Domain Specialisation and Applications of Model-Based Testing

Percy Antonio Pari Salas

BEng (UCSM), MSc (USP)

A dissertation submitted in fulfilment of the requirements of the degree of Doctor of Philosophy for the School of Information Technology, Bond University.

March 2010

Typeset in LaTeX 2_ε

Statement of Originality

The work presented in this thesis is, to the best of my knowledge and belief, original, except where acknowledged in the text. I hereby declare that I have not submitted this material either in whole or in part, for a degree at this or any other university.

Percy Antonio Pari Salas

Date: March 31st, 2010

Submitted for examination: November 2009

Approved for the degree of Doctor of Philosophy: March 2010

Abstract

Software testing, one of the most important methods for quality assurance, has become too expensive and error prone for complex modern software systems. Test automation aims to reduce the costs of software testing and to improve its reliability. Despite advances in test automation, there are some domains for which automation seems to be difficult, for example, testing software to reveal the presence of security vulnerabilities, testing for conformance to security properties that traverse several functionalities of an application such as privacy policies, and testing asynchronous concurrent systems.

Although there are research works that aim to solve the problems of test automation for these domains, there is still a gap between the practice and the state of the art. These works describe specific approaches that deal with particular problems, generally under restricted conditions. Nevertheless, individually, they have not made noticeable impact on the practice in test automation for these domains. Therefore, there is a need for an integrated framework that binds specific approaches together in order to provide more complete solutions. It is also important for this framework to show how current test automation efforts, tools and frameworks, can be reused. This thesis addresses this need by describing a general model-based testing framework and its specialisation for the testing domains of security vulnerabilities, privacy policies and asynchronous systems.

The main characteristic of the general framework resides in the separation between behavioural (control) and data generation specifications. This framework is defined on the basis of labelled transition systems and context free grammars. Labelled transition systems allow behavioural models to be kept simple and tractable while extended context free grammars allow for the generation of data values that later will be used in the execution of test cases. The extended grammars in the data generation models contain a representation of the global state of the system, which allows for example, the history of the execution of test cases to influence the generation of subsequent data values.

Besides the general pattern described in the behavioural and data generation models, each specialised testing domain requires models that represent particular characteristics of the

system and the testing objectives for that domain. Vulnerability testing requires a model that describes the properties of a system that make it vulnerable and another one that describes what are considered to be the malicious intentions of an attacker. Privacy policies testing, requires the addition of a model that describes the conditions under which the execution of a defined operation is restricted or permitted. An important characteristic of the privacy policies described in this thesis is that they include the concept of obligations, this is, actions that require to be performed before the execution of the test case is considered successful. This framework considers test cases that fulfil bounded obligations.

In testing asynchronous systems, this thesis focuses in a defined subclass of systems in which actions can be partitioned into controllable and observable actions where execution of controllable actions is decided by the testing framework and observable actions designate the response of the system to the controllable stimuli. Different from other approaches for asynchronous systems, this thesis uses sets instead of queues to keep tracking of expected observable responses. This allows the present approach to deal with imperfect communication channels and with delays and loss of information, where the order of the observations is not important.

The practical applicability of the approaches presented in this thesis is demonstrated in several case studies from various domain applications, namely web-based applications, financial exchange protocols and operating systems. Particularly, the case study on operating systems demonstrates the integration of the general approach with an existing testing framework. This case study describes advantages, disadvantages and trade-offs of such integration.

Acknowledgements

I would like to start by acknowledging and thanking Dr. Padmanabhan Krishnan, my supervisor, for his thorough guidance, mentoring, patience and support throughout the development of this thesis.

I would also like to thank KJRoss and Associates and Smart Testing Technologies for funding my research. Particular mention deserves Dr. Kelvin Ross for many useful discussions and suggestions for the research project, and mainly for believing in me.

Many thanks go also to Bond University for providing the necessary support to undertake many research tasks, especially for supporting my attendance to various conferences and providing economic support during the writing of this thesis.

A general thank you goes to my colleagues and friends within the School of Information Technology and the Business Faculty at Bond University for making my time in the school a mostly enjoyable experience. However, particular thanks go to: Shane Bracher and James Larkin for countless chats, discussions and games in our shared office, for introducing me to the school, to bridge and to cricket; to Adrian Gepp for an enjoyable time sharing the same house, for many useful discussions and for proof reading some parts of this thesis; to Emma Chávez and Pedro Gómez for their unconditional friendship, the many chats in Spanish and for proof reading the early drafts of some chapters of this thesis.

Last but not least, I would like to thank my family, especially my parents, Nilda and Alfonso (RIP), for everything, and my wife Rita and my daughter Sarah, my two treasures on Earth, for so many wonderful moments, and for their unconditional love, support and patience.

Contents

1 Introduction **1**

1.1 Motivation and statement of problems 1

1.1.1 Software testing 2

1.1.2 Test automation 2

1.1.3 Model-based testing 4

1.1.4 Problems of current automation 5

1.2 Aims and research question 7

1.3 Main contributions 8

1.4 Thesis outline 10

2 Literature review **12**

2.1 Preliminaries 12

2.1.1 Labelled transition systems 12

2.1.2 Input-Output transition systems 14

2.2 Software test automation 17

2.2.1 Action-words and keywords 19

2.3 Model-based Testing 21

2.3.1 Process 22

2.3.2 Models 24

2.3.3 Test case generation 26

2.3.4 Test case execution 29

2.4 Software security 30

2.4.1 Privacy policies . . . 31
2.4.2 Security vulnerabilities . . . 34
2.5 Concurrent asynchronous systems . . . 40
2.6 Testing tools . . . 43
2.6.1 TGV/CADP . . . 43
2.6.2 TorX/CADP . . . 44
2.6.3 Microsoft Spec Explorer . . . 45
2.6.4 SmartMBT . . . 46

3 Test automation framework **48**
3.1 General framework . . . 48
3.2 Models . . . 52
3.2.1 Behavioural models . . . 52
3.2.2 Data generation models . . . 53
3.2.3 Composing behavioural and data models . . . 55
3.3 Test generation . . . 56
3.4 Test execution . . . 58
3.5 Concluding remarks . . . 58

4 Framework specialisation for different testing domains **60**
4.1 Vulnerability testing . . . 60
4.1.1 Modelling for testing vulnerabilities . . . 60
4.1.2 Generating tests to reveal vulnerabilities . . . 62
4.2 Privacy policies testing . . . 66
4.2.1 Modelling for testing preservation of privacy . . . 66
4.2.2 Generating tests for a privacy policy . . . 71
4.3 Asynchronous systems testing . . . 75
4.3.1 A different kind of system: Example . . . 75
4.3.2 Modelling asynchronous systems . . . 77
4.3.3 Testing asynchronous systems . . . 79

4.4 Concluding remarks 83

5 Case studies **85**

5.1 Overview 85

5.2 Device drivers testing 86

5.2.1 The system under test 87

5.2.2 Test generation with models based in the SDTS suite 91

5.2.3 Test generation with fine-grained models 97

5.2.4 Discussion 103

5.3 Vulnerability testing 105

5.3.1 A web based login function 106

5.3.2 The WebGoat application 112

5.3.3 Discussion 116

5.4 Privacy testing 119

5.4.1 Web browsing privacy: the same origin policy 119

5.4.2 Children's Online Privacy Protection Act policy 131

5.4.3 Discussion 137

5.5 Asynchronous systems testing 138

5.5.1 The FIX Protocol 139

5.5.2 Modelling the FIX Protocol 140

5.5.3 Test generation and execution 142

5.5.4 Discussion 147

5.6 Lessons learned 149

6 Conclusions and Future Research Issues **153**

6.1 Thesis summary 153

6.2 Answer to the initial research question 156

6.3 Future research 157

A Acronyms **159**

List of Figures

2.1 LTS model of a vending machine 13
2.2 Demonic completed IOTS model of a vending machine 16
2.3 Angelic completed IOTS model of a vending machine 16
2.4 A general architecture for a keyword-driven testing framework 19
2.5 Model subjects of model-based testing (source: [122]) 26
2.6 Intended vs. implemented software behaviour (source: [115]). 35

3.1 Relationship between models and SUT 51

4.1 Faulty contexts for test case generation 64
4.2 A simple system model. 74
4.3 Simplified model of a file exchanging system 76

5.1 Structure of test suites in the TET framework 89
5.2 Test scenario specification 90
5.3 Test purpose code 91
5.4 Library function *label_smi* code 92
5.5 Library function *build_fs* code 93
5.6 Model of the SDTS suite 95
5.7 Model representing a test that creates and mounts a partition 99
5.8 Extract of the state chart for the generated test sequence. 102
5.9 Behavioural model of a login functionality 106
5.10 Specification of the implemented application 107

5.11 An *authentication* attack 108
5.12 An *information-disclosure* attack 109
5.13 Behavioural model for the WebGoat application. 113
5.14 Attacker model for the WebGoat application. 113
5.15 AttackURL action of the attacker's model 115
5.16 Credit card numbers disclosed by an SQL injection attack 117
5.17 Behavioural model of a web browser 122
5.18 Implementation details of caching and history features 123
5.19 URL instances and their links 126
5.20 Data generation grammars for same origin policy testing 128
5.21 Model of a web application that subscribes users for a service. 134
5.22 Data generation grammars for COPPA policy testing 136
5.23 Valid sequences of actions in the FIX protocol 140
5.24 Simplified model of the FIX protocol 142
5.25 Linking to the SUT 144
5.26 The extended SmartMBT 145
5.27 Architecture of the implemented model-based framework 150

List of Tables

2.1 Test automation frameworks 18

4.1 Example of privacy rules 74

5.1 States of a model for the SDTS suite 94

5.2 Execution verdicts for test cases in SDTS suite 97

5.3 States in the model of the HBA driver 100

5.4 New model actions and their corresponding script code 103

5.5 Attacks that result in unauthorised authentication 111

5.6 Attacks that force an SQL engine error 112

5.7 Rules for the same-origin policy. 121

5.8 Rules for the caching feature in the same origin policy. 125

5.9 Rules for the history feature in the same origin policy. 125

5.10 Sequences of actions for testing the caching feature 129

5.11 Test cases for the history rule 129

5.12 Sequences of actions for testing the history feature 130

5.13 Test cases for the history rule 131

5.14 Verdicts from the execution of test cases for the same origin policy 132

5.15 Privacy rules in the COPPA policy 133

5.16 Rule generated from the difference of rules $rule2 - rule1$ in Table 5.15 137

5.17 Generated strings for COPPA policy 137

5.18 Evolution of the state of an asynchronous testing process 146