Automatic Page Objects Generation with APOGEN

Andrea Stocco[1], Maurizio Leotta[1], Filippo Ricca[1], Paolo Tonella[2]

[1] DIBRIS – Università di Genova, Italy
[2] Fondazione Bruno Kessler, Trento, Italy
andrea.stocco@dibris.unige.it, maurizio.leotta@unige.it, filippo.ricca@unige.it, tonella@fbk.eu

Abstract. Page objects are used in web test automation to decouple the test cases logic from their concrete implementation. Despite the undeniable advantages they bring, as decreasing the maintenance effort of a test suite, yet the burden of their manual development limits their wide adoption. In this demo paper, we give an overview of APOGEN, a tool that leverages reverse engineering, clustering and static analysis, to automatically generate Java page objects for web applications.

1 Introduction and Motivation

Automated web test code created for tools such as Selenium[3] is renown for being difficult to maintain as the application under test evolves [1]. When the same functionality must be necessarily invoked within multiple test cases (e.g., user login), a major drawback is the duplication of code within the test suite.

Page objects can effectively improve the maintainability and longevity of a web test suite [1], because they hide the technical details about how the test code interacts with the web page behind a more readable and business-focused facade. Indeed, they can be considered as an API toward the web application: the web pages are represented as object-oriented classes, encapsulating the functionalities offered by each page as methods. In this way, the tests specification is well separated from their concrete implementation.

There are clear advantages stemming from the adoption of page objects within the test code [1]. However, their manual development is expensive and existing tools (e.g., *OHMAP*[4], *SWD Page Recorder*[5], or *WTF PageObject Utility Chrome Extension*[6]) offer poor assistance in the creation of the source code [3]. In short, most of the page objects development effort is still on the shoulders of the tester.

Our tool APOGEN [3, 4] is the first solution providing a considerable degree of automation, offering a more complete page objects generation tool, that can be used as a baseline to create well-architected, and thus more maintainable, web test suites.

The demo paper is organised as follows: Section 2 describes the high level architecture of APOGEN and illustrates its functioning from the user's perspective, by means of a running example. Conclusions are drawn in Section 3.

[3] http://www.seleniumhq.org/
[4] http://ohmap.virtuetech.de/
[5] https://github.com/dzharii/swd-recorder
[6] https://github.com/wiredrive/wtframework/wiki/WTFPageObjectUtilityChromeExtension

2 Tool Architecture and Running Example

We now explain the tool architecture, how a web tester can automatically generate page objects using APOGEN, and how such page objects are used for the construction of a web test case. Fig. 1 shows the high level architecture of APOGEN [4]. APOGEN has been developed in Java, making use of several external libraries and tools.

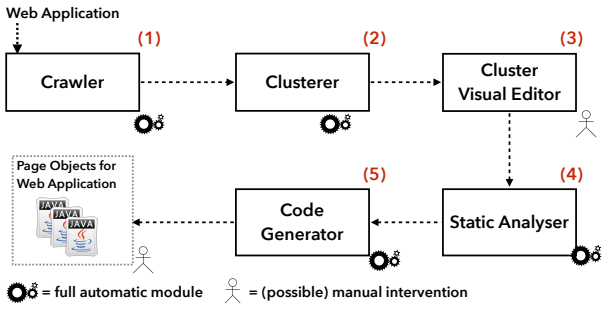

Fig. 1. High level architecture of APOGEN

The Crawler **(1)** is built on top of Crawljax [2], a state of the art tool for fully customisable exploration of highly-dynamic web applications. Since the model retrieved by the Crawler can be huge, the Clusterer **(2)** groups conceptually correlated web pages within the same cluster [4], using clustering algorithms available from the popular machine learning library WEKA [5].

The Clusters Visual Editor (CVE) **(3)** is a web-based tool developed using the D3 library[7]. It supports the tester with an interactive cluster visualisation and editor facility, allowing her to inspect and modify the clustering results. Indeed, CVE allows the tester to interactively move nodes to the cluster they should belong to, in order to manually refine the output of the Clusterer (see the stickman in Fig. 1).

The Static Analyser **(4)** uses JavaParser[8] and XMLUnit[9]. The former is used to gather information from the web pages Document Object Model (DOM) and build an abstract representation for each cluster of web pages. The latter, instead, is used to collect the dynamic portions of the web pages within the same cluster (performing *intra-cluster DOM differencing*), on top of which the tester might create test case assertions.

In the last step, the Code Generator **(5)** transforms each cluster into a Java page object, tailored for the Selenium WebDriver framework[10]. The Code Generator uses JavaParser to iteratively create from scratch the abstract syntax trees (AST) of the Java page objects. The class name derives from information associated with the cluster, together with a standard package name (po) and the necessary WebDriver imports. The class is populated with a WebElement instance for each web element in the cluster, properly annotated with a @FindBy annotation, specifying the locator (XPath or CSS). The class constructor contains a Selenium WebDriver variable to control the browser

[7] http://d3js.org/
[8] http://javaparser.github.io/javaparser/
[9] http://www.xmlunit.org/
[10] http://www.seleniumhq.org/projects/webdriver/

and resorts to the *PageFactory*[11] pattern to initialise the web elements at once. The methods that APOGEN generates are of three types: *navigations* between page objects, representing the links and the graph transitions (e.g., login page → home page), *actions* wrapping every data-submitting form and exposing the associated functionality (e.g., the login form), and *getters* – methods which retrieve textual portions of a web page that can be used to verify the behaviour of the web application through test case assertions (e.g., the total of a shopping cart).

The output of APOGEN is a set of Java page objects that reflect the pages of the web application, organised using the Page Factory design pattern, as supported by the Selenium WebDriver framework. The generated code can of course be modified according to the tester's wishes. A more detailed description and evaluation of the tool can be found in our recent papers [3, 4], while a web page containing demo videos is available at: http://sepl.dibris.unige.it/APOGEN.php.

Running APOGEN on PETCLINIC. Let us consider PETCLINIC[12], a veterinary clinic web application allowing veterinarians to manage data about pets and their owners. PETCLINIC makes use of technologies as Java Spring Framework, JavaBeans, MVC presentation layer and Hibernate. It consists of 94 files of various type (Java, XML, JSP, XSD, HTML, CSS, SQL, etc.), for a total of about 12 kLOC, of which 6.1 kLOC accounting for Java source files (63 Java classes). Hence, it is a medium size web system, with features and technologies that are quite typical of many similar systems available on the web.

We provided APOGEN with the URL of PETCLINIC (http://localhost:9966/petclinic/ on ours local machine), together with the data necessary for the login and form navigation. This task can be performed either via the tool's GUI, or by setting a configuration file. In the next step, the Crawler **(1)** reverse-engineered a graph-based representation of the web application, coming up with 26 nodes, i.e., 26 dynamic states of the web pages, and 105 event-based transitions between such nodes.

However, the manual inspection of such graph was challenging. Indeed, the high number of dynamic states (26) and transitions (105) made the visualisation of the graph quite tangled, definitely undermining its understandability and reducing the effectiveness of the automated page object creation. For this reason, the Clusterer **(2)** executed a clustering algorithm over the graph, with the aim of grouping within the same cluster web pages conceptually correlated among each other. Clusterer's default setting is [clustering algorithm=*"Hierarchical Agglomerative"*, feature vector=*"DOM tree-edit distance"*], because this was empirically found to be effective in producing clusters of web pages close to those manually defined by a human tester [4]. In the case of PETCLINIC, 10 clusters were found and displayed by CVE **(3)**. We manually inspected the clustering results. Since the Clusterer was able to find the best page-to-cluster assignment automatically, no manual adjustments were necessary. It is worth to mention that, without the use of clustering, APOGEN would have been generated 26 page objects for PETCLINIC (a 160% increment in the amount of generated page objects, and therefore of duplicated and useless code). In the next steps of the approach, the Static Analyser **(4)**,

[11] https://code.google.com/p/selenium/wiki/PageFactory

[12] https://github.com/spring-projects/spring-petclinic

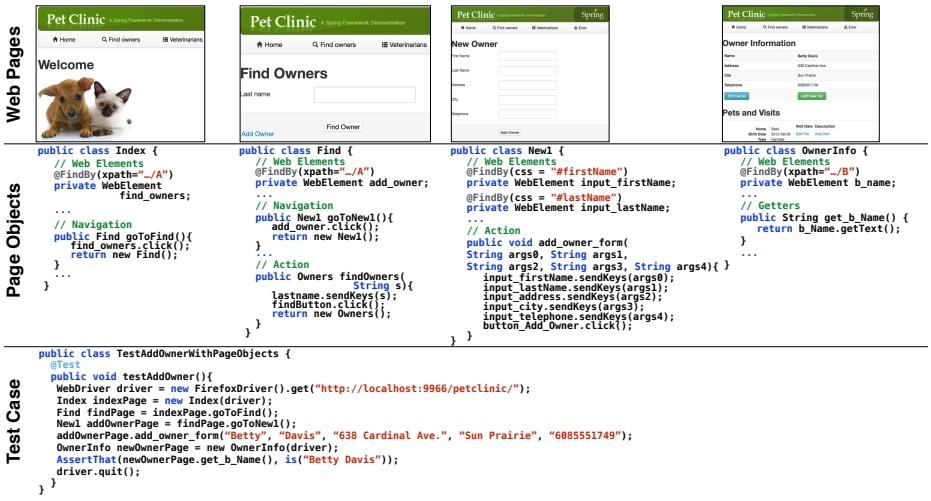

Fig. 2. Page objects generated by APOGEN to support a web test case development

and the Code Generator **(5)** ran to completion and automatically generated 10 Java page objects for PETCLINIC.

Page Objects to Support Test Case Development. Fig. 2 shows a test case for the "Add Owner" functionality of PETCLINIC, developed using the methods of the page objects generated by APOGEN. For space constraints, we limit the code only to the methods that are used by the test, in the considered test scenario. We can see how the page objects effectively realise the use case scenario steps as methods, and thus, are an effective aid for the tester during the creation of a real web test case for PETCLINIC.

3 Conclusions

We presented APOGEN, a prototype research tool for the automatic generation of page objects to be used for web applications testing. APOGEN leverages a combination of non-trivial techniques, such as reverse-engineering, machine learning, web-visualisation, HTML static analysis and differencing, and AST creation. APOGEN represents the most advanced state of the art tool for the automatic generation of page objects for web applications, because it is the first solution providing a high degree of automation.

References

1. M. Leotta, D. Clerissi, F. Ricca, and P. Tonella. Approaches and tools for automated end-to-end web testing. *Advances in Computers*, 101:193–237, 2016.
2. A. Mesbah, A. van Deursen, and S. Lenselink. Crawling Ajax-based web applications through dynamic analysis of user interface state changes. *TWEB*, 6(1):3:1–3:30, 2012.
3. A. Stocco, M. Leotta, F. Ricca, and P. Tonella. Why creating web page objects manually if it can be done automatically? In *Proc. of AST*, pages 70–74. IEEE, 2015.
4. A. Stocco, M. Leotta, F. Ricca, and P. Tonella. Clustering-aided web page objects generation. In *Proc. of 16th International Conference of Web Enginnering*, ICWE, 2016.
5. I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

A Appendix

APOGEN's home-page is at: http://sepl.dibris.unige.it/APOGEN.php
The web page includes the papers references, the tool source code, a demo of the Clusters Visual Editor (CVE), and two demo videos. A user manual with step-by-step instructions on how to install and use APOGEN is available at:

http://www.disi.unige.it/person/StoccoA/apogen/APOGEN-user-manual.pdf

The tool demonstration will be along the same spirit as the example in Section "Running APOGEN on PETCLINIC" of the demo paper. However, it will run interactively on a real-world application and it will include many more details on the benefits in real testing scenarios, the tool architecture, and the functionalities it offers. The demo will consists of three parts:

- An overview of APOGEN, where we discuss its general design principles and its overall usage workflow (as in Fig. 1);
- A demonstration consisting of a step-by-step illustration of how to use APOGEN to generate page objects for a sample real-world application. We will conduct the demo on a web application hosted on our local machine, to avoid any delays, or potential communication overhead. We will show how to setup the tool parameters via the GUI, illustrating each step of the page objects generation, and discussing each intermediate module (e.g., the use of the CVE, see Figure 3);
- A concluding part, where we will develop one or more simple functional test cases, using the freshly generated page objects of APOGEN for the sample web application. The aim is dual: (1) demonstrate the effective aid for the tester during the creation of real web test cases, and (2) illustrate the structure and readability of the generated code.

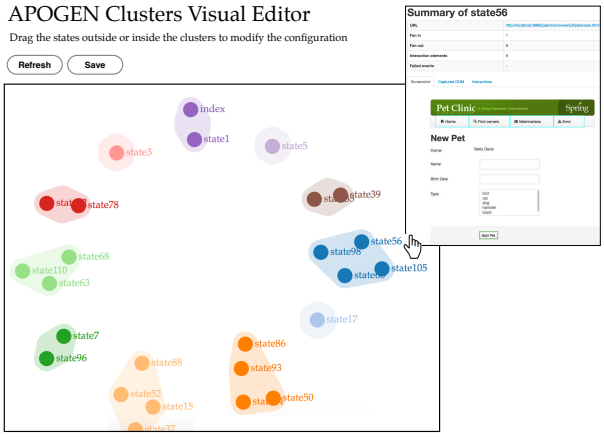

Fig. 3. Clusters Visual Editor (CVE) of APOGEN