A GUI Environment to Manipulate FSMs for Testing GUI-based Applications in Java

Jessica Chen and Suganthan Subramaniam
School of Computer Science, Univ. of Windsor
Windsor, Ont. Canada N9B 3P4
{xjchen,subram2} @cs.uwindsor.ca

Abstract

The development of GUI-based applications has raised a lot of new issues, one of them being how to effectively test complicated graphical user interactions. In this paper, we present a visual environment for manipulating test specifications of GUI-based applications in Java. In our approach, the internal representation of a test specification, which contains the contexts of GUI input and output, is generated interactively by running the Application Under Test (AUT). In this way, existing testing tools, such as tools for test case generation, can possibly be applied on it. We provide a graphical interface to obtain such kind of internal test specifications so that testers do not need to know the details of the internal representation, and the test specification can be easily modified. We present our running prototype which let users graphically manipulate the test specification given in the form of a Finite State Machine, and the implementation of AUT is a GUI-based Java application.

1. Introduction

Testing an implementation of a system with respect to its specification has been extensively studied in the past. Many effective techniques and related testing tools have been developed to support unit testing (including class testing), integration testing, system testing, interoperability testing, etc. Some of the tools have also been integrated with those developed for other stages, e.g. software requirement specification, software design, in the software life cycle. With the advances of software development and the increasing use of web programming techniques, more and more applications now involve Graphical User’s Interface (GUI), which makes the applications much easier to learn and use. On the other hand, the development of GUI-based applications has raised a lot of new issues, one of them being how to effectively test complicated graphical user interactions. Although there exist some off-the-shelf automated tools (see for examples [3, 21, 23]) specially designed for testing GUI-based applications, they all suffer some common weakness e.g. lack of testing requirement specification, and are far from sufficient or effective enough to be widely used. In current software industries, testing GUI-based applications is still mainly carried out manually. This situation is partially due to the lack of related research work. Little work has been done to study the methodology for testing GUI-based applications. In particular, to the knowledge of the authors, no work has been done to provide ways to link GUI-based application testing with traditional testing so that tools developed specially for GUI-based application testing can be easily integrated with existing testing tools. The present work serves as the original contribution to this issue.

One of the main difficulties in testing GUI-based applications lies in describing the input and output. Unlike in traditional applications, input and output in GUI-based applications may have a lot of different context, such as mouse clicks on different GUI-components. These contexts are usually related to the GUI-components, which can have fairly complicated structures and relationships. As a consequence, to accurately describe a test case or test specification, we have to describe all the GUI input events together with their target components and all the GUI output as certain attribute changes of some GUI-components. Furthermore, we need to save such descriptions in certain format so that regression testing can be carried out easily. Capture/Reply is the most widely used technique to automatically obtain such an internal description of a test case (usually in a LOG file). However, the approach suffers the lack of flexibility because we can only replay whatever we recorded while slight changes to the application or the test case may invalidate the recorded test case. Manually modifying the recorded internal representation of the test cases according to the changes in the requirements or design specification, or manually editing test scripts according to certain script languages is very tedious and difficult.

A possible way out is to provide supporting tools which