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.

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]: Testing tools

General Terms

Test scripts, test automation, GUI regression testing

1. INTRODUCTION

Manual black-box testing of *Graphical User Interface (GUI)-based APplications (GAPs)* is tedious and laborious because nontrivial GAPs contain hundreds of GUI screens and thousands of GUI objects. Test automation plays a significant role in reducing the high cost and significant effort of testing GAPs [2][3]. In order to automate testing of GAPs, test engineers write programs using scripting languages (e.g., JavaScript and VBScript), and these programs (*test scripts*) mimick users by performing actions on GUI objects of these GAPs using some underlying testing frameworks (e.g., QuickTest Pro, Rational Functional Tester). An extra effort put in writing testing scripts is paid off when these scripts are run repeatedly to determine if GAPs behave as desired.

Unfortunately, releasing new versions of GAPs breaks their corresponding test scripts thereby obliterating the benefits of test automation [1]. Consider a situation when a combo box is replaced with a text box in the successive release of some GAP. Statements that select different values in this combo box will not work when executed on a text box. This simple modification may invalidate many statements in test scripts that reference this GUI object. As many as 74% of the test cases become unusable during GUI regression testing [4]. Given the complexity of these scripts, it takes from hours to days for test engineers to fix them so that they can test successive releases of the corresponding GAPs.

We propose to demonstrate a tool *Reducing Effort in Script-based Testing (REST)* for guiding test personnel through changes in test scripts so that they can test their modified GAPs. The core of our proposed demonstration is to enable test personnel 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. The result of this analysis is issued warnings that help test engineers to fix errors in test scripts so that they can test successive releases of their GAPs.

2. AN OVERVIEW OF REST

This section presents core ideas behind 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 in order to determine what GUI objects are modified. Second, using this information we detect what references to GUI objects are affected by changes in these objects of the successive releases of GAPs. Next, 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 API calls that access and manipulate GUI objects. Consider a test script `VbWindow(e1).VbEdit(e2).Set "Joe"` written for the QuickTest Professional (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 ORs. However, in general, these names are not

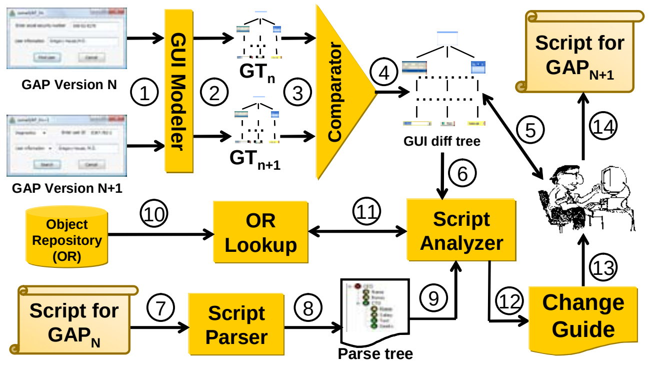


Figure 1: The architecture of REST.

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. We discuss how we model GAPs in the next section.

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 GUIs versions `N` and `N+1` of the running successive releases 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.

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 $\texttt{GT}_\texttt{n}$ and $\texttt{GT}_\texttt{n+1}$ for the versions `N` and `N+1` respectively. These trees are compared `(3)` by the GUI tree Comparator in order 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 $\texttt{GT}_\texttt{n}$ and $\texttt{GT}_\texttt{n+1}$ 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 $\texttt{GT}_\texttt{n}$ and all objects of the tree $\texttt{GT}_\texttt{n+1}$ 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. Normally, 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 is the core component that analyzes test scripts to determine the impact of GUI changes on these scripts. To do that, `(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 the GAP of the 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.

3. REFERENCES

[1] S. Berner, R. Weber, and R. K. Keller. Observations and lessons learned from automated testing. In *Proceedings of the ICSE '05*, pages 571–579, New York, NY, USA, 2005. ACM.

[2] E. Dustin, J. Rashka, and J. Paul. *Automated Software Testing: Introduction, Management, and Performance.* Addison-Wesley, September 2004.

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

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

APPENDIX

A. INTRODUCTION

The demonstration will begin with a brief introduction to the regression testing process with test scripts and the difficulties of reusing these test scripts during software evolution - a simple modification of a GUI object may invalidate many statements in test scripts that reference this object.

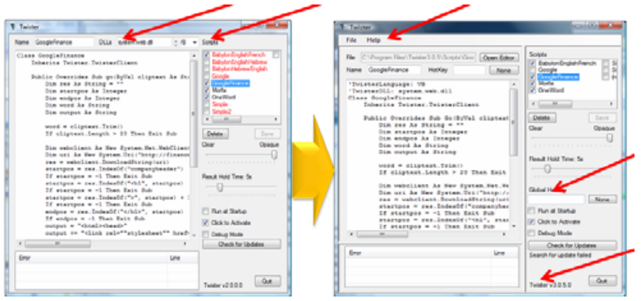
Following the introduction, we will present the state-of-the-practice of test scripts maintenance: as many as 74% of existing test scripts are rewritten for successive releases of the corresponding GUI applications (GAPs) because given the complexity of these test scripts, it takes hours even days to comprehend and then fix them. The current practice leads to high cost of testing even IT disruption.

The discussion will lead us into introducing our tool Reducing Effort in Script-based Testing (REST): a new way to repair test scripts efficiently and correctly. REST extends existing test scripts to test successive releases of applications with a high degree of automation by: (1) determine changes in the GUIs of applications; (2) find affected statements in test scripts; and (3) guide the script repair process.

B. A MOTIVATING EXAMPLE

A motivation example of a real GUI application (`Twister`) using the above efficient way to repair test scripts is presented for two successive releases of `Twister` 2.0 and `Twister` 3.0.5.

`Twister` is a Windows-based open-source application that captures text on the clipboard and runs scripts on that text to translate it, get dictionary definitions, look up stock quotes, etc. GUI screens for two successive releases of this GAP are shown in Figure 2 with the block arrow that shows evolution from previous release to the next one.



```
Class GoogleFinance
    Inherits Twister.TwisterClient

    Public Overrides Sub go(ByVal cliptext As St
        Dim res As String = ""
        Dim startpos As Integer
        Dim endpos As Integer
        Dim word As String
        Dim output As String

        word = cliptext.Trim()
        If cliptext.Length > 20 Then Exit Sub

        Dim webclient As New System.Net.WebClient
        Dim uri As New System.Uri("http://financ
        res = webclient.DownloadString(uri)
        startpos = res.IndexOf("companyheader")
        If startpos = -1 Then Exit Sub
        startpos = res.IndexOf("<h1", startpos)
        If startpos = -1 Then Exit Sub
        startpos = res.IndexOf(">", startpos) + 1
        If startpos = -1 Then Exit Sub
        endpos = res.IndexOf("</h1>", startpos)
        If endpos = -1 Then Exit Sub
        output = "<html><head>"
        output += "<link rel=""stylesheet"" href=
```

```
'TwisterLanguage: VB
'TwisterDLL: system.web.dll
Class GoogleFinance
    Inherits Twister.TwisterClient

    Public Overrides Sub Go(ByVal cliptext
        Dim res As String = ""
        Dim startpos As Integer
        Dim endpos As Integer
        Dim word As String
        Dim output As String

        word = cliptext.Trim()
        If cliptext.Length > 20 Then Exit

        Dim webclient As New System.Net.We
        Dim uri As New System.Uri("http:/
        res = webclient.DownloadString(uri
        startpos = res.IndexOf("companyhea
        If startpos = -1 Then Exit Sub
        startpos = res.IndexOf("<h1", star
        If startpos = -1 Then Exit Sub
```

Figure 2: Snapshots of `Twister`, versions 2.0 and 3.0.5.

Red arrows shown in Figure 2 indicate the GUI objects that are added or deleted. Some buttons are deleted from the previous release, and a menu bar with menu items are added in the next release, as well as the property values of few GUI objects are modified. The example will illustrate how a few small differences between these two GUIs can have a big impact on the test scripts.

C. ARCHITECTURE OF REST

This discussion leads directly into the overview of our approach, the architecture of REST, which is shown earlier in Figure 1 with detailed interpretation in Section 2.3.

We will also provide a brief background on accessibility technologies. The goal of accessibility technologies is to provide different aids to disabled computer users. Specific aids include screen readers for the visually impaired, visual indicators or captions for users with hearing loss, and software to compensate for motion disabilities. Accessibility technologies provide a wealth of sophisticated services required to retrieve attributes of GUI elements, set and retrieve their values, and generate and intercept different events. Most computing platforms include accessibility technologies, such as *Microsoft Active Accessibility (MSAA)* technology and *Sun Microsystems Accessibility* technology.

D. DEMONSTRATING REST

Following the discussion of the architecture of REST, we plan to demonstrate REST by walking through the differences between the two successive releases of the corresponding GAP, parsing and analyzing test scripts to find the affected statements, and suggesting the solutions to resolve the potential problems.

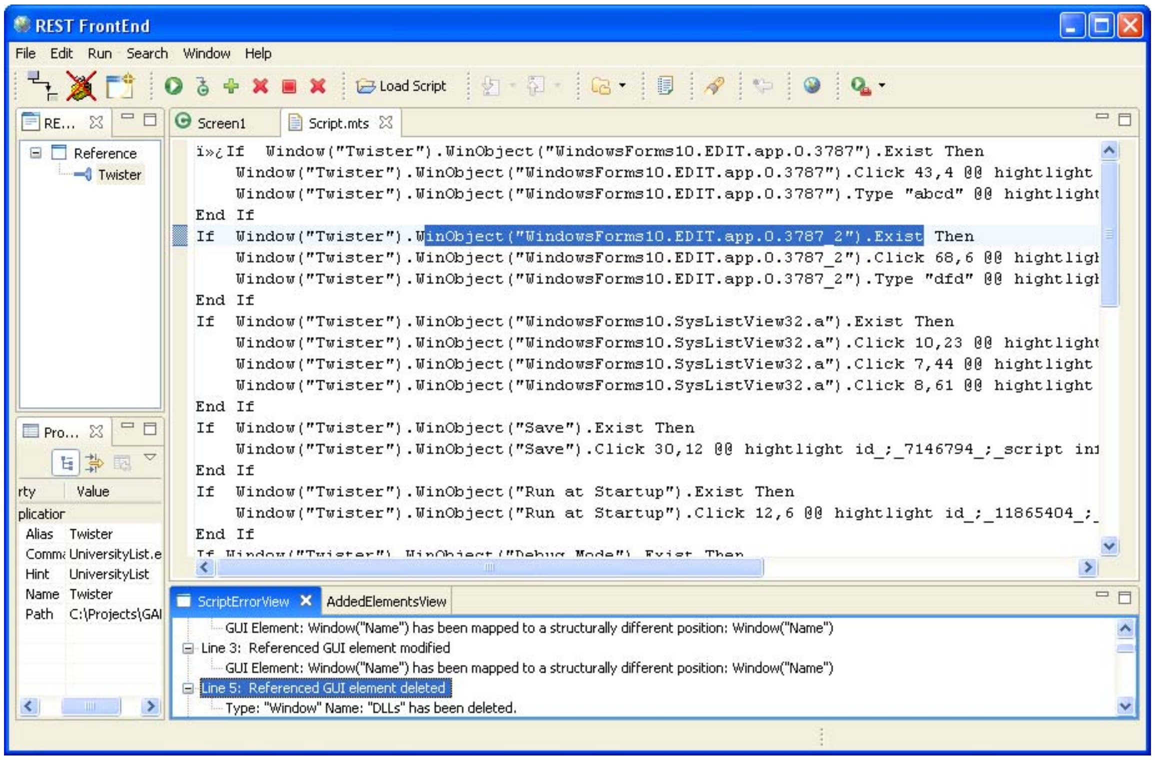
The front end of our tool is shown in Figure 3. We built REST as a plugin for Eclipse. The dockable window `REST Explorer` that is shown in the left top corner displays properties of the currently run GAP. The dockable window `Properties` that is shown in the lower left corner displays values of different properties of GUI objects. The script window that is shown in the main view shows the script, and the dockable window on the bottom of the main view shows the tab `ScriptErrorView` with the list of warnings of failures that this script may exhibit when applied to the next release of the GAP. Other tabs show the lists of deleted and added GUI objects.

A key characteristics of our approach is that test engineers interact with GAPs by performing actions against GUI objects instead of running test scripts in a debugger in order to find where they break. That is, the user first navigates to the screen for the GAP of the previous release and to the corresponding screen in the successive release of the GAP. Then the user presses a button on the REST toolbar to capture the structures of the screens and then transcode these structures into models that will be analyzed by the GUI comparator.

Once the models of the GUIs are compared, the lists of added and deleted elements are shown in the corresponding tabs of the dockable window. The user can select entries in these list in order to view these GUI objects on the GAPs. When an entry is selected, a frame is drawn around the GUI object with the tooltip window displaying the information about the selected element. In addition, the user can moves the cursor over any GUI object, and REST displays a context menu that allows the user to check to see what other object this selected object is mapped to, add or remove the existing mapping, or mark this object as added or removed. This way the user can modify the results of GUI comparison especially given that these results may contain errors.

The user views the results of the analysis in the tab ScriptErrorView. By clicking on any warning messages, the reference in the script to GUI object is selected along with drawing a frame around the affected object on the corresponding GAP. A short movie demonstrating how REST works is available at our website[1].

[1]http://www.markgrechanik.com/Rest/Rest.html



rty	Value
Alias	Twister
Comm:	UniversityList.e
Hint	UniversityList
Name	Twister
Path	C:\Projects\GAI

```
ï»¿If  Window("Twister").WinObject("WindowsForms10.EDIT.app.0.3787").Exist Then
     Window("Twister").WinObject("WindowsForms10.EDIT.app.0.3787").Click 43,4 @@ hightlight
     Window("Twister").WinObject("WindowsForms10.EDIT.app.0.3787").Type "abcd" @@ hightlight
End If
If   Window("Twister").WinObject("WindowsForms10.EDIT.app.0.3787_2").Exist Then
     Window("Twister").WinObject("WindowsForms10.EDIT.app.0.3787_2").Click 68,6 @@ hightligh
     Window("Twister").WinObject("WindowsForms10.EDIT.app.0.3787_2").Type "dfd" @@ hightligh
End If
If   Window("Twister").WinObject("WindowsForms10.SysListView32.a").Exist Then
     Window("Twister").WinObject("WindowsForms10.SysListView32.a").Click 10,23 @@ hightlight
     Window("Twister").WinObject("WindowsForms10.SysListView32.a").Click 7,44 @@ hightlight
     Window("Twister").WinObject("WindowsForms10.SysListView32.a").Click 8,61 @@ hightlight
End If
If   Window("Twister").WinObject("Save").Exist Then
     Window("Twister").WinObject("Save").Click 30,12 @@ hightlight id_;_7146794_;_script in
End If
If   Window("Twister").WinObject("Run at Startup").Exist Then
     Window("Twister").WinObject("Run at Startup").Click 12,6 @@ hightlight id_;_11865404_;_
End If
If Window("Twister").WinObject("Debug Mode").Exist Then
```

```
    GUI Element: Window("Name") has been mapped to a structurally different position: Window("Name")
⊟ Line 3:  Referenced GUI element modified
    GUI Element: Window("Name") has been mapped to a structurally different position: Window("Name")
⊟ Line 5:  Referenced GUI element deleted
    Type: "Window" Name: "DLLs" has been deleted.
```

Figure 3: The front-end of REST.

E. CONCLUDING REMARKS

Time permitting, we hope to discuss some future directions and elicit feedback from the audience. We are particularly interested in conducting a case study that is aimed at finding out how much time a test personnel will save to repair test scripts using REST. Other directions include applying machine learning techniques to improve precision of mapping each GUI objects between two successive GUI trees automatically and imposing a GUI type system to facilitate reasoning the cause of script failures.