Automated Visual GUI Testing for the Space Network

Charles Song
Research Scientist
Fraunhofer USA CESE
Space Network

• Proprietary GUI running on OpenVMS
  – The system includes lots of screens
  – Developed in ADA, Fortran etc.

• Current testing practices
  – Create test cases & documentation
  – Manually issue commands and verify results

• Tedious work that should be automated...
SN Testing Challenges

• OpenVMS does NOT have robust GUI testing frameworks
  – Cannot perform testing natively

• SN uses proprietary GUI libraries
  – Cannot port system to other platforms

• GUI libraries also depend on DECNet protocol
  – Difficult to transport GUI to other platforms
Platform Solution

• Transport GUI windows to Linux via X Server
  – Requires DECNet components for Linux
    • DECNet enabled Linux kernel and DECNet tools
    • Custom-compiled DECNet-enabled X Server
  – Fedora Core 6 Linux release with limited support

• Run DECNet-enabled Linux in a Virtual Machine on Windows
  – Finally access to modern GUI testing tools!
Platform Solution

• Transporting SN windows to a modern platform
Reference-Based Testing Tools

• Most robust approach to GUI testing
  – E.g. Selenium for testing web applications
  • Examines underlying HTML code
However...

• X Server does NOT retain GUI information needed for meaningful testing

• Cannot use Reference-based tools for SN testing...
Record & Replay Testing Tools

• Should not use Record & Replay tools...

Click (776, 379)

Click (776, 473)
A Potential Solution

• A screenshot-based GUI automation tool named Sikuli
  – Searches for given patterns on the screen
  – Controls keyboard/mouse inputs
  – Some support for Optical Character Recognition (OCR)

• However, Sikuli NOT designed for testing
  – Automation scripts expressed in low-level interactions
    • Click, type, find...
  – Difficult to reuse test code
    • Scripting vs OO design
PiGuiT: Visual GUI Testing

• Platform-independent GUI Testing
  – Built on top of Sikuli’s Java API
  – Works on all modern platforms with any GUIs technology
  – Designed to bring OO design to GUI testing

• Three-step approach to GUI testing
  – Define “model” for GUI under test
    • Details of GUI look and feel / interactions
  – Abstract higher level GUI logic
  – Create test cases using high level abstractions
PiGuiT Workflow

GUI Model ➞ Custom API ➞ Test Cases

PiGuiT GUI ➞ PiGuiT Core ➞ Tester

Test Cases:
- testCreateScreenElem (0.512 s)
- testAddPadding (0.547 s)
- testCurrentState (0.745 s)
- testExistsColor (0.510 s)
- testFindPredictors (8.935 s)
- testRightClick (6.957 s)
- testSaveScreenElem (2.862 s)
- testCaptureImage (4.171 s)
- testCreateScreenElemFromList (0.514 s)
- testDragDropElemToAnother (13.226 s)
- testTypeWithShiftModifier (1.798 s)
- testTypeWithSpecialKeys (5.850 s)
- testRegionComparators (0.511 s)
PiGuiT Workflow

GUI Model

PiGuiT GUI
Predictors

- Predictors are images that can be used to identify elements of GUI
  - Basically anything users can see on the screen
  - Used for GUI interactions and verification
Predictor States

• Each predictor can have one or more states
  – Element may have several states that are displayed graphically
Components

Components give logical structure to GUI elements

– Can use one or more predictors to define component location/boundary
PiGuiT GUI: Component

- What if a predictor is not unique?
  - E.g. view details of Person of Interest from “googleplus”
PiGuiT GUI: Component

- Designate one predictor as the anchor
- Pick the closest predictor in relative direction
Actions

• A GUI event is represented as an action

• Action types
  – Mouse clicks
  – Keyboard input
PiGuiT GUI: Actions

• Click Action
  – Left, right, double click on a predictor
  – Offset to the predictor

• Drag and Drop (or swipe)

• Type Action
  – Parameters: __<parameter>__
  – Special keys: enter / modifiers: e.g. shift

• Assert/wait action
  – Predictor exist / not exist
  – Wait until predictor appears
Action Sequences

• Logical grouping of a set of actions
  – Performed in a particular sequence in the context of a component

1. Type TAB + TAB
2. Type email + TAB
3. Type password + ENTER

sequence_Login_User()
PiGuiT Workflow

Custom API

PiGuiT Core
Custom API

• PiGuiT
  – Component
  – Action sequence

• Components encoded as classes

```java
LoginComponent login = new LoginComponent();
```

• Action Sequences become their methods

```java
login.sequence_Login_User(email, password);
```
PiGuiT Workflow

Test Cases

- testCreateScreenElem (0.512 s)
- testAddPadding (0.547 s)
- testCurrentState (2.745 s)
- testExistsColor (0.510 s)
- testFindPredictors (8.935 s)
- testRightClick (6.957 s)
- testSaveScreenElem (2.862 s)
- testCaptureImage (4.171 s)
- testCreateScreenElemFromList (0.514 s)
- testDragDropElemToAnother (13.226 s)
- testTypeWithShiftModifier (1.798 s)
- testTypeWithSpecialKeys (5.850 s)
- testRegionComparators (0.511 s)
PiGuiT Login Test

- Test cases are written against high level logic
  - Test cases are shielded from GUI changes

```java
@Test
public void testValidLogin() {
    LoginComponent login = new LoginComponent();

    login.sequence_Login_User(email, password);
    Assert(login.sequence_Login_Successful());
}
```

- Unless the fundamental functionalities of the software change, the test cases remain the same!
Leverage Java Good Practices

• Using JUnit framework to automate GUI test case execution
  – Setup() – brings up screen under test
  – Teardown() – resets screen if necessary
  – Tests – independent, repeatable, self-contained
  – Asserts – verify predictors and component states

  – @BeforeClass/@AfterClass – run expensive setup/cleanup code
    • E.g. Sending all vectors before running SHOs
SN Testing Progress

• Deployed **configurable** initialization scripts for SN equipment
  – Tedious manual steps before every testing session
  – 20 – 30 minutes of tester effort saved

• Developed **extensible** test suite for regression testing
  – Submits and verifies Shos to test SN
    • Extend test suite by adding Shos to specified directories
  – 4 - 6 hours of testing runs overnight unattended
Lessons Learned

• SN fonts are especially difficult for OCR
  – Image recognition is more reliable

• SN GUI does NOT scale on resolution change
  – Aspect ratios change between GUI elements
  – Always maximize SN window on test machine
Lessons Learned

• Image string search == substrings search
  – Differentiate by adding space to the end of predictor

• Automated tests do not need to be perfect to be useful
  – Simplify assumptions
    • E.g. check for link state, no need to verify data rate
    • E.g. take down all equipment and reconfigure
Future Work

- Streamline API generation/jar import
- Improved logging for visual GUI testing
- Custom OCR solution for difficult fonts
- Knowledge transfer PiGuiT technology to NASA testers/developers

Charles Song
csong@fc-md.umd.edu