GUIDE: A GUI DifferEntiator

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

Abstract

Applications with Graphical User Interfaces (GUIs) are ubiquitous. Nontrivial GUI-based applications (GAPs) evolve frequently, and understanding how GUIs of different versions of GAPs differ is crucial for various tasks such as testing and project effort estimation.

We offer a novel approach for comparing GUIs. We built a tool called GUI DifferEntiator (GUIDE) that allows users to visualize differences between GUIs of running GAPs automatically, so that the identified changes can be served as guidance to test the new release of the GAP or to estimate the effort of a project.

1. Introduction

GUI-based applications (GAPs) (including web applications) are ubiquitous. Nontrivial GAPs contain hundreds of GUI screens and thousands of GUI objects, and it takes years to develop a large-scale GAP. During development, GAPs evolve through many versions, and understanding how GUIs of different versions of GAPs differ is crucial for various tasks such as testing and project effort estimation.

Consider testing as an example. Currently GUI testing is either performed manually or using some underline testing framework (e.g., QuickTest Pro, Rational Functional Tester) to capture users' interactions with the GUI as test scripts and replay these scripts later on a new version. An impediment to using this approach is that releasing new versions of GAPs with modified GUIs breaks their corresponding test scripts [5] [3]. The modifications between two successive releases of GAPs can cause as many as 74% of the test cases to become unusable during GUI regression testing [6]. Knowing the difference between two versions helps test engineers to identify the affected statements that reference the changed GUI objects in the test scripts and to write new code to test the new GUI objects in the new version.

The other application of GUI comparison is in prototyping. GAPs are typically designed using agile processes, namely, rapid prototyping, short iterations and quick feedback. A prototype built upon initial requirements is presented to different stakeholders in order to receive early feedback [4]. This feedback often leads to changes in the prototype and the original requirements as stakeholders refine their vision. When a substantial number of requirements change, the existing prototype is often discarded, a new one is built, and the cycle repeats. In order to estimate development and testing effort, GUIs of prototypes should be compared in order to determine how they differ.

Finding differences between GUIs can be done by comparing the source code of their corresponding GAPs. However, there are two fundamental limitations to comparing GUIs using GAPs' source code. First, source code of GAPs is not always available especially in black-box testing [2], and GUI testing is black-box as operations are performed on GUI objects rather than objects in the source code. Second, even if the source code is available, there are limitations that render approaches of comparing GUIs using source code ineffective. Consider a situation when GUI objects are created using the API call CreateWindow, which takes a number of parameter variables including a string variable that holds the value of the type of the GUI object. The value of this variable is often only known at runtime, making it impossible to derive GUI from the source code.

We offer a novel approach for comparing GUIs. Our approach is non-intrusive, platform and language-independent, and it adopts distributed design. We developed a tool called GUI DifferEntiator (GUIDE) that allows users to visualize differences between the GUIs of running GAPs, and exploit the information to achieve various tasks.

2. An Overview of GUIDE

This section explains how we model GAPs, presents the comparison algorithm of our approach, and describes how we implement GUIDE.

2.1. Modeling GAPs

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. Each GUI object is represented as a set of properties of the object and values of the properties.

2.2. Comparison Algorithm

We designed a comparison algorithm for computing mappings between GUI objects of GUI trees. This algorithm takes as its input the GUI trees Γ_n and Γ_{n+1} for two successive releases of the same GAP and computes the mapping set μ that contains relations between GUI objects θ and ρ of these trees along with their match score σ, $((\theta, \rho) \in \mu, \sigma), \theta \in \Gamma_n, \rho \in \Gamma_{n+1}, 0 \leq \sigma \leq 1$. The main idea of the algorithm is to compute the match score σ for each pair of GUI objects between successive releases of the GAP, and the final mappings between GUI objects are determined on a basis of the highest scores.

2.3. Implementation

GUIDE is designed keeping two concerns in mind. First, it should be non-intrusive, platform and language-independent. As we mentioned earlier in Section 1 that GUI testing is typically black-box testing, we have no access to the source code. It is important for us to find a way to access and manipulate the properties and the behavior of the GUI objects. We adopted accessibility technologies to retrieve attributes of GUI objects, set and retrieve their values, and generate and intercept different events. Accessibility technologies provide different aids to disabled computer users, they are mandated on most platforms by Electronic and Information Accessibility Standards [1]. The main idea of most implementations of accessibility technologies is that GUI objects expose a well-known interface that exports methods for users to write code to access and control GUI objects of GAPs as if these objects were standard programming objects. Second, it should be distributed so that two versions of a GAP can run on two different machines. We put this concern upfront mainly because we found it is hard to compare two GUIs side by side especially when one overlaps with the other. Moreover, we don't want the two versions to interfere with each other as some GAPs (e.g., SAP applications) are too large (occupy too many resources) to run on the same machine.

GUIDE has two main components, Front-end and Agent, shown in Figure 1. Agent (up right on Figure 1) communicates with Front-end by exchanging XML data via socket. It uses the accessibility layer to inject a hook into the GAP. The hook spawns a thread within the GAP's process in order to build a channel between Agent and the GAP so that Agent can send commands and receive notifications of the events that occur within the GAP. Front-end is an eclipse plug-in (bottom left on Figure 1). It provides different views for users to specify the versions for comparison and visualize the differences.

The inputs to GUIDE are two running successive versions N and N+1 of the GAP (provided by user by selecting from a list of running processes). The first step involves modeling the GAPs (Section 2.1). Agent obtains information about the structure of the GUI and all properties of individual objects using the accessibility layer. The state XML GUI trees GT_n and GT_{n+1} for the versions N and N+1 respectively as the outputs from the first step are sent to Front-end. These trees are compared by Front-end using the algorithm mentioned in Section 2.2 to determine what GUI objects are modified between the versions of the GAPs. The outputs are sorted out and displayed via different views, e.g., Added Elements View for listing newly introduced elements to the new version. In general, it is an undecidable problem to compute correct mappings between two GUI trees fully automatically. We also provide tools 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.

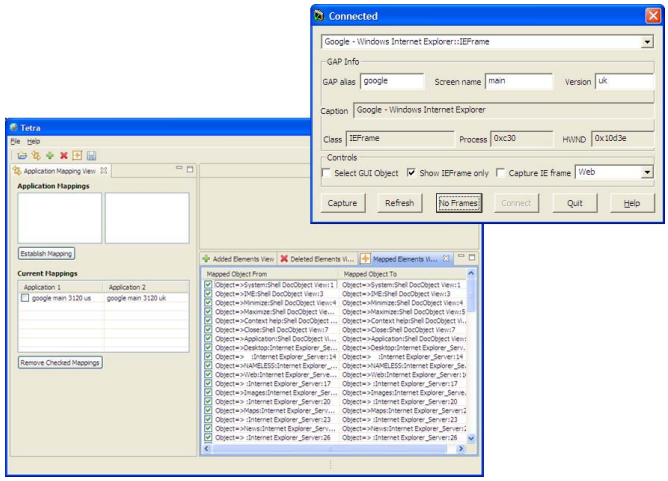

Figure 1. GUIDE Components.

References

[1] Section 508 of the Rehabilitation Act. *http://www.access-board.gov/508.htm.*

[2] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, 2nd edition, 1990.

[3] S. Berner, R. Weber, and R. K. Keller. Observations and lessons learned from automated testing. In *ICSE '05*, pages 571–579. ACM, 2005.

[4] B. W. Boehm, T. E. Gray, and T. Seewaldt. Prototyping vs. specifying: A multi-project experiment. In *ICSE '84*, pages 473–484. IEEE Press, 1984.

[5] C. Kaner. Improving the maintainability of automated test suites. *Software QA*, 4(4), 1997.

[6] A. M. Memon and M. L. Soffa. Regression testing of GUIs. In *FSE'03*, pages 118–127, Sept. 2003.