REST: A Tool for Reducing Effort in Script-based Testing

Qing Xie, Mark Grechanik, and Chen Fu
Accenture Technology Labs, Chicago, IL 60601, USA
{qing.xie,mark.grechanik,chen.fu}@accenture.com

Abstract

Since manual black-box testing of GUI-based APplications (GAPs) *is tedious and laborious, test engineers create test scripts to automate the testing process. These test scripts interact with GAPs by performing actions on their GUI objects. An extra effort that test engineers put in writing test scripts is paid off when these scripts are run repeatedly. Unfortunately, releasing new versions of GAPs breaks their corresponding test scripts thereby obliterating benefits of test automation. We propose a tool called Reducing Effort in Script-based Testing (REST) for guiding test personnel through changes in test scripts so that they can use these modified scripts to test new versions of their respective GAPs. During demonstration of REST we will show how this tool enables test personnel to maintain and evolve test scripts with a high degree of automation and precision.*

1 Introduction

Manual black-box testing of *GUI-based APplications (GAPs)* is tedious and laborious because nontrivial GAPs contain hundreds of GUI screens and objects. Test automation is the key to reduce test time as well as cost [1]. In order to automate testing of GAPs, test engineers write programs using scripting languages (e.g., VBScript), and these programs (*test scripts*) mimic users by performing actions on GUI objects using underlying testing frameworks (e.g., QuickTest Pro). An extra effort put in writing testing scripts is paid off when these scripts are run repeatedly.

Unfortunately, releasing new versions of GAPs breaks their corresponding test scripts thereby obliterating the benefits of test automation. A simple modification on a GUI object may invalidate many statements in test scripts that reference this object. As many as 74% of the test cases become unusable during GUI regression testing [2]. Given the complexity of these scripts, it may take days for test engineers to fix them so that they can test successive releases of the corresponding GAPs.

We demonstrate a tool for *Reducing Effort in Script-based Testing (REST)* for guiding test engineers through changes in test scripts so that they can test their modified GAPs. The core is to enable test engineers to maintain and evolve test scripts with their respective GAPs by providing assistance in determining how to change these scripts with a high degree of automation and precision. The input to REST are GUIs of the successive releases of the same GAP and a test script for the prior release of the GAP. After extracting models of these releases, these models are compared and modified GUI objects are located. Then, the test script is analyzed to determine references to these modified GUI objects and their impact on other statements in this script. REST issues warnings that help test engineers to fix errors in test scripts for successive releases of the GAPs.

2 An Overview of REST

This section presents core ideas of our approach, explains how we model GAPs, and describes the REST architecture.

2.1 Core Ideas

In order to enable checking of references to GUI objects in test scripts statically, we base our solution on four core ideas. First, we compare these GUIs to determine what GUI objects are modified. Second, we detect what references to GUI objects are affected by modifications in these objects of the successive releases of GAPs. Once it is known what statements in test scripts are affected by the modifications to GUIs, we analyze the script to determine what other statements are affected as a result of using values computed by the statements that reference modified GUI objects.

The fourth idea is about inferring values of parameters to *Application Programming Interface (API)* calls that access and manipulate GUI objects. Consider a test script `VbWindow(e1).VbEdit(e2).Set "Joe"` written for the QTP platform. The parameters to the API calls are constant string values that are the names of collections of property values of GUI objects in *Object Repositories (ORs)*. However, in general, these names are not defined as constants but as string expressions whose values are computed at runtime. When a navigation statement is `VbWindow(e1).VbEdit(e2)`, where `e1` and `e2` are string variables, performing analyses becomes difficult. Our idea is to correlate navigation statements with GAP models so that we can infer possible values of string variables.

2.2 Modeling GAPs

We model GAPs as GUI screen-based state machines whose states are defined as collections of GUI objects, their properties (e.g., style, read-only status, etc.), and their values. When users perform actions on GUI objects they change the state of the GAP. In a new state, GUI objects may remain the same, but the values of some of their properties may change.

GUIs of GAPs can be represented as trees whose nodes are composite GUI objects that contain other GUI objects (e.g., frame), leaves are primitive or simple GUI objects (e.g., buttons), and parent-child relationships between nodes or leaves defines a containment hierarchy. In order to operate on GUI objects, scripts must reach nodes of the GAP trees at arbitrary depth. This is accomplished via path expressions that are queries whose results are sets of nodes. A path expression defines a unique traversal through the GUI tree, and it is a sequence of API calls whose parameters are string expressions that compute names of collections of property values of GUI objects as they are specified in ORs, and these calls are separated by dots to reflect the containment hierarchy of GUI objects.

2.3 Architecture

The architecture of REST is shown in Figure 1. Solid arrows show command and data flows between components, and numbers in circles indicate the sequence of operations in the workflow. The inputs to REST are two running successive versions `N` and `N+1` of the GAP, the OR, and the test script for the GAP of the version `N`. Usually, test engineers create one test script per GUI screen to make test design modular. We exploit this property in our architecture.

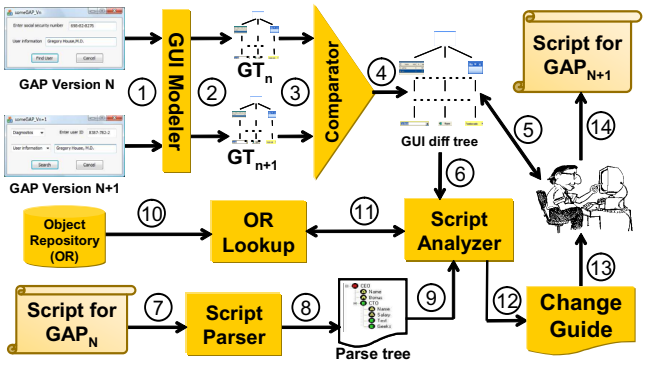

Figure 1. The architecture of REST.

The first step involves modeling the GAPs. To do that, both GAPs are navigated to the GUI screen for which the given test script is designed. The GUI Modeler `(1)` obtains information about the structure of the GUI and all properties of individual objects using the accessibility layer, which is the underlying technology for controlling and manipulating GAPs that is common to major computing platforms.

The GUI Modeler outputs `(2)` GUI trees GT_n and GT_{n+1} for the versions `N` and `N+1` respectively. These trees are compared `(3)` by the GUI tree Comparator to determine what GUI objects are modified between the versions of the GAPs. The Comparator outputs `(4)` GUI difftree that is a combination of the GUI trees GT_n and GT_{n+1} with matched GUI objects mapped to each other.

In general, it is an undecidable problem to compute correct mappings between two GUI trees fully automatically. We use a semiautomatic mapping algorithm to compute the matching score between each GUI object in the GUI tree GT_n and all objects of the tree GT_{n+1} and map the one with the highest score that is above certain threshold. We provide tools `(5)` for the user to modify mappings between GUI objects manually. Given that GUI screens contain less than a hundred GUI objects, fixing incorrectly identified mappings manually does not require any serious effort.

The Script Analyzer analyzes test scripts to determine the impact of GUI changes on these scripts. `(7)` the script for GAP of the version `N` is parsed using the Script Parser and `(8)` the parse tree is generated. This tree contains an intermediate tree representation of the test script where references to GUI objects are represented as nodes.

Recall that GUI objects are described using unique names with which property values of these objects are indexed in ORs. These names are resolved `(10)` into the values of properties of GUI objects using the component OR Lookup `(11)` with which the Script Analyzer interacts when it performs analyses.

Thus the Script Analyzer takes `(9)` the parse tree, `(6)` the GUI difftree, and `(11)` values of properties of GUI objects as its inputs and `(12)` produces a change guide that contains messages about possible failures in test scripts. We categorize these failures into two types: *Wrong-Path (WP)* errors occur when a script accesses GUI objects that may be deleted or moved to a different position in the GUI hierarchy of the successive release of the GAP. *Changed-Object (CO)* errors occur when statements access GUI objects whose types and properties are changed, or when statements operate on GUI objects without considering new constraints that are imposed on these objects in the successive version of the GAP. Test engineers `(13)` review these messages and modify the original test script for GAP version `N` so that it can test the successive version `N+1` of this GAP. In addition, they make corresponding modifications to the property values of the GUI objects in the OR.

References

[1] M. Fewster and D. Graham. *Software Test Automation: Effective Use of Test Execution Tools*. Addison-Wesley ACM Press, September 1999.

[2] A. M. Memon and M. L. Soffa. Regression testing of GUIs. In *Proceedings of the ESEC and FSE-11*, pages 118–127, Sept. 2003.