Review of GUI Reasoning and Testing Article

TP #05 Peer Reviewed by: _____

Alex Laird
CS 3310-01
Date: October 2, 2009
Operating Systems
Fall 2009
Computer Science, (319) 360-8771
alexdlaird@cedarville.edu.edu

Grading Rubric	Max	Earn
On Time/Format	1	
Correct	5	
Clear	2	
Concise	2	
TOTAL	10	

ABSTRACT

This paper is a review of the scholarly journal article *A Generic Library for GUI Reasoning and Testing* by João Carlos Silva, João Saraiva, and José Cressac Campos[2].

Keywords

Scholarly, Review, Graphical, User, Interface, Reasoning, Testing

1. INTRODUCTION

The journal article reviewed discusses how essential the user interface is to an application and how frequently its implementation is overlooked or treated flippantly. Moreover, the easy-to-use tools that are provided by many Integrated Development Environments (IDE) create slow Graphical User Interfaces (GUI) and generated bloated and ugly code.

The proposal is that designers need to spend more time making their GUIs intuitive and friendly rather than daunting powerhouses. To do this, sophisticated reverse engineering methods were created to test and analyze several GUI implementations to come to a better understanding of what it takes to create a proper front-end for an application.

2. BEHAVIORAL MODELS

The idea is a tool would generate a behavioral model[1] from a given GUI from the various components and their actions. It treats a specific GUI as a Finite State Machine (FSM) and diagrams out the paths that can be taken through the interface. From these models, analysis can be made to eliminate unnecessary paths, shorten paths for a particular action, and combine paths that execute the same or similar actions.

GUISURFER was the first tool developed to do this. Specifically, it was used to derive and test a user interface developed using the Swing toolkit in Java. However, other such tools have been made to generate behavioral models for other programming languages as well.

.

3. THE PARSING PROCESS

The tool should parse through the code of a GUI and find specific objects that are to be physically manifest in the GUI, for instance a JButton. The program should take objects it finds in the GUI and place them into a model. It should break the GUI object's behaviors down and represent them in the model as well. Therefore, if a JButton is to validate a username and password and then launch a new window, these actions would be represented in the model and could effectively be manifest in a state diagram as well.

One method of effectively parsing the GUI is by creating slices[3] from each GUI component. An Abstract Syntax Tree (AST) should be created and used to model all GUI elements for easy referencing and testing.

4. MODEL-BASED GUI TESTING

From the parsed model, tests can be more easily run on the GUI using a tool called QuickCheck Haskell. The programmer specifies a list of testing functions for QuickCheck to execute, and QuickCheck does so using a large number of randomly generated cases through the functions, specifically in reference to the GUI. QuickCheck then reports back to the user how many tests were run, whether they were run effectively, and the percentage of tests that took x number of event sequence lengths.

5. CONCLUSIONS

The idea behind reverse engineering GUI code to optimize performance is one that should be explored more. The frameworks, specifically GUISURFER, that have been made towards this endeavour prove very useful. The interfaces in use today are frequently unclear, muddled, over complex, and sometimes even nonfunctional for all practical purposes. Though the ultimate test of a GUIs usefulness will always ultimately be a user testing it, taking strides to provide a usable GUI the *first* time are refreshing to hear about.

6. REFERENCES

[1] P. Bumbulis. Combining formal techniques and prototyping in user interface construction and verification. *University of Waterloo*, 1996.

[2] J. C. Silva, J. Saraiva, and J. C. Campos. A generic library for GUI reasoning and testing. *ACM*, 2009.

[3] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 1995.