2	1	3	0.33
OB+IB	5	3	3	3	0.33	1	0.5
B+OB+IB	6	4	4	4	3	2	1

Consistency Ratio =0.0501

Normalized Matrix of alternatives with respect to Complexity (E5)								
	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.036491
OB	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.058787
IB	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.099639
B+OB	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.081867
B+IB	0.21	0.21	0.21	0.21	0.21	0.21	0.21	0.207896
OB+IB	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.178103
B+OB+IB	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.337216

Pair-wise Comparison Matrix of alternatives with respect to Development Stage (E5)							
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB
B	1	0.5	0.33	0.33	0.25	0.2	0.17
OB	2	1	0.5	0.5	0.33	0.33	0.25
IB	3	2	1	2	0.5	0.33	0.25
B+OB	3	2	0.5	1	0.5	0.33	0.25
B+IB	4	3	2	2	1	3	0.33
OB+IB	5	3	3	3	0.33	1	0.5
B+OB+IB	6	4	4	4	3	2	1

Consistency Ratio =0.0501

Normalized Matrix of alternatives with respect to Development Stage (E5)								
Stage	B	OB	IB	B+OB	B+IB	OB+IB	B+OB+IB	Eigen-Vector
B	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.036491
OB	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.058787
IB	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.099639
B+OB	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.081867
B+IB	0.21	0.21	0.21	0.21	0.21	0.21	0.21	0.207896
OB+IB	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.178103
B+OB+IB	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.337216

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E5, Scenario 1)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	5	7
Test Cost	0.2	1	2
Development Stage	0.14	0.5	1

Consistency Ratio =0.0125

Normalized Matrix of the main criteria with respect to Goal (E5, Scenario 1)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.74	0.74	0.74	0.739564
Test Cost	0.17	0.17	0.17	0.166591
Application Domain	0.09	0.09	0.09	0.093845

Pair-wise Comparison Matrix of the sub-criteria with respect to Test adequacy (E5, Scenario1)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	6
Clock Region Coverage	0.17	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy (E5, Scenario1)			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.86	0.86	0.85702
Clock Region Coverage	0.14	0.14	0.14298

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E5, Scenario1)		
	Test Traces Length	Test Execution time
Test Traces Length	1	0.17
Test Execution time	6	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost (E5, Scenario1)			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.14	0.14	0.14298
Test Execution time	0.86	0.86	0.85702

Pair-wise Comparison Matrix of the sub-criteria with respect to Application domain (E5, Scenario 1)			
	Importance	Complexity	Development Stage
Importance	1	3	4
Complexity	0.33	1	3
Development Stage	0.25	0.33	1

Consistency Ratio =0.0629

Normalized Matrix of the sub-criteria with respect to Application domain (E5, Scenario 1)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.61	0.61	0.61	0.614469
Complexity	0.27	0.27	0.27	0.268324
Development Stage	0.12	0.12	0.12	0.117206

Pair-wise Comparison Matrix of the main criteria with respect to Goal (E5, Scenario 2)			
	Test Adequacy	Test Cost	Application Domain
Test Adequacy	1	6	2
Test Cost	0.17	1	0.25
Development Stage	0.5	4	1

Consistency Ratio =0.0086

Normalized Matrix of the main criteria with respect to Goal (E5, Scenario 2)				
	Test Adequacy	Test Cost	Application Domain	Eigenvector
Test Adequacy	0.59	0.59	0.59	0.587583
Test Cost	0.09	0.09	0.09	0.089043
Application Domain	0.32	0.32	0.32	0.323374

Pair-wise Comparison Matrix of the sub-criteria with respect to Test adequacy (E5, Scenario2)		
	Fault Coverage	Clock Region Coverage
Fault Coverage	1	2
Clock Region Coverage	0.5	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test adequacy (E5, Scenario2)			
	Fault Coverage	Clock Region Coverage	Eigenvector
Fault Coverage	0.67	0.67	0.666667
Clock Region Coverage	0.33	0.33	0.333333

Pair-wise Comparison Matrix of the sub-criteria with respect to Test Cost (E5, Scenario2)		
	Test Traces Length	Test Execution time
Test Traces Length	1	0.17
Test Execution time	6	1

Consistency Ratio =0.00

Normalized Matrix of the sub-criteria with respect to Test Cost (E5, Scenario2)			
	Test Traces Length	Test Execution time	Eigenvector
Test Traces Length	0.14	0.14	0.14298
Test Execution time	0.86	0.86	0.85702

Pair-wise Comparison Matrix of the sub-criteria with respect to Application domain (E5, Scenario 2)			
	Importance	Complexity	Development Stage
Importance	1	3	4
Complexity	0.33	1	3
Development Stage	0.25	0.33	1

Consistency Ratio =0.0629

Normalized Matrix of the sub-criteria with respect to Application domain (E5, Scenario 2)				
	Importance	Complexity	Development Stage	Eigenvector
Importance	0.61	0.61	0.61	0.614469
Complexity	0.27	0.27	0.27	0.268324
Development Stage	0.12	0.12	0.12	0.117206