Web Application Model Generation through Crawling and UI Pattern Inferring

Clara Sacramento, Ana C. R. Paiva
Departamento de Engenharia Informtica
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
ei09090@fe.up.pt; apaiva@fe.up.pt;

***Abstract*—A great deal of effort in model-based testing is related to the creation of the model. The model itself, while a powerful tool of abstraction, can be a source of bugs. This paper presents a dynamic reverse engineering approach that aims to extract part of the model of an existing Web application through the identification of User Interface (UI) patterns. This approach explores the Web application via crawling, saves information related to the interaction (crawl history, HTML pages and their URLs), analyzes the gathered information, and infers the UI patterns via a set of heuristics rules.**

***Keywords*—Reverse Engineering, Web Application, UI Patterns, Web Scraping, Web Crawling**

I. Introduction

Web applications are getting more and more important, and can now handle tasks that before could only be performed by desktop applications [1], like editing images or creating spreadsheet documents. However, despite their growing relevance, they still suffer from a lack of standards and conventions [2], unlike desktop and mobile applications. This means that the same task can be implemented in many different ways, which makes automated testing difficult to accomplish and inhibits reuse of testing code.

GUIs (Graphical User Interfaces) of all kinds are populated with recurring behaviors that vary slightly. For example, authentication (*login/password*) is a common behavior in many software applications. These behaviors (patterns) are called User Interface (UI) patterns [3] and are recurring solutions that solve common design problems. Due to their widespread use, UI patterns allow users a sense of familiarity and comfort when using applications.

However, while UI patterns are familiar to users, their implementation may vary significantly. For a login, in some cases an error message may appear when the authentication fails; in others, the software application simply erases the inserted data and doesnt send a message to the user. Despite this, it is possible to define generic and reusable test strategies to test them after a configuration process to adapt the tests to those different possible applications [4].

That is the main idea behind the PBGT (*Pattern-based GUI Testing*) project, in which this research work is developed. In the PBGT approach, the user builds a test model containing instantiations of UI patterns, and later uses that model to test their occurrences on Web applications. The goal of the work described in this paper is to continue the work done in [5] on the extraction process (PARADIGM-RE), where a reverse engineering process was developed to automatically identify the presence of UI Patterns on existent Web applications, provided the user interacts manually with the Web application with Selenium IDE [1] recording his actions. The work described in this paper aims to automatize the process: crawl a Web application, search for UI patterns in its pages, and finally produce a model with the results of the search.

The rest of the paper is structured as follows. Section II presents an overview of the PBGT project, setting the context for this work. Section V addresses the related work, as well as the tools available to perform the needed tasks. Section III describes how the system was implemented. Section IV provides a practical example of the system proposed. Section **??** provides the conclusions, reports some of the problems found and points out the future work.

II. PBGT Overview

A. Architecture

As mentioned before, the focus of this article is a component of an investigation project named PBGT (*Pattern-based GUI Testing*) [6]. The goal of this investigation project is to develop a model-based GUI testing tool and approach, usable as an industrial tool. This project has five parts: a DSL (*Domain Specific Language*) named **PARADIGM** to define GUI testing models based on UI patterns; **PARADIGM-RE**, a Web application reverse engineering tool whose purpose is to extract UI patterns from Web pages without access to their source code, and use the extracted patterns to generate a test model defined in PARADIGM; a modeling and testing environment, named **PARADIGM-ME**, made to support the creation of test models; an automatic test case generation tool, named **PARADIGM-TG**, that generates test cases from test models defined in PARADIGM; and finally a test case execution tool, named **PARADIGM-TE**, which executes test cases, analyzes their coverage, and returns detailed execution reports. The architecture and workflow of the project can better be seen in Fig. 1. Activities with cogs are automatic, and activities with stickmen require user input.

B. Supported Patterns

The UI Patterns defined in the PARADIGM language, and therefore supported by PARADIGM-RE, are:

[1] Selenium IDE: http://docs.seleniumhq.org/docs/02_selenium_ide.jsp

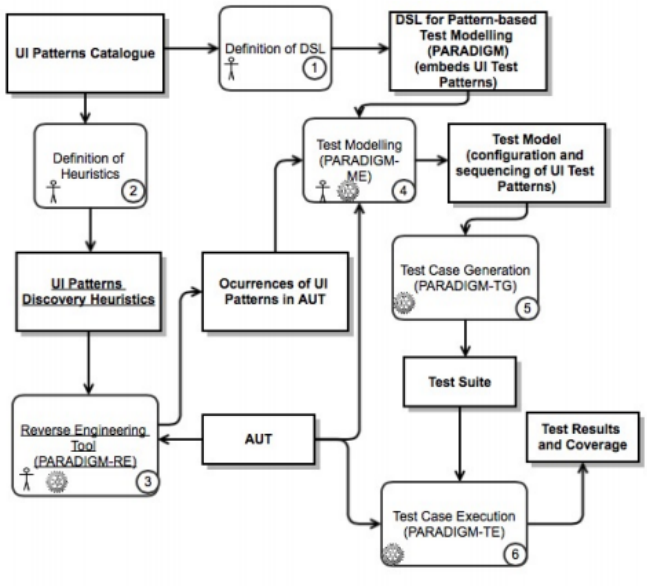

Fig. 1. An overview of the PBGT project [5]

Login

This pattern is commonly found in Web applications, especially in the ones that restrict access to functionalities or data. Usually consists of two input fields (a normal input box for email or username, and a cyphered text for the password) and a submit button, with optionally a "remember me" checkbox, to save the authentication data for the next visits. The authentication process has two possible outcomes: valid and invalid. Some websites may also include captcha verification upon too many failed login attempts.

Search

This pattern consists of one or more input fields, where the user inserts keywords to search, and a submit button to start the search. The search may be submitted dynamically, and the submit button be omitted. Upon success, the website shows a list of results; upon failure, an error message may be shown.

Sort

This pattern sorts a list of data by a common attribute (price, name and relevance) according to a defined criteria (ascending or descending, alphabetically, etc.). For example, in a Web store a user can sort a list of a specific type of products according to their price in order to identify the cheapest one.

Master Detail

This pattern is present in a webpage when selecting an element from a set results in filtering/updating another related set accordingly. For example, clicking on a checkbox associated to a brand may include (or exclude) products of that brand in a product search result list. The set of selectable elements is called *master* and the updated list is called *detail*. Generally the only elements updated are the elements belonging to the *detail* set.

Menu

This pattern is very common in webpages. It's usually defined as a tree structure with several navigational options, to provide easier access for users.

Input

This pattern is any kind of input field that allows the user to insert data. May be a text box or textarea element.

Call

This pattern is any kind of element where a click triggers a change of page. May be a link or a button.

C. Produced Models

The models produced by PARADIGM-RE after the RE process consist of a XML file that contains all the information about the UI Patterns found: their definition and configurations. The PARADIGM model generated by the RE tool does not contain the connectors between the UI Patterns, only the UI Patterns and their configurations. These models are meant to be loaded to the PARADIGM-ME tool and completed manually.

III. Crawling Approach

IV. Evaluation

V. Related Work

Reverse engineering is "the process of analyzing the subject system to identify the system components and interrelationships and to create representations of the system in another form or at a higher level of abstraction" [7]. There are two methods of applying reverse engineering to a system: the dynamic method, in which the data are retrieved from the system at run time without access to the source code, and the static method, which obtains the data from the system source code [8]. There is also the hybrid method, which combines the two previous methods, and the historical method, which includes historic information to see the evolution of the software system [9]. These approaches follow the same main steps: collect the data, analyze it and represent it in a legible way, and in the process allow the discovery of information about the system's control and data flow [10].

There are plenty of approaches that extract information from Web applications [11], [12], [13], [14]. ReGUI [15], [16] is a dynamic reverse engineering tool made to reduce the effort of modeling the structure and behavior of a software application GUI. Duarte, Kramer and Uchitel defined an approach for behavior model extraction which combines static and dynamic information [17].

In the area of Web crawling, there are many open source crawlers, the most notable being Apache Nutch [2], Bixo [3], Heritrix [4], Crawler4j [5], and Scrapy [6]. We must also consider

[2]Apache Nutch: http://nutch.apache.org/
[3]Bixo: http://bixo.101tec.com/
[4]Heritrix: http://bit.ly/1dYRV2n
[5]Crawler4j: http://code.google.com/p/crawler4j/
[6]Scrapy: http://scrapy.org

HTML parsers and extractors, to process the crawling results. Scrapy, being a crawling and scraping framework, counts on both lists. Other notable parsers are Mechanize [7], and Beautiful Soup [8].

There are also plenty of approaches that crawl a Web application, and in the process extract information. Crawljax [18] is a tool that obtains graphical site maps by automatically crawling through a Web application. Mesbah *et al.* proposed an automated technique for generating test cases with invariants from models inferred through dynamic crawling [19]. Another approach by Mesbah *et al.*, named FeedEx [20] is a feedback-directed Web application exploration technique to derive test models. It uses a greedy algorithm to partially crawl a RIA's GUI, and the goal is that the derived test model capture different aspects of the given Web application's client-side functionality. WebDiff [21] is a tool that searches for cross-browser inconsistencies by analyzing a website's DOM and comparing screenshots obtained in different browsers. Dincturk *et al.* [22] proposed a RIA crawling strategy using a statistical model based on the model-based crawling approach introduced in [23] to crawl RIAs efficiently. Dallmeier *et al.*'s Webmate [24], [25] is a tool that analyzes the Web application under test, identifies all functionally different states, and is then able to navigate to each of these states at the users request.

User Interaction (UI) patterns, in particular the ones supported by the tool, are well-documented in a various number of sources [26], [3], [27], [28]. Lin and Landay's approach [29] uses UI patterns for Web applications that run on PCs and mobile phones, and prompt-and-response style voice interfaces. Pontico *et al.*'s approach [30] presents UI patterns common in eGovernment applications.

Despite the fact that there are plenty of approaches to mine patterns from Web applications, no approaches have been found that infer UI patterns from Web applications beside the work extended in this paper [5], [4]. The approaches found deal mostly with Web mining, with the goal of finding relationships between different data or finding the same data in different formats. Brin [31] presented an approach to extract relations and patterns for the same data spread through many different formats. Chang [32] proposes a similar method to discover patterns, by extracting structured data from semi-structured Web documents. Freitag [33] proposed a general-purpose relational learner for information extraction from Web applications.

REFERENCES

[1] J. J. Garrett *et al.*, "Ajax: A new approach to web applications," 2005.

[2] L. L. Constantine and L. A. Lockwood, "Usage-centered engineering for web applications," *Software, IEEE*, vol. 19, no. 2, pp. 42–50, 2002.

[3] M. Van Welie, G. C. Van Der Veer, and A. Eliëns, "Patterns as tools for user interface design," in *Tools for Working with Guidelines*. Springer, 2001, pp. 313–324.

[4] I. C. Morgado, A. C. Paiva, J. P. Faria, and R. Camacho, "Gui reverse engineering with machine learning," in *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2012 First International Workshop on*. IEEE, 2012, pp. 27–31.

[5] M. Nabuco, A. C. Paiva, R. Camacho, and J. P. Faria, "Inferring ui patterns with inductive logic programming," in *Information Systems and Technologies (CISTI), 2013 8th Iberian Conference on*. IEEE, 2013, pp. 1–5.

[6] R. M. Moreira, A. C. Paiva, and A. Memon, "A pattern-based approach for gui modeling and testing," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*. IEEE, 2013, pp. 288–297.

[7] E. J. Chikofsky, J. H. Cross *et al.*, "Reverse engineering and design recovery: A taxonomy," *Software, IEEE*, vol. 7, no. 1, pp. 13–17, 1990.

[8] T. Systä, "Dynamic reverse engineering of java software," in *ECOOP Workshops*, 1999, pp. 174–175.

[9] G. Canfora, M. Di Penta, and L. Cerulo, "Achievements and challenges in software reverse engineering," *Communications of the ACM*, vol. 54, no. 4, pp. 142–151, 2011.

[10] M. J. Pacione, M. Roper, and M. Wood, "A comparative evaluation of dynamic visualisation tools," in *10th Working Conference on Reverse Engineering*, 2003, pp. 80–89.

[11] S. Elbaum, S. Karre, and G. Rothermel, "Improving web application testing with user session data," in *Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society, 2003, pp. 49–59.

[12] S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A. S. Greenwald, "Applying concept analysis to user-session-based testing of web applications," *Software Engineering, IEEE Transactions on*, vol. 33, no. 10, pp. 643–658, 2007.

[13] D. Amalfitano, A. R. Fasolino, and P. Tramontana, "Rich internet application testing using execution trace data," in *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*. IEEE, 2010, pp. 274–283.

[14] I. Andjelkovic and C. Artho, "Trace server: A tool for storing, querying and analyzing execution traces," in *JPF Workshop, Lawrence, USA*, 2011.

[15] I. Coimbra Morgado, A. Paiva, and J. Pascoal Faria, "Reverse engineering of graphical user interfaces," in *ICSEA 2011, The Sixth International Conference on Software Engineering Advances*, 2011, pp. 293–298.

[16] I. Coimbra Morgado, A. C. Paiva, and J. Pascoal Faria, "Dynamic reverse engineering of graphical user interfaces," *International Journal On Advances in Software*, vol. 5, no. 3 and 4, pp. 224–236, 2012.

[17] L. M. Duarte, J. Kramer, and S. Uchitel, "Model extraction using context information," in *Model Driven Engineering Languages and Systems*. Springer, 2006, pp. 380–394.

[18] D. Roest, "Automated regression testing of ajax web applications," Master's thesis, Delft University of Technology, February 2010.

[19] A. Mesbah, A. van Deursen, and D. Roest, "Invariant-based automatic testing of modern web applications," *Software Engineering, IEEE Transactions on*, vol. 38, no. 1, pp. 35–53, 2012.

[20] A. M. Fard and A. Mesbah, "Feedback-directed exploration of web applications to derive test models," in *Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering (ISSRE). IEEE Computer Society*, 2013, p. 10.

[21] S. R. Choudhary, H. Versee, and A. Orso, "Webdiff: Automated identification of cross-browser issues in web applications," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–10.

[22] M. E. Dincturk, S. Choudhary, G. von Bochmann, G.-V. Jourdan, and I. V. Onut, "A statistical approach for efficient crawling of rich internet applications," in *Web Engineering*. Springer, 2012, pp. 362–369.

[23] K. Benjamin, G. Von Bochmann, M. E. Dincturk, G.-V. Jourdan, and I. V. Onut, *A strategy for efficient crawling of rich internet applications*. Springer, 2011.

[24] V. Dallmeier, M. Burger, T. Orth, and A. Zeller, "Webmate: a tool for testing web 2.0 applications," in *Proceedings of the Workshop on JavaScript Tools*. ACM, 2012, pp. 11–15.

[25] ——, "Webmate: Generating test cases for web 2.0," in *Software Quality. Increasing Value in Software and Systems Development*. Springer, 2013, pp. 55–69.

[26] J. Tidwell, *Designing interfaces*. O'Reilly, 2010.

[7] Mechanize (Python version): http://wwwsearch.sourceforge.net/mechanize/

[8] Beautiful Soup: http://bit.ly/1iVZlbl)

[27] T. Neil. 12 standard screen patterns. Accessed: 2014-01-22. [Online]. Available: http://designingWebinterfaces.com/designingWeb-interfaces-12-screen-patterns

[28] D. Sinnig, A. Gaffar, D. Reichart, P. Forbrig, and A. Seffah, "Patterns in model-based engineering," in *Computer-Aided Design of User Interfaces IV*. Springer, 2005, pp. 197–210.

[29] J. Lin and J. A. Landay, "Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 1313–1322.

[30] F. Pontico, M. Winckler, and Q. Limbourg, "Organizing user interface patterns for e-government applications," in *Engineering Interactive Systems*. Springer, 2008, pp. 601–619.

[31] S. Brin, "Extracting patterns and relations from the world wide web," in *The World Wide Web and Databases*. Springer, 1999, pp. 172–183.

[32] C.-H. Chang, C.-N. Hsu, and S.-C. Lui, "Automatic information extraction from semi-structured web pages by pattern discovery," *Decision Support Systems*, vol. 35, no. 1, pp. 129–147, 2003.

[33] D. Freitag, "Information extraction from html: Application of a general machine learning approach," in *AAAI/IAAI*, 1998, pp. 517–523.