Experiences of System-Level Model-Based GUI Testing of an Android Application

Tommi Takala, Mika Katara
Tampere University of Technology,
tommi.takala@tut.fi

Julian Harty
eBay

Special thanks to former TEMA team at Tampere University of Technology: Henri Heiskanen, Antti Jääskeläinen, Lotta Liikkanen, Mika Maunumaa, Mika Mäenpää, Antti Nieminen, and Heikki Virtanen

Contents

Introduction

Model-based GUI testing – Lots of study, few real-life reported experiences

Keyword-based test automation for the Android Emulator

- System wide GUI testing
- Integration to an automatic test generation system

All presented tools and models available as open source

Actual data presented and results can be openly discussed

GUI Test Automation

Problems one needs to solve:

1. Verifying the GUI
   - Bitmap comparison (maintenance overkill!)
   - Optical Character Recognition (accuracy problems)
   - Application Programming Interface (seldom available)
2. Emulating user input
   - Through an API
   - Mechanical robots

Figure: Optofidelity Ltd.

GUI Test Automation in Android

- Instrumentation framework
  - Allows running applications in an instrumentation mode with a test application
  - Provides access to the contents of the application
  - Test cases are written with JUnit
  - Requires application source code
  - No system wide access => test one application at a time
- Robotium
  - Open source project that builds on Instrumentation
  - Can be used to test an APK packaged application => no source code required
  - Still no support for system wide automation

GUI Test Automation in Android Cont.

Window Service (hierarchy viewer)

Lists the GUI contents as a tree (widgets, their properties and layout)

Used in the hierarchy viewer application

Problems:

- Not all properties are listed
- Slow!
- Only available in the emulator and hardware unlocked devices without security features.

Monkey

Pseudo-random monkey testing tool

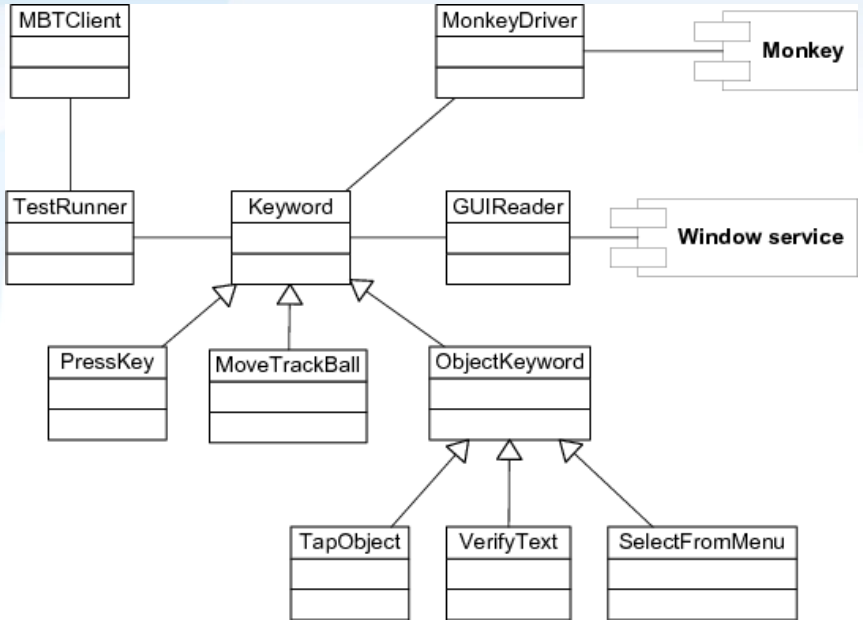
Network interface for receiving GUI events from clients

No GUI verification

Keyword-driven testing

- Keyword: an abstraction that describes user actions or GUI state verifications:
  - Press key <key>
  - Tap screen <coordinate>
  - Drag <coordinate1> => <coordinate2>
  - Search text from the screen <text>
- Separates test automation logic from the test cases
  - Improved maintainability
  - Writing test cases is simple => no need for programming skills

Keyword-driven testing in Android

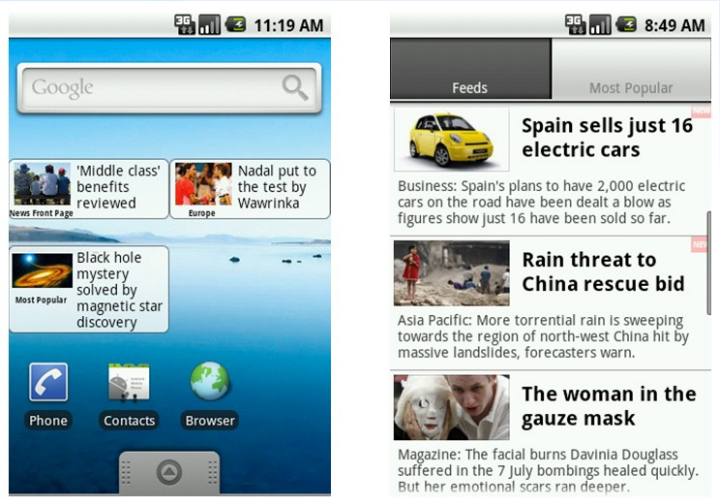


- Monkey network interface to enable user input
- Window service for verifying GUI contents
- Integrates to an MBT tool using a socket interface
- Keywords can also be executed interactively from prompt or from a file
- Implemented with Python
- Extensible: Add new keywords to the object hierarchy

Case Study: BBC News Widget

- Goal: to trial automatic test generation on Android.
  - How to model
  - Amount of work needed
  - What kind of tests can be generated
  - Ability to find bugs
- TEMA toolset was used as a model-based testing tool
- The generated tests were executed using the keyword-driven automation tool
- BBC News was selected as an application under test
  - Free
  - Widely used
  - Permission to discuss any findings

BBC News Widget



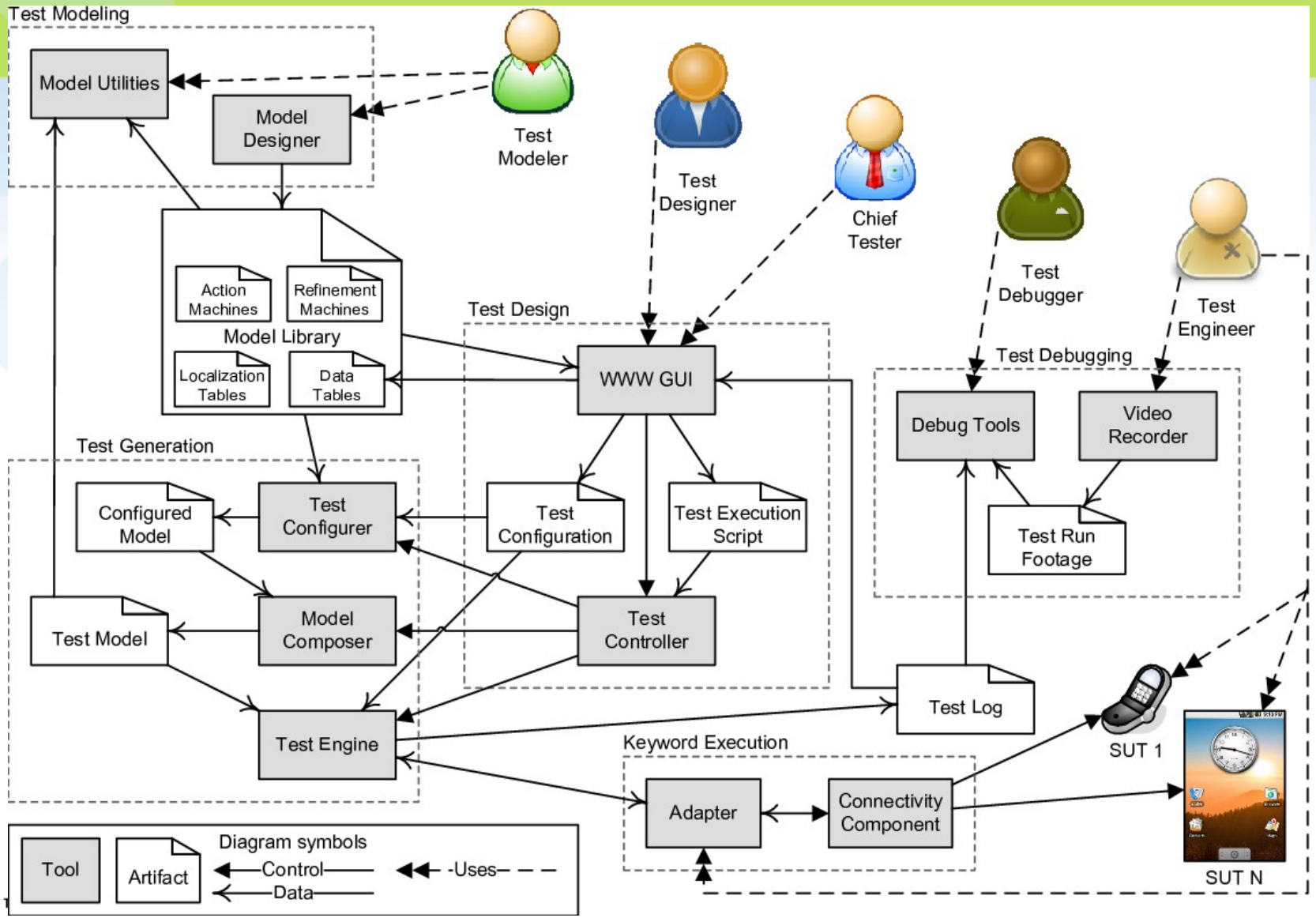
- RSS reader optimized for BBC news feeds
- Available from the Android Market
  - over million downloads
  - Free
- Had been previously manually tested

http://jimblackler.net/

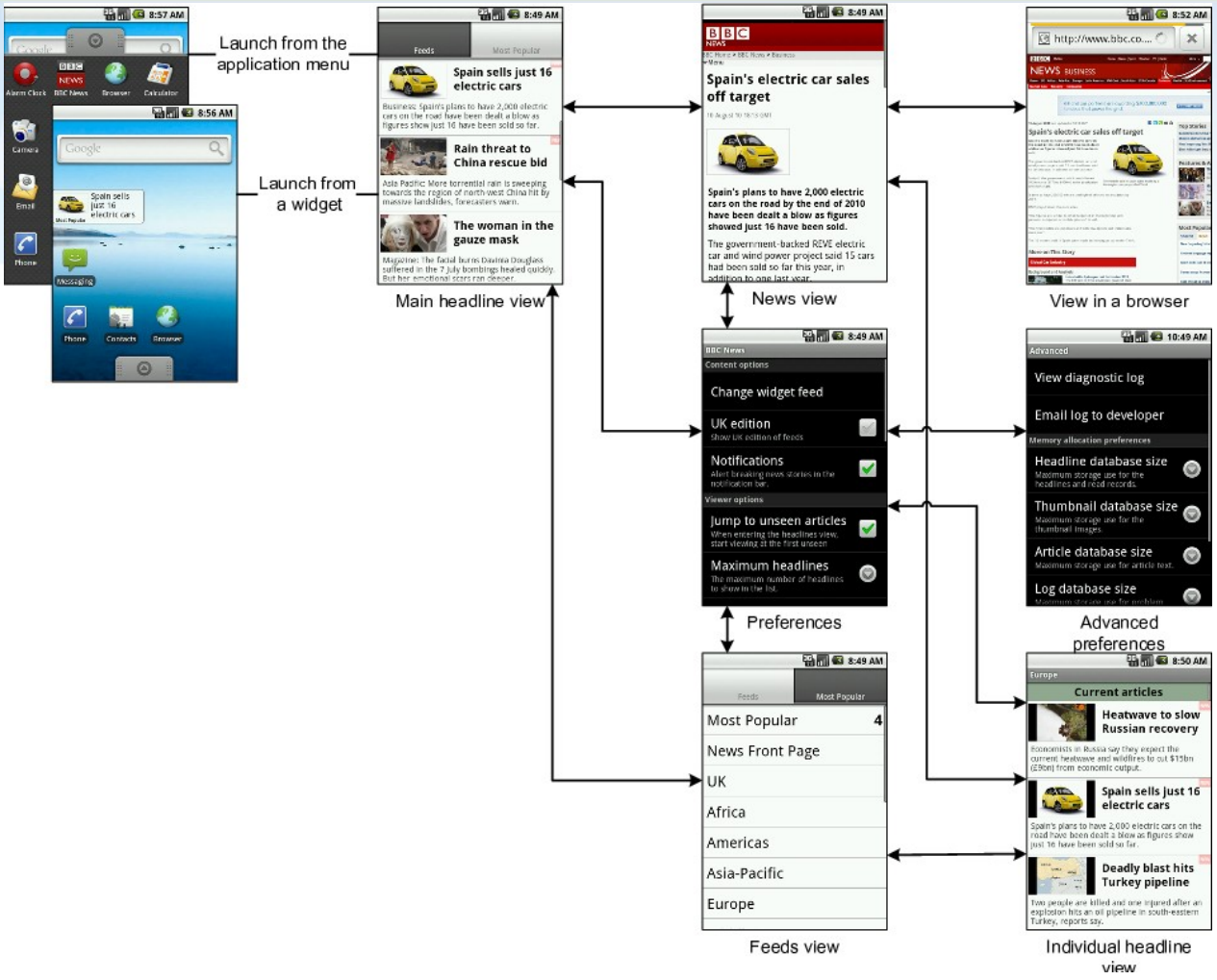
Model-Based Testing with TEMA

- Contains tools for modeling, test design, test generation, debugging, etc...
- State machine models (lsts) based on action words and keywords
  - Two-levels: action machines and refinement machines
    - High-level action machines are application specific (e.g. mms application)
    - Low-level refinement machines are SUT specific (e.g. Android, N900)
  - Two-level structure eases model maintenance and supports model reuse
  - Support for concurrency
- Online test generation (vs offline)
  - Enables testing nondeterministic systems
  - Enables long period testing
- One of the main benefits is hiding the complexity of MBT behind simple testing modes.
  - Modes include: random, use case, basic coverage, …
- Available from http://tema.cs.tut.fi

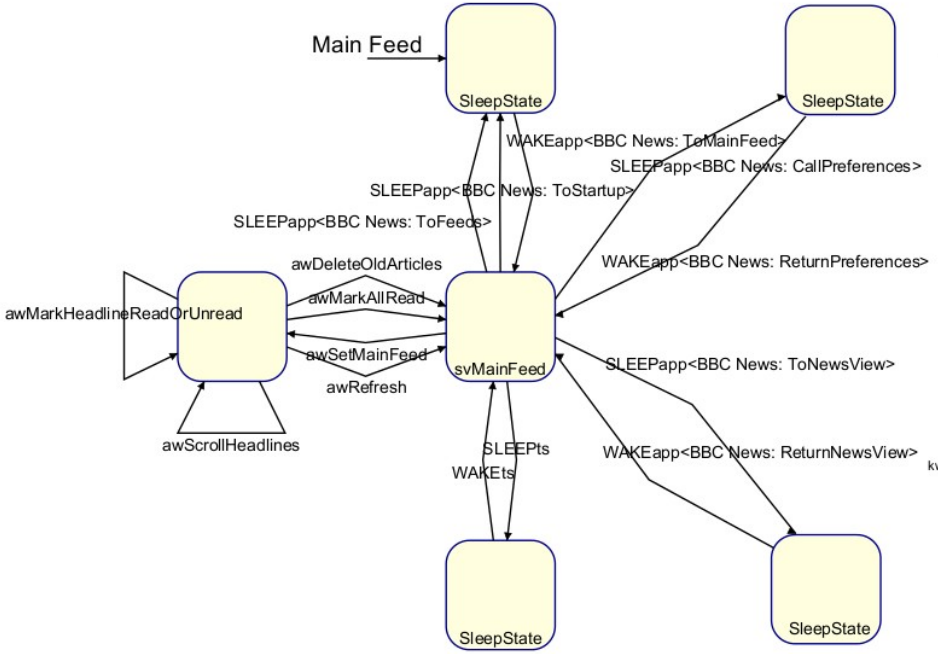
TEMA Architecture



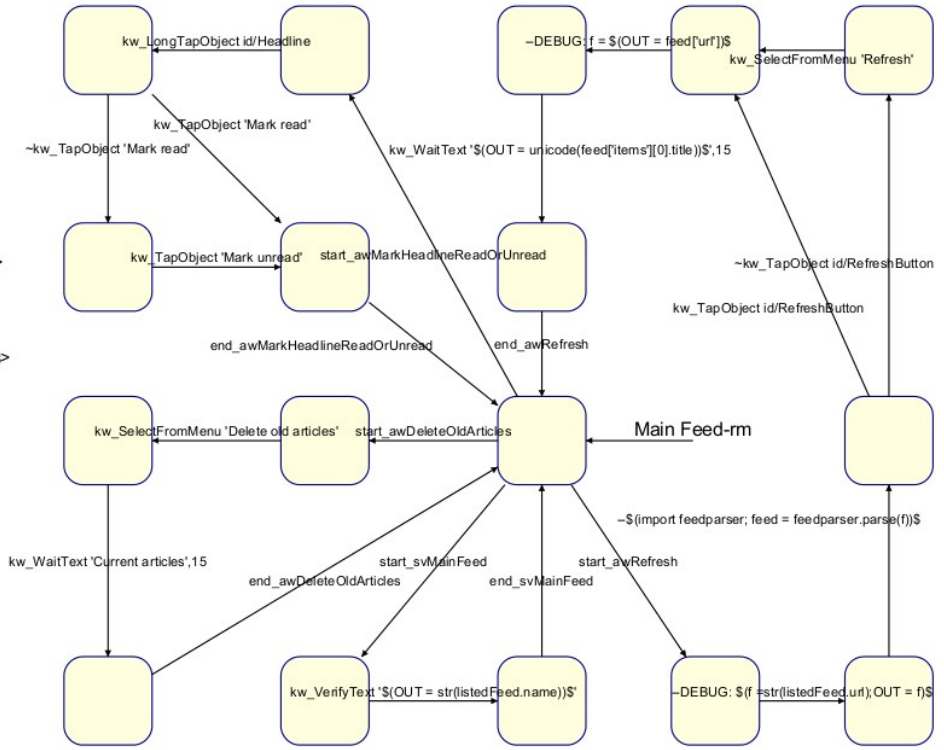
GUI Views – Starting point for modeling



Action machine vs Refinement machine



(a) Action machine: main headline view.



(b) Refinement machine: main headline view

Case summary

- Modeling the BBC News (16 state machines) took a few days time
  - With usability improvements in the modeling tool this could be made even faster
  - During the case, the application was updated thrice and the platform once (2.1 → 2.2)
    - Maintaining the models was fast – no need to update a huge set of test cases
- Random mode was used in the test generation
  - Setting up the test run takes practically no time at all
  - 240 separate test runs lasting over 115 hours in total
    - 27 000 action words – 50 000 keywords
  - The longest run lasted over three hours
  - Average duration was 30 minutes
  - Emulator lost Internet connection in every couple of hours

Bugs...

#	Description	Found
1	A button gets stuck on a disabled state	modeling
2	When the main feed is changed, the feed header is not updated	
3	OK and Cancel buttons do not save/restore values in preferences	
4	When the widget feed is changed, the feed view is not updated	
5*	Application sometimes crashes when refreshing feeds after modifying the preferences.	
6	Max headline value zero not handled properly in preferences	
7	Inconsistency when switching between multiple widget feeds	
8	News item count not updated properly in the feeds view	
9	Preferences shows "Select default feed" although the application was started from a widget.	execution
10	Extra whitespaces in news titles	
11	Refresh inconsistency when marking news read	
12	Widget forgets its state when set to the foreground	
13*	Update frequency value zero not handled properly	
14	Widget launches the main app if the widget feed is unavailable	

- 14 bugs found in total
  - 8 during modeling
  - 6 in random test execution from the model
- Two of the bugs caused the application to crash
- Even quite small inconsistencies were found – easily missed by a manual tester
- TEMA has proven strong in finding concurrency related bugs – unfortunately not applicable in this case

Industrial perspective

The bugs found by the modeling and test execution were new and we don't expect they would have been found through manual or other automated testing.

- However, because they were in rarely exercised paths of the code they were not critical to fix

The TEMA tools currently require significant effort to learn how to use them, more effort than many commercial developers may be willing to invest.

We recommend further investment in the project to make the tool easier to use, and would like to see some of the current claims tested, e.g. reuse of high-level models across various phone platforms.

We would also like to see it used to test a variety of commercial software to ascertain the quality and utility of the bugs found by the tools.

Thank you!

TEMA toolset and the keyword tool now available from:
http://tema.cs.tut.fi

Other case studies:

- S60
- Mobile Linux
- Java

Acknowledgements:
Tekes, Nokia, Ixonos, Symbio, Cybercom Plenware, F-Secure, Qentinel, Prove Expertise, Academy of Finland (grant #121012), Graduate School on Software Systems and Engineering (SoSE), Jim Blackler