PBGT Tool: An Integrated Modeling and Testing Environment for Pattern-Based GUI Testing

Rodrigo M. L. M. Moreira
INESC TEC & Dept. of Informatics Engineering
Faculty of Engineering of the University of Porto
Porto, Portugal
pro08007@fe.up.pt

Ana C. R. Paiva
INESC TEC & Dept. of Informatics Engineering
Faculty of Engineering of the University of Porto
Porto, Portugal
apaiva@fe.up.pt

ABSTRACT

Pattern Based GUI Testing (PBGT) is a new methodology that aims at systematizing and automating the GUI testing process. It is supported by a Tool (PBGT Tool) which provides an integrated modeling and testing environment that supports the crafting of test models based on UI Test Patterns, using a GUI modeling DSL called PARADIGM. The tool is freely available as an Eclipse plugin, developed on top of the Eclipse Modeling Framework. This paper presents PBGT Tool, which has been successfully used in several projects, and more recently at industry level.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Reliability/Verification*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing Tools*; D.2.13 [**Software Engineering**]: Reusable Software—*Reuse models*

Keywords

Model-Based GUI Testing; Pattern-Based GUI Testing; GUI Modeling; GUI Testing

1. INTRODUCTION

Nowadays the majority of software applications feature a Graphical User Interface (GUI). GUIs have widespread and consistently represent an important role in the success of the software. The sequence of events that users can perform in a GUI are enormous. Thus, it is important to test GUIs for their functional correctness. GUIs are typically implemented using UI Patterns [15, 8]. UI Patterns are recurring solutions of a given functionality allowing to solve common GUI design problems. Depending on the desired purpose, each UI Pattern can be implemented differently. For instance, the authentication functionality is a popular representation of the Login UI Pattern. A typical implementation of this UI Pattern includes two input fields – username and password – and a submit button. In addition, the submit button can be accompanied by other buttons (actions) such as "Clear" and options like "Keep me signed in". The submission of this data can trigger several outcomes. Upon authentication failure, the system may display an error message in a particular area of the form or display a pop-up window with the error message. The error message may also vary from "Login failed" to "Invalid username", among others. Concerning a successful logon, we still have different possible implementations, i.e., the end-user may be redirected to a different location or he can stay in the same location but with a different UI that enables higher privileged features that were not available before.

PBGT is a new model-based GUI testing approach that provides generic test strategies (UI Test Patterns [8]), with possible different configurations for testing different implementations of a UI Pattern. A UI Test Pattern is able to test a GUI that was implemented using a set of UI Patterns. This testing methodology is supported by a tool. These UI Test Patterns are defined within a domain specific language, PARADIGM, developed specifically for the PBGT context.

This paper presents the PBGT tool (a video demonstration can be found in `http://www.fe.up.pt/~apaiva/tools/PARADIGM5min.mp4`), which is a fully integrated testing environment as it provides functionalities for modeling (either manually or automatically), configuration, automated test case generation, automated test case execution and test coverage analysis. This tool has been used to conduct several case studies [8, 7, 2, 13, 9] and within few hours, testers were able to find failures in the underlying software. Furthermore, studies [7] indicate that models written in PARADIGM require less effort than building the same models in Spec#[1] and VAN4GUIM [6]. At this moment, the tool is able to target web and mobile (Android based) applications.

The rest of the paper is structured as follows. Section 2 presents an overview of PBGT: the PARADIGM language, the PBGT components and the PBGT process describing scenarios upon the usage of the tool. Section 3 presents background information and related work (tools). Section 4 discusses the potential impact of the environment to the broader community. Section 5 draws conclusions and points future work directions.

2. OVERVIEW: PBGT

Pattern Based GUI Testing (PBGT) is a new emerging model-based GUI testing approach that aims at systematiz-

[1]http://research.microsoft.com/en-us/projects/specsharp/

ASE'14, September 15-19, 2014, Vasteras, Sweden.

ACM 978-1-4503-3013-8/14/09 ...$15.00.
http://dx.doi.org/10.1145/2642937.2648618.

ing and automating the GUI testing process, by benefiting from UI Test Patterns and therefore promoting reuse of GUI testing strategies. UI Test Patterns are elements within a Domain Specific Language (PARADIGM) from which it is possible to build models describing the GUI testing goals in a higher level of abstraction. PBGT has the ability to automatically generate test cases from those models and execute them over a GUI.

2.1 Components

PBGT is supported by a tool, built on the top of the Eclipse Modeling Framework (EMF), comprised by five main components:

- **PARADIGM** – a domain specific language (DSL) for building GUI test models based on UI Test Patterns;
- **PARADIGM-RE** – a reverse engineering component whose purpose is to extract PARADIGM models from web pages, without requiring access to the source code;
- **PARADIGM-TG** – an automated test case generation component that builds test cases from PARADIGM models;
- **PARADIGM-TE** – a test case execution component that analyzes their coverage (details about coverage can be found in [16]) and returns detailed execution reports;
- **PARADIGM-ME** – a modeling environment that supports the building and configuration of test models.

2.2 Language

PBGT requires a GUI model written in PARADIGM. The goal of this DSL is to gather applicable domain abstractions, i.e., UI Test Patterns, allowing to specify relations between them and also provide a way to structure models in different levels of abstraction, in order to deal with complexity.

PARADIGM was built having reusability as one of its forces, allowing UI Test Patterns to be adapted for testing different UI Patterns implementations after a configuration step. Further, this language also promotes reusability by reusing existing elements or extending them to be reused by others. This DSL was developed having as basis a set of guidelines and best practices [1, 14].

Elements

The PARADIGM metamodel is illustrated in Figure 1. An *Element* is an abstract entity that represents the concepts within PBGT domain. A model written in PARADIGM begins with the *Init* element and finishes with the *End*. These elements are specializations of *Element*. PARADIGM models can be structured in different levels of abstraction in order to deal with complexity and to facilitate organization. Structural *Form* elements were created particularly for this purpose. A *Form* embodies a model (or sub-model), with an *Init* and *End* elements. *Groups* are also structural elements but they are used to gather elements that may be executed arbitrarily. *Behavioral* elements represent the UI Test Patterns that define generic test strategies for testing UI Patterns.

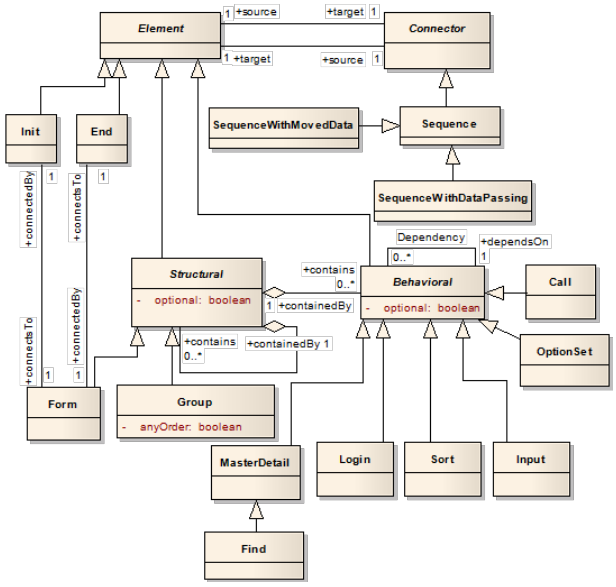

Figure 1: PARADIGM Language metamodel.

Connectors

PARADIGM language defines connectors that are inspired from "ConcurTaskTrees" (CTT) [12], namely: (1) "*Sequence*" – indicates that the test strategy of the target element cannot start before finishing the tests of the source element; (2) "*SequenceWithDataPassing*" – has the same behavior as "*Sequence*" and, additionally, indicates that the target element receives data from the source element; and (3) "*SequenceWithMovedData*" – which has a similar meaning to the "*SequenceWithDataPassing*" connector, yet, the source element transfers data to the target, so the source loses the data that was transferred. "Dependency" is another type of relation indicating that the target element depends on the properties of a set of source elements.

UI Test Patterns

UI Test Patterns provide a generic test strategy to test possible different implementations of UI Patterns. UI Test Patterns correspond to the Behavioral elements (Figure 1) of the PARADIGM DSL. In [5] these UI Test Patterns were described according to best practices from the field of Pattern Languages.

A UI Test Pattern describes a generic test strategy [8], formally defined by a set of test goals, for later configuration, denoted as

$$< Goal, V, A, C, P >$$

Goal is the ID of the test. **V** is a set of pairs {[variable, inputData]} relating test input data with the variables involved in the test. **A** is the sequence of actions to perform during test case execution. **C** is the set of possible checks to perform during test case execution. Finally, **P** is a Boolean expression (precondition) defining the conditions over variables that determine when it is possible to execute the test strategy.

UI Test Patterns are divided into two categories: (1) Base UI Test Patterns that represent the initial set of the patterns and; (2) High-Order UI Test Patterns are built from the combination of two or more Base UI Test Patterns. High-

Order UI Test Patterns promote reuse even further and allow the definition of libraries of test patterns that may be useful for specific contexts.

The PARADIGM language has seven Base UI Test Patterns:

- **Input** – this pattern should be used to test the behavior of input fields for valid and invalid input data;
- **Login** – this pattern should be used to verify user authentication. The goal is to check if it is possible to authenticate with a valid username/password and check if it is not possible to authenticate otherwise;
- **Master/Detail** – this pattern should be used to test two related objects (master and detail) in order to verify if changing the master's value, correctly updates the contents of the detail;
- **Find** – the purpose of this pattern is to check if the result of a search is the correct set of values;
- **Sort** – this pattern is used to check if the result of a sort action is ordered accordingly to the chosen sort criteria;
- **Call** – this pattern should be used to check the behavior of the corresponding call, for instance, check if a link opens a new page;
- **Option Set** – this pattern should be used to check the expected behavior for multiple selections.

Constraints

Besides the syntax (elements and connectors), the PARADIGM models have to be built according to a set of constraints that cannot be expressed directly in the UML model. In order to find and tune those constraints, we translated the PARADIGM UML metamodel (Figure 1) to Alloy [3], so we could analyze the generated instances (that should represent possible valid GUI test models). This was an iterative process ([7]) that ended when the instances obtained from the Alloy model corresponded to valid well-formed PARADIGM models.

2.3 Process

The PBGT process sets a total of six main steps: (1) modeling; (2) configuration; (3) test case generation; (4) test case execution; (5) results analysis and (6) model update (when required). Both test case generation and execution are fully automated steps. The modeling phase can be performed manually from scratch (in a forward software development process) or it can be performed automatically, to obtain part of the model by a reverse engineering process from an existing software application under test. This is done by the PARADIGM-RE component that explores the application under test and infers the set of existing UI Patterns building, afterwards, a PARADIGM model with the UI Test Patterns that are appropriate to test them.

In a forward development process, the tester may start building the application test model during requirements elicitation and evolve such model along the development. Then, the tester configures each UI Test Pattern with the required data (input data, preconditions, and checks to perform during test execution). Afterwards, the PARADIGM-TG component generates the test cases calculating first all the paths between *Init* and *End* elements within the model. From those paths, test cases are generated considering the configurations provided previously by the tester. To execute the tests, it is necessary to establish previously a mapping between the model UI Test Patterns and the UI Patterns of the web application under test which will allow to identify the web elements that correspond to a certain UI Test Pattern to interact with during test case execution.

The mapping is established by the tester through a pointing and click process. Along this process the information of the web elements is saved by Selenium Web Driver[2]. Besides some properties, the PBGT tool saves also images of the elements through Sikuli[3] and their area coordinates to cope with situations where it is not possible to identify those elements through their properties.

Once the mapping is established, PARADIGM-TE executes the generated test cases over the application under test and produce reports with the results of the tests.

3. COMMUNITY IMPACT

The PBGT Tool has recently been released to public. One of the goals of PBGT is to promote the adoption of model-based testing process into the industry. The following aspects can be seen as forces that, in our perspective, and based on received feedback, contribute for its adoption:

- **Reusability concerns** – UI Test Patterns can be reused during the GUI modeling and testing process;
- **Reduced efforts** – when compared with other GUI modeling approaches, models can be crafted and configured in short time [7];
- **Goal focus** – typical Model-Based GUI Testing tools are centered in modeling the behavior of the application. With PBGT the focus is directed towards modeling testing goals;
- **Platform independent** – PBGT Tool can be used to model and test web applications and also mobile applications [8, 2];
- **No source code is required** – PBGT Tool does not require access to the source code of the systems under test, in order to create or generate GUI models from them;
- **Low maintenance and evolutionary** – With few steps it is possible to extend the initial set of UI Test Patterns, and also to adjust current test strategies (or create new ones) to support new UI trends;
- **Simple to use** – With few knowledge on testing activities, users can start modeling and testing software in short time.

4. RELATED WORK

Several model-based tools already exist for GUI testing. However, the majority was not built with reusability concerns from the start. Furthermore, some do not provide extension mechanisms. Others are only able to generate the

[2]http://docs.seleniumhq.org/projects/webdriver/
[3]http://www.sikuli.org/

model only if the GUI was implemented in a specific language.

GUITAR [10] is one popular tool for MBGT. Models are generated directly from an executable using this tool. It uses Event Flow Graphs (EFG) to create a GUI model, capturing the flow of events, featuring all possible event interactions in the UI. As best of our knowledge, no tool exists to model an EFG by hand so it is not possible to edit the ripped models. For complex applications GUI ripping falls short in generating the full model.

GUIs can be modeled with Spec#, which is a specification language developed by Microsoft Research. GUI Spec# models are the input to Spec Explorer [4], an advanced model-based testing tool. It automates the generation and execution of test cases. The GUI mapping tool [11], is an extension of the Spec Explorer tool for model-based GUI testing. It aims to automatically generate a .Net assembly with methods that simulate user actions upon the GUI application under test. In order to test GUIs, it requires considerable effort to map user actions from the model to real actions in the GUI.

Other approach uses finite-state machines (FSMs), using a GUI Test Automation Model (GuiTam) [4] to automatically create the state models using dynamic analysis.

5. CONCLUSIONS AND FUTURE WORK

This paper introduced the PBGT Tool, a new model-based GUI testing tool, which provides an integrated modeling and testing environment. The tool has been used in several projects, and at industry level [7].

The seven UI Test Patterns described in the paper, have proven to be enough to model a wide range of web applications and lately mobile applications. However, considering the latter, we aim to increase the set of UI Test Patterns in order to test specific mobile events (e.g., swipe). Furthermore, at this moment, PBGT tool cannot test desktop and iOS[5] applications. We aim to develop and integrate drivers for that purpose in the near future.

We also intend to promote the tool, by advertising and by conducting on-site demos, so it could facilitate further industry adoption.

6. ACKNOWLEDGMENTS

This work is financed by the ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-020554.

7. REFERENCES

[1] Design Guidelines for Domain Specific Languages. In M. Rossi, J. Sprinkle, J. Gray, and J.-P. Tolvanen, editors, *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*, pages 7–13, 2009.

[2] P. Costa, M. Nabuco, and A. C. R. Paiva. Model-based testing for Mobile Applications. In *The 9th International Conference on the Quality of Information and Communications Technology, QUATIC*. IEEE Computer Society, 2014.

[3] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press; 2nd Revised edition, 2011.

[4] Y. Miao and X. Yang. An FSM based GUI test automation model. In *Control Automation Robotics Vision (ICARCV), 2010 11th International Conference on*, pages 120–126, Dec 2010.

[5] R. Moreira and A. Paiva. Towards a Pattern Language for Model-Based GUI Testing. In *Proceedings of the 19th European conference on Pattern Languages of Programs (EuroPLoP)*, 2014.

[6] R. M. L. M. Moreira and A. C. R. Paiva. Visual Abstract Notation for GUI Modelling and Testing – VAN4GUIM. In *ICSOFT (SE/MUSE/GSDCA)*, pages 104–111. INSTICC Press, 2008.

[7] R. M. L. M. Moreira and A. C. R. Paiva. A GUI Modeling DSL for Pattern-Based GUI Testing - PARADIGM. In L. A. Maciaszek and J. Filipe, editors, *ENASE*. SciTePress, 2014.

[8] R. M. L. M. Moreira, A. C. R. Paiva, and A. Memon. A Pattern-Based Approach for GUI Modeling and Testing. In *Proceedings of the 24th International Symposium on Software Reliability Engineering*, ISSRE'13, Pasadena, CA, USA, 2013. IEEE Computer Society.

[9] M. Nabuco, A. C. R. Paiva, and J. P. Faria. Inferring User Interface Patterns from Execution Traces of Web Applications. In *Software Quality workshop of the 14th International Conference on Computational Science and Applications (ICCSA)*, 2014.

[10] B. N. Nguyen, B. Robbins, I. Banerjee, and A. M. Memon. Guitar: an innovative tool for automated testing of GUI-driven software. *Autom. Softw. Eng.*, 21(1):65–105, 2014.

[11] A. C. R. Paiva, J. C. P. Faria, N. Tillmann, and R. F. A. M. Vidal. A Model-to-Implementation Mapping Tool for Automated Model-Based GUI Testing. In K.-K. Lau and R. Banach, editors, *ICFEM*, volume 3785 of *LNCS*, pages 450–464. Springer, 2005.

[12] F. Paternò, C. Mancini, and S. Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*, INTERACT '97, pages 362–369, London, UK, UK, 1997. Chapman & Hall, Ltd.

[13] C. Sacramento and A. C. R. Paiva. Web Application Model Generation through Reverse Engineering and UI Pattern Inferring. In *The 9th International Conference on the Quality of Information and Communications Technology, QUATIC*. IEEE Computer Society, 2014.

[14] M. Strembeck and U. Zdun. An approach for the systematic development of domain-specific languages. *Softw. Pract. Exper.*, 39(15):1253–1292, Oct. 2009.

[15] J. Tidwell. *Designing Interfaces*. O'Reilly, Sebastopol, CA, 2011.

[16] L. Vilela and A. C. R. Paiva. PARADIGM-COV - A Multimensional Test Coverage Analysis Tool. In *In 9ª Conferencia Ibérica de Sistemas y Tecnologías de Información (CISTI)*, 2014.

[4] http://research.microsoft.com/en-us/projects/specexplorer/

[5] http://ios-driver.github.io/ios-driver/