Industrial Adoption of Automatically Extracted GUI Models for Testing

Pekka Aho[1,2] pekka.aho@vtt.fi, Matias Suarez[3] matias.suarez@f-secure.com, Teemu Kanstrén[1,4] teemu.kanstren@vtt.fi, and Atif M. Memon[2] atif@cs.umd.edu

[1] VTT Technical Research Centre of Finland, Oulu, Finland
[2] University of Maryland, College Park, MD, USA
[3] F-Secure Ltd, Helsinki, Finland
[4] University of Toronto, Toronto, Canada

Abstract. Crafting the models for effective model-based testing (MBT) requires deep understanding of the problem domain and expertise on formal modeling, and creating the models manually from the scratch requires a significant amount of effort. When an existing system is being modeled and tested, there are various techniques to automate the process of producing the models based on the implementation. Especially graphical user interface (GUI) applications have been a good domain for reverse engineering and specification mining approaches, but the existing academic approaches have limitations and restrictions on the GUI applications that can be modeled, and none of them have been adopted by the industry for testing commercial software. Although using implementation based models in testing has restrictions and requires special consideration, the generated models can be used in automated testing and supporting various manual testing actions. In this paper we introduce an industrial approach and platform-independent Murphy tool set for automatically extracting state models for testing GUI applications.

1 Introduction

Model-based testing (MBT) is a technique of generating test cases from behavioral models of the system under test (SUT). The idea is to provide more cost-effective means for extensive testing of complex systems. Instead of manually writing a large set of test cases, a smaller set of test models are built to describe generally the behavior of the SUT and how it should be tested. A test generator tool is then used to automatically generate test cases from these models. There are several benefits, including easier test maintenance due to fewer artifacts to update, higher test coverage from the generated test cases, and documenting the SUT behavior in higher level models which helps in sharing the information and understanding the system [1]. However, crafting the models requires a great deal of expertise in formal modeling and a deep understanding of the problem domain. Constructing the models manually from the scratch requires also a significant amount of effort [2].

There are several approaches aiming to reduce the time required for designing the test models for MBT by automating some parts of the modeling process, such as creating models through reverse engineering or specification mining. Especially in the area of graphical user interface (GUI) software, there are promising academic approaches to automatically construct models based on observing an existing application and using the models for testing purposes, such as [3] [4] [5] [6]. Unfortunately most of these approaches have limitations and restrictions on the GUI applications that can be modeled, and so far none of them have been adopted by the industry to test commercial software products.

As the generated models are based on the behavior of the observed implementation, instead of the specifications or expected behavior, it is challenging to automatically generate meaningful test oracles. In most of the dynamic GUI reverse engineering approaches for testing, the test oracle is based on the observed behavior of an earlier version of the GUI application. Using this kind of test oracle, changes and inconsistent behavior of the GUI can be detected, but validation and verification against the specifications is problematic. Although using implementation based models in testing has restrictions and requires special consideration, the generated models can be used in automated testing and supporting various manual testing actions.

In this paper we introduce a platform-independent industrial approach and Murphy tool set for automatically extracting finite state machine (FSM) based models for testing GUI applications. The approach is based on observation and analysis of the GUI during automated interaction and execution of the application.

2 Background and Related Work

There are a few approaches using static analysis of the source code for automatically constructing models of the GUI software, such as [7], but the dynamic approaches that involve executing the GUI application and observing the application during the run-time are better suited for extracting the behavior of GUI applications [2].

Grilo et al. [2] describe a dynamic approach for reverse engineering GUI applications using a combination of manual and automated steps in the modeling process. The tool uses Microsoft UI automation library for the automated steps and the created model has to be manually validated and completed with the expected behavior. The final models are in Spec# format and can be used for model-based GUI testing (MBGT).

Memon et al. [8] have extensively published their research on GUI Ripping, a technique for dynamically reverse engineering models of GUI applications for test automation purposes. Memons team has implemented GUITAR tool set, a model-based system for automated GUI testing, to execute and observe Java GUI applications to generate models for MBGT. The main target of GUITAR tool set has been Java desktop applications, but it can also be used to model other GUI applications, such as web and android applications, to some extent.

Miao et al. [5] propose a finite-state machine (FSM) based GUI Test Automation Model (GUITAM). In GUITAM, a state of the GUI is modeled as a set of opened windows, GUI objects (widgets) of each window, properties of each object, and values of the properties. Events or GUI actions performed on the GUI may lead to state transitions and a transition in GUITAM is modeled with the starting state, the event or GUI action performed, and the resulting state. To reduce the amount of states into computationally feasible level, not all different property values are considered for distinguishing different states of GUITAM. The authors provide only a short introduction of the tool for automatically constructing the models and the tool is not publicly available.

Aho et al. [9] present GUI Driver, a dynamic reverse engineering tool for Java GUI applications. In [3] the authors introduce an iterative process of manually providing the valid input values into the GUI application and automatically improving the created models. They highlight the importance of increasing the level of GUI automation in order to include all parts of the GUI application in the created models. Unfortunately the tool is currently restricted to Java based GUI applications only.

GUITAR, GUI Driver and GUITAM model a GUI window in the same way: A window consists of widgets, properties of the widgets, and values of the properties. GUITAM and GUI Driver use a similar FSM-based GUI state model where the nodes are states of the GUI and the edges are user actions that trigger transitions between the states. In event flow graph (EFG) [8] or event interaction graph (EIG) [10], created by the GUITAR tool set, the nodes of the model are events or user actions, and the edges capture the flow of the GUI events.

Amalfitano et al. [6] have researched automated modeling and testing of Android applications. The approach is based on a tool that explores the application GUI by simulating real user events on the user interface and reconstructs a GUI tree model. The nodes of the tree represent individual user interfaces in the Android application, while edges describe event-based transitions between interfaces. The GUI exploration technique supports the automatic derivation of test cases that can be executed both in crash testing and regression testing processes.

3 Automated Extraction of GUI Models for Testing

F-Secure Ltd is a software company from Finland having both client and server side products related to safety and security, such as virus protection, including applications with GUI for the end users. F-Secure have developed a tool set called Murphy for automatically extracting models of GUI applications from the user interface (UI) flow, and using the created models for GUI testing.

Murphy dynamically analyses the GUI while automatically interacting with the application, as if it were an end user trying out all the possible user interactions, such as entering text in a text field, pressing a button or a link, selecting items or ticking checkboxes. The main idea is to traverse through all the possible states of the GUI application and automatically construct a finite state machine

(FSM) based model of the observed behavior during the execution, or as we call it, crawling the GUI.

The goal of the Murphy tool is not to find all the possible paths to reach a specific node, but to discover as many nodes as possible. The reason for this is that the GUI applications tend to have a very large number of paths between the nodes, making it impractical to try to reach a specific node again through a different path. Instead, with an appropriate level of abstraction, covering all the states of the GUI is more practical approach. However, it is possible to customize the scripts of Murphy tool to contemplate special cases, for example if the models are meant to be used for testing all the possible transitions between specific states.

To accomplish platform independency, Murphy uses various approaches called drivers for recognizing elements and windows of the GUI application. Among the already implemented drivers, one uses Windows APIs for UI element detection and enumeration, another uses a proprietary API developed by F-Secure for enumerating and querying windows and elements of the UI, and a third driver simulates the end user by cycling through the UI elements by pressing the 'tab' key and analyzing the changes in the screen to determine the elements and behavior of the GUI. The idea is to compare automatically taken screenshots to find the changing areas, such as the bounding rectangle of the selected element on the screen and the shape of the mouse cursor, and reason the structure and behavior of the GUI based on the clues that the GUI application offers for the end user.

Internally Murphy creates a directed graph to model the behavior of the GUI application. The nodes of the model are states of the GUI screen or window, and the edges are actions that the end user could perform in that specific state of the GUI, in a similar way as in [9] and [5]. An edge of the model could be for example pressing an OK button, selecting a specific item in a drop down box, or entering a predefined value in a text field. A node of the model, i.e. the state of a GUI application, is defined mainly by its appearance, excluding data values, such as texts in the text fields, selected values of drop down boxes or status of check boxes. In other words, a dialog that has an OK button enabled represents a different node than otherwise similar dialog with the OK button disabled, but a dialog would represent the same node regardless of the value in a text field of the dialog. During the UI crawling, screenshots of the GUI are automatically captured after each interaction, and they are used for visualization of the resulting graph; a picture of the GUI in that specific state is presenting the node in the graphical presentation of the model. The images are also used by one of the drivers for detecting changes in the UI and the interactions that are available for the end user.

Murphy provides generic UI crawling and window scrapping services as a library. The process of extracting the model is mostly automated and fully customizable, and Murphy provides hooks and callbacks for such customizations. The script that is used for invoking the generic UI crawling library can be modified with application specific rules, but often the generic UI crawler library is

sufficient for generating the models. For simple GUI applications, the invocation script will merely setup the initial state, such as start up the application to be modeled, and then invoke the generic UI crawler library. For more complex GUI applications, application specific modifications can be used for adjusting or correcting the behavior of the generic UI crawler library, for example adding extra edges to a node when all interactions were not properly recognized, instructing the crawler of the values to be used in certain text fields, removing edges from a node, or instructing the crawler not to visit specific edges of the specified nodes. For example, in a dialog for selecting the language for the installation of an application, the invocation script could be modified to instruct the UI crawler library to crawl only the English version of the UI flow. The model extraction would have defined the edge for selecting the installation language into the model, but the UI crawler would ignore it and select English.

In order to limit the scope of the UI crawler into the areas of interest, Murphy has the notion of boundary nodes. Boundary nodes are used for marking the nodes that Murphy should not go beyond during the UI crawling, for example when the GUI application launches a web browser and opens a web page, or when it is not feasible to crawl through the whole help system of the GUI application. Boundary nodes are manually specified into the invocation script. In our experience, it was also useful to add edges representing special actions with environment considerations, for example performing certain UI action when access to the network is not available, or when running low on memory. These special edges have to be manually inserted into the invocation script but in some cases they enriched the resulting model in useful ways.

The Murphy tool has been able to satisfactory extract most parts of the UI flow of the GUI applications of F-Secure with very little user intervention. The invocation scripts were usually between 10 and 200 lines of Python code and produced models capable of exercising in the range of 80% of the possible UI flows. It is important to notice, that most of the GUI applications used in experimenting the approach and Murphy tool set were flow based applications having a relatively low amount of the possible values that the end user may use as input into the system.

4 Discussion and Conclusion

In this paper we have introduced an approach and Murphy tool set for automatically extracting GUI models that can be used in testing GUI applications. Compared to the related approaches, the main advantage of Murphy tool set is that it does not restrict the modeled GUI application to a specific programming language or execution platform. To accomplish platform independency, Murphy uses various approaches called drivers for recognizing elements and windows of the different types of the modeled GUI applications.

We have promising preliminary experiences on using the generated models for testing commercial software products in industrial testing environment. It seems that using the automatically generated models for automating the GUI

testing would reduce the amount of hand written code related to GUI testing compared to manually written test scripts, which reduces the maintenance effort related to test code. Also, with the help of Murphy tool set, it is possible to use the models to support and reduce the effort required for manual GUI testing.

So far, we have used the approach and Murphy tool set only on desktop and web applications, but in future we plan to use the approach also on mobile applications.

Acknowledgment

This work was partially supported by grant number CNS-1205501 by the US National Science Foundation, and a part of ITEA2/ATAC project funded by the Finnish Funding Agency for Technology and Innovation TEKES.

References

1. M. Utting, and B. Legeard, *Practical Model-Based Testing: A Tool Approach*, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 2006.
2. A.M.P. Grilo, A.C.R. Paiva, and J.P. Faria *Reverse Engineering of GUI Models for Testing*, 5th Iberian Conference on Information Systems and Technologies (CISTI), Santiago de Compostela, Spain, 2010.
3. P. Aho, N. Menz, and T. Raty, *Enhancing generated Java GUI models with valid test data*, 2011 IEEE Conference on Open Systems (ICOS 2011), 25-28 Sep 2011, Langawi, Malaysia.
4. A. M. Memon *An event-flow model of GUI-based applications for testing*, Software Testing, Verification and Reliability, Volume 17, Issue 3 (Sep 2007).
5. Y. Miao, and X. Yang *An FSM based GUI Test Automation Model*, 11th Int. Conf. Control, Automation, Robotics and Vision Singapore, 7-10th Dec 2010.
6. D. Amalfitano, A. R. Fasolino and P. Tramontana *A GUI Crawling-Based Technique for Android Mobile Application Testing*, 3rd Int. Workshop on Testing Techniques & Experimentation Benchmarks for Event-Driven Software, IEEE CS Press, 2011, pp. 252-261.
7. J.C. Silva, C. Silva, R.D. Gonalo, J. Saraiva, and J.C. Campos *The GUISurfer tool: towards a language independent approach to reverse engineering GUI code*, Proc. 2nd ACM SIGCHI symposium on Engineering interactive computing systems, Berlin, 2010, Germany, pp. 181-186
8. A. M. Memon, I. Banerjee, and A. Nagarajan *GUI ripping: reverse engineering of graphical user interfaces for testing*, Proc. 10th Working Conference on Reverse Engineering (WCRE'03). IEEE Comp Society, Washington DC, USA.
9. P. Aho, N. Menz, T. Raty, and I. Schieferdecker *Automated Java GUI Modeling for Model-Based Testing Purposes*, 8th Int. Conf. on Information Technology : New Generations (ITNG2011), April 11-13, 2011, Las Vegas, Nevada, USA.
10. Q. Xie, and A. M. Memon *Rapid crash testing for continuously evolving GUI-based software applications*, Proc. 21st IEEE Int. Conf. on Software Maintenance (ICSM'05), IEEE Computer Society, Washington DC, USA, 473-482.