# A Model for GUI Automated Testing Framework in Software System

J.Prabhu
Research Scholar,
CMJ University,
Shillong, Meghalaya, India

G.Gunasekaran, PhD.
Principal
Meenakshi college of Engineering,
K.K.Nagar, Chennai-78, India

## ABSTRACT

A pervasive method for GUI testing is the Capture and Playback (CP) technique. This commonly used technique cannot be used until an Application Under Test (AUT) is completely developed. In this paper we propose a specification driven approach to test GUI-Based java programs as an alternative to the CP technique. We introduce a GUI-event test specification language based on Java Script from which an automated test engine is generated.

The esteem of Java as a scripting language is its ease of use and its standard format that have made writing a test script using our proposed specification language makes it an easy task. Beside the ability to test AUT before being completely developed we have implemented our approach that can generate the test specification file for an already existing AUT. The Tool generates GUI events, where Captures and Playback event responses to automatic verification of the results for the test cases which are written to a test log file. This approach supports M-version testing, where each version of the application is intended to satisfy the same specification.

**General Terms**
Software Testing, Testing Model, Architecture

**Keywords**
GUI Testing, Testing Tools, Java Script , Graphical User Interfaces, Application Under Testing

## 1. INTRODUCTION

Graphical User Interface (GUI) has become an important and accepted way of interacting with software leading to more and more complex GUIs. Although they make software easy to use from user's perspective, they complicate the challenges in testing the correctness of a GUI [1].

The main challenge is that GUI - based programs are event driven, where an event is triggered when the user interacts with the program through GUI. Common user interactions include moving or clicking the mouse, selecting a graphic object, typing into a text field, or closing a window. The fundamental difference between event-driven programs and data-driven programs complicate test automation. Simple test automation involving input or output redirection that is adequate for data-driven programs will not be suffice for GUI based testing which requires a combination of data and event stimuli. Special tools are needed to simulate inputs and user actions that occur through the graphical user interface [2].With the commonly used CP tools, a test designer interacts with the GUI of the AUT and all the events are recorded in a test script.

The test script can later be replayed by the CP tool to recreate user interactions. CP tools are effective in saving the development time of GUI test programs; however, a deficiency of CP-based technologies is that test scripts cannot be generated before an AUT is ready for testing.

Thus, the captured test scripts have more description of the system behavior than a system specification. It may also be difficult to maintain, when the system specification changes. Furthermore, test-first programming can never be applied with CP tools, since nothing can be captured in advance [3]. Therefore, there are researches studying specification based approaches for GUI testing .These researches aim at defining GUI specification languages for the definition of system behavior and then generate test scripts based on the specification languages. The test scripts written in these specification languages can be automatically executed to perform verification of the AUT [3]. It contains methods to reproduce all user actions that can be performed on Swing/AWT GUI components, such as pushing buttons or typing text. The reflection API is used to identify and access GUI components defined inside the software under test, it permits the development of a test harness capable of generating application-specific GUI events, capturing the responses to the events, and verifying correct behavior.

In this paper, we worked on the specification language used for testing in the current state of the art to be written as JAVA Scripting which provides the specification language with standardized format, ease of use and small learning time. Since specification language is made of fully JAVA Script compliant, it takes advantage of existing JAVA Script development tools such as Visual editor for modeling, editing, transforming, and debugging scripting technologies. Next we provide a generator that converts test specification written in Java into a program which is used to generate events that implement user test cases given in specification script. The generator also contains a test oracle to verify that correct response occur and produce a log file for the given test script. Then we provide a visual tool to automatically generate specification script from an already existing AUT that helps the test designer to test already existing applications, as he has to edit only the user actions to be executed and its expected states using a visual hierarchy configuration.

## 2. GUI TESTING

GUI testing is vital for quality assurance because the GUI tests are performed from the view of the end users of the application. Mostly, all the functionality of the application can be invoked through the GUI and therefore GUI tests can cover the entire application [8]. Because manual testing of GUI software is tedious and laborious; there is a great need for reducing the high costs by means of automated GUI testing. The most popular tools used to test GUIs are Capture/Playback tools [8]. Some tools record mouse coordinates of the user actions as test cases. The problem with

such tools is that even a minor modification in the GUI breaks the corresponding test cases. One approach to overcome this problem is to capture GUI widgets rather than mouse coordinates. Although replay is automated, significant effort is required to create the test cases and to detect the failures. Another popular approach is to invoke the methods of underlying code as if initiated by the GUI [9]. Due to GUI software being created using rapid prototyping, the GUI is constantly changing during the development, making maintenance of capture/Playback or test scripts very expensive [10]. Therefore the current GUI testing techniques used in practice are incomplete, and a substantial amount of manual effort is required from the test designer [11]. A number of research results have shown AUT as a promising solution to overcome the maintenance weakness of capture and Playback tools [12].

## 2.1 Various Open Source GUI Testing Tools

**Abbot** is a framework for GUI testing. Its basic functionality allows a developer to write GUI unit tests in the form of Java methods which call the Abbot framework to drive an application's GUI. Besides tests written in Java, Abbot also allows the specification of tests in the form of XML test scripts. It provides a script editor called Costello for editing such scripts. Besides the manual editing of test scripts, Costello also supports the recording of scripts by capturing the events occurring in a running application [14].

**Jacareto** is a GUI captures and replay tool supporting the creation of animated demonstrations, the analysis of user behavior, as well as GUI test automation. Given this broad spectrum of applications, Jacareto provides a number of extra features, such as the highlighting of specific components in the GUI, extension points to capture and replay application-specific semantic events, or the embedding of Jacareto into the GUI application for providing macro-record and replay functionality. Jacareto comes with two front-ends, CleverPHL, a graphical tool with extensive support for recording, editing, and replaying interaction scripts and Picorder [15].

**Pounder** is exclusively focused on capturing and replaying interactions for GUI testing. It stores interaction scripts as XML files and is not intended to be used for manually writing tests. Compared with Abbot and Jacareto, Pounder is a lightweight tool, as can be seen by its narrow focus and its small size [16].

**Marathon** seems to be an open-source version of a more powerful commercial product. Besides providing a recorder and a player, Marathon also comes with an extensive editor for interaction scripts. Marathon records interaction logs as Python scripts [18].

**JFC Unit** is an extension that enables GUI testing based on the JUnit testing framework. JFC Unit allows a developer to write Java GUI tests as JUnit test case methods. The main focus of JFC Unit is the manual creation of GUI tests (following JUnit's approach), but a recording feature has been added in a recent version [17].

**Table 1 Open Source GUI Tools**

| | TOOL | | | | |
|---|---|---|---|---|---|
| Fields | Abbot | Jacareto | Pounder | JFC | M.thon |
| Text field | • | • | • | | |
| MouseMove | • | • | • | | |
| MouseDrag | • | • | • | | |
| MouseClick | | • | • | | |
| Component | • | • | • | | |
| Scrolling | • | • | • | | |
| FileDialog | | • | • | • | |
| ComboBox | | • | • | • | • |
| Timing | | | | | |

The coverage of various testable items in the above discussed open source GUI Tools are summarized in Table 1[6]

## 2.2Automated Modeling and Testing of Java GUIs

The GUI Driver tool that we implemented is a proof-of-concept and a tool chain for automated modeling and testing of Java GUI applications. Our aim is to reduce the manual effort required to create models for GUI testing by providing tool support for automatically generating models suitable for AUT. In our approach, the models are generated while automatically executing and observing the AUT. The generated models include structural tree models presenting the GUI components and their properties, and a GUI state model presenting the behavior of the GUI and mapping the structural models into the abstract states. The Application are generated for each state of the GUI application and saved into JAVA Scripting files. Comparing two consecutive structural models makes it possible to observe the changes happening in the GUI, while automatically interacting with the GUI application. The GUI state model represents the behavior of the GUI application as a state machine, presenting GUI states as nodes and interactions between the states as edges. The structural models are automatically mapped into the abstract states of the GUI state model. For the GUI state model, we also provide human readable graphical representation to allow checking the implementation against the requirements of the system. Normally, AUT based on the requirements and automatically generating a test suite based on that model to check whether it fulfils the requirements. In our approach the generated models are based on the actual implementation, so the generated GUI state model should be compared with the requirements of the system. Another goal for the graphical modeling is to allow manual elaboration of the model, for example, adding valid and invalid input values for text fields of the GUI. Also, automatically generated graphical models can help developers to understand and analyze an existing implementation if proper models of the system are missing.

An example of the GUI Tool model is shown in Figure 1.

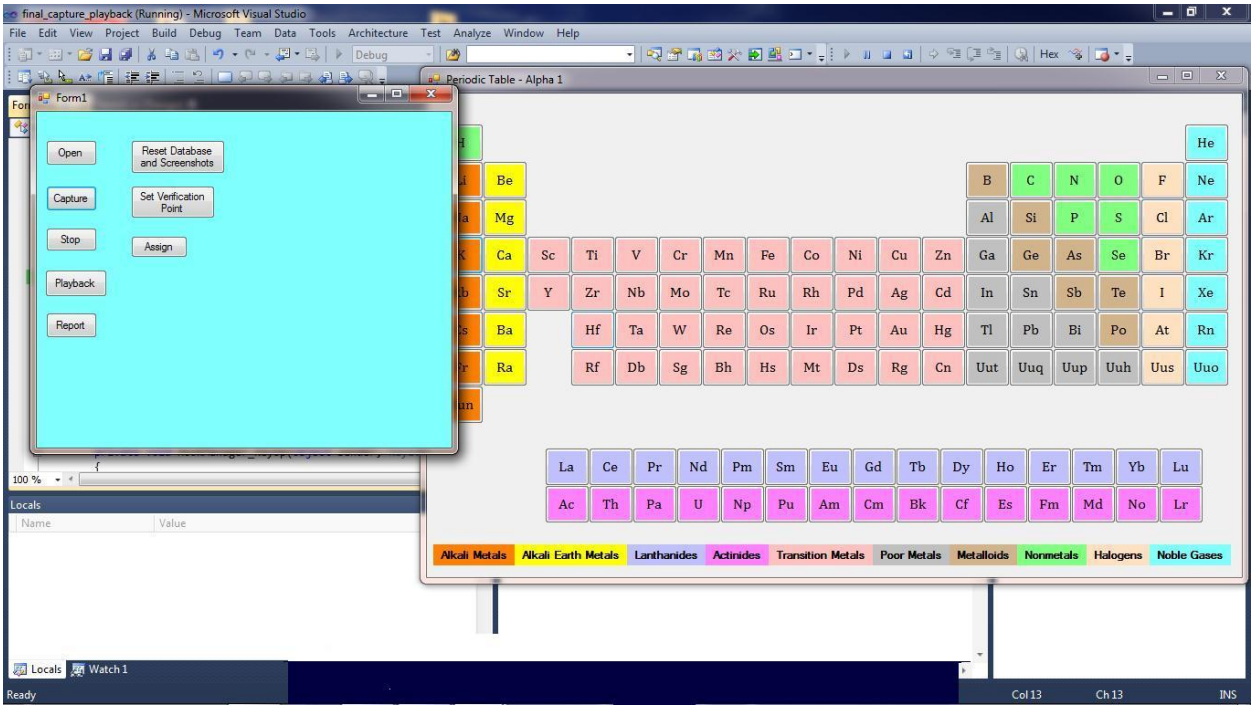


**Figure 1: Generated GUI Application model displayed by GUI Testing tool.**

## 3. GUI TOOL ARCHITECTURE

We have implemented the GUI tool to support our approach. It is a tool for programmatically driving a Java GUI application and generating models representing the states and behavior of the GUI. The created models are then used for AUT purposes. In the high level architecture of the GUI Tool, presented in Figure 2, the main parts of the tool are GUI Tool, Event Recorder, Event Dispatcher, Method invoker, Event log, Call Log .

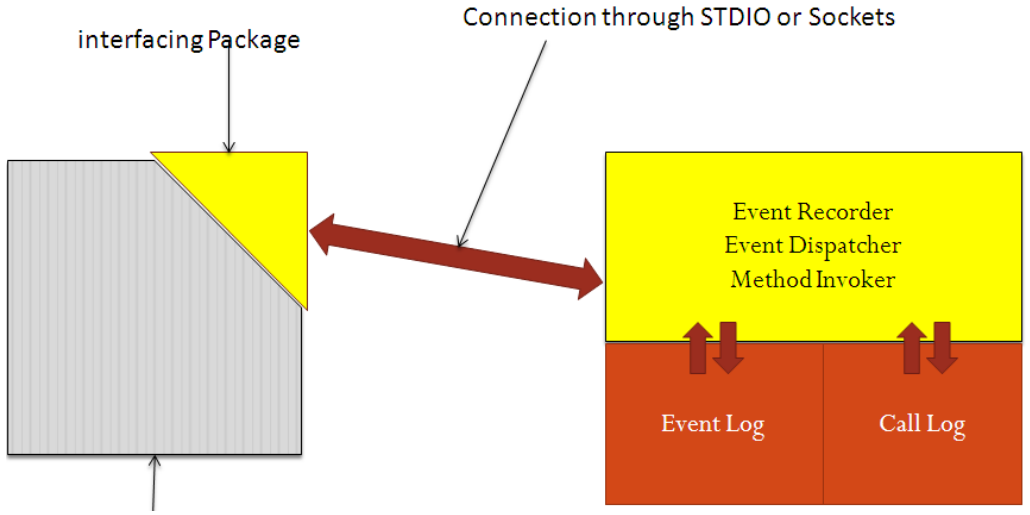


**Figure 2: High level architecture of the GUI Tool.**

After starting the AUT, the GUI Tool identifies the accessible and focused GUI window of the Java GUI application, creates a structural model of the window including the GUI components and their properties, and saves the model in an Log file. Then, it identifies the enabled and visible interactions, selects and executes one of the available GUI actions and creates a new structural model after each executed GUI action. In case of selecting exit or close action during the execution, the AUT is restarted. The Model Generator observes the behavior of the GUI, checks if a specific GUI state is a new one or one of the old states, and creates a state model of the GUI application suitable for AUT. Also, the generated structural models are mapped to the abstract states of the GUI state model. Currently, the GUI state model is

saved in JASON format that is applicable for the generation of test sequences with an open source AUT tool [13].The Test Executor is able to parse test sequences generated by the AUT tool [13], execute the test sequences on the AUT, verify whether the expected states were reached after executing the specified GUI actions, and create a test report including information about executed GUI actions, reached GUI states, and abnormal behavior like unhandled exceptions.

## 3.1 Automatic selection of GUI actions

The GUI Tool uses certain rules and preferences for selecting one of the GUI actions provided by the AUT. The default way is to select one of the GUI actions that have not been selected in that GUI state earlier.

The State-based logging: in this logging type the start and the end time of each event that uniquely define a state are stored in the log file .This file type contains a set of interval records each one of them is characterized as 'begin interval', 'end interval', 'continuation interval' and 'complete interval'. The enabled actions of consecutive states are compared and new GUI actions are preferred over the ones that were available in the previous state. That way after opening a drop-down menu it is probable that one of the actions of that menu is selected. If there are many evenly preferable actions, one is selected randomly.

## 3.2 Identifying the visited GUI states.

After generating a structural model of the AUT, the GUI tool compares the reached GUI state to the states visited earlier to check, whether the state is a new one or one of the already visited GUI states. The first criteria for a GUI state to be considered the same as one of the already visited states is that both states must have the same enabled GUI actions. The secondary generates the test cases of the Application. The number of the structural changes that are tolerated for similar GUI states can be specified in the settings.

## 3.3 Test-Suite Reduction

Test-suite reduction may be divided into two kinds in more detail: reduced test suite and minimized test suite. Minimized test suite is a test suite that cannot be reduced any more. If we use the heuristics algorithm is to reduce the test suite shown. But this heuristics algorithm only considers coverage degree of test case for test requirements, and does not consider the characteristic of MC/DC, moreover, It does not consider the capability of test case to reveal error [4]

## 3.4 GUI Test case Generation.

The planning has also been used to manage the state space explosion by eliminating the need for explicit states. A description of the GUI is manually created by a tester; this description is in the form of planning operators, which model the preconditions and effects (post conditions) of each GUI event. Test cases are automatically generated from tasks (pairs of initial and goal states) by invoking a planner which searches for a path from the initial to the goal state.[7] System data is a necessary ingredient for testing since internal data is the base of any ERP system and will most likely be processed during any execution.[5]

## 4. Conclusion and Future Enhancement

In this paper we presented an approach for automatic modeling of Java GUI applications for Application Under testing (AUT) purposes, implemented proof-of-concept tool support for the approach, and combined the implemented GUI tool with an open source AUT tool to form a tool chain for automated modeling and testing of Java GUI applications. Our approach aims to reduce the amount of manual effort required to model GUI applications to enable automated testing. The strengths of our approach in comparison to the automated testing tools include automatically generating human readable graphical models while requiring none or only a little manual effort.

The graphical models provides the test engineers, a way to manually elaborate the models, for example inserting valid input values for specific input fields of the GUI application, and allow comparison between testing tools based on the actual implementation and requirements of the system. The tool chain was used to automatically model and test several open source Java GUI applications, resulting in the breakthrough of several unknown errors and usability problems.

The approach seems hopeful but there are also limitations. As our approach is based on running and observing existing software, the AUT must be an executable Java GUI application, and the models are based on the actual implementation instead of designed and expected features.

Also, more work is needed on the GUI Tool, as the proof-of-concept implementation is not yet mature enough to be used in the software industry.

In future, we plan to improve the GUI Tool so that the generated models and reports would inform about the detected usability issues and include information about the changes that happened in the GUI after a specific interaction. The GUI Tool should indicate more clearly the states that should be manually elaborated in the model and support iterative modeling containing manual and automated phases. Also, we plan to extend the approach to be also usable on other kinds of GUI applications.

## 5. REFERENCES

[1] A. M. Memon, M. E. Pollack, and M. L. Soffa. "Hierarchical GUI Test Case Generation using Automated Planning", IEEE Transactions on Software Engineering, 27(2), February 2001, pp.144-155.

[2] Y. Sun and E. L. Jones, "Specification-Driven Automated Testing",Proceedings of the 42nd Annual Southeast Regional Conference, pp 140-145, 2004.

[3] Woei-Kae Chen, Tung-Hung Tsai and Hung-Hsing Chao "Integration of Specification-based and CR-based Approaches for GUI Testing", Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05), pp 1-6, 2005

[4] J.Prabhu, N.Malmurugan, G.Gunasekaran, R.Gowtham" Study of ERP Test-Suite Reduction Based on Modified Condition/Decision Coverage", proceedings of the Second International Conference on Computer Research and Development (ICCRD) pp373-378,2010

[5] J.Prabhu, N.Malmurugan " Finding Bugs in ERP Application Using Test Data", International Journal of Computer Theory and Engineering, Vol. 3, No. 5, October 2011,pp 690-696

[6] J.Prabhu, N.Malmurugan " A Survey on Automated GUI Testing Procedures", European Journal of Scientific Research, Vol.64 No.3 (2011), pp. 456-462

[7] Xun Yuan, Atif M. Memon, "Generating Event Sequence-Based Test Cases Using GUI Runtime State Feedback" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 36, NO. 1, JANUARY/FEBRUARY 2010, pp. 81-95

[8] K. Li and M. Wu, "Effective GUI Test Automation: Developing an Automated GUI Testing Tool", SYBEX Inc.,Alameda, CA, 2004.

[9] A. M. Memon, "Automatically repairing event sequence based GUI test suites for regression testing", ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 18, Issue 2 (November 2008), Article No. 4

[10] M. Grechanik, Q. Xie, and C. Fu, "Maintaining and Evolving GUI-Directed Test Scripts", Proceedings of the 2009 IEEE 31st International Conference on Software Engineering (ICSE 2009), pp. 408-418.

[11] A. M. Memon, "An event-flow model of GUI-based applications for testing", Software Testing, Verification & Reliability, Volume 17, Issue 3 (September 2007), pp. 137 -157.

[12] M. Utting and B. Legeard, "Practical Model-Based Testing: A Tools Approach", Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 2006.

[13] http://mbt.tigris.org/

[14] http://www.abbot.sourceforge.net/

[15] http://www.jacareto.sourceforge.net/

[16] http://www.pounder.sourceforge.net/

[17] http://www.jfcunit.sourceforge.net/

[18] http://www.marathontesting.com/