Automatic Page Object Generation with APOGEN

Andrea Stocco, Maurizio Leotta, Filippo Ricca, Paolo Tonella

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.

Digital Object Identifier (DOI):

`http://dx.doi.org/10.1007/978-3-319-38791-8_42`

Automatic Page Object 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[1] is renowned 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 offer poor assistance in the creation of the source code [5]. In short, most of the page objects development effort is still on the shoulders of the tester.

Our tool APOGEN [5, 6] 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. Section 3 illustrates the tool functioning from the user's perspective, by means of a running example. Conclusions are drawn in Section 4.

2 Tool Architecture

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. APOGEN has been developed in Java and makes use of several external libraries and tools. Fig. 1 shows the high level architecture of APOGEN [6].

[1] http://www.seleniumhq.org/projects/webdriver/

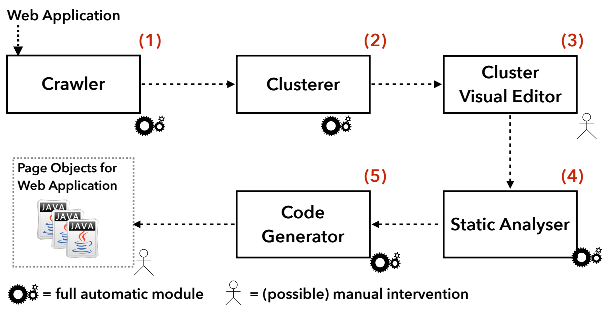


Fig. 1. High level architecture of APOGEN

The Crawler **(1)** is built on top of Crawljax [4], 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 [6], using clustering algorithms available from the popular machine learning library Weka.

The Cluster Visual Editor (CVE) **(3)** is a web-based tool developed using the D3 library[2]. 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[3] and XMLUnit[4]. 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. The Code Generator uses JavaParser to iteratively create from scratch the abstract syntax trees (AST) of the Java page objects. The class constructor contains a Selenium WebDriver variable to control the browser and resorts to the *PageFactory* 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. A more detailed description and evaluation of the tool can be found in our recent papers [5, 6], while a web page containing the source code and demo videos is available at: http://sepl.dibris.unige.it/APOGEN.php.

[2] http://d3js.org/ [3] http://javaparser.github.io/javaparser/ [4] http://www.xmlunit.org/

3 Running APOGEN on PetClinic

Let us consider PetClinic[5], 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.

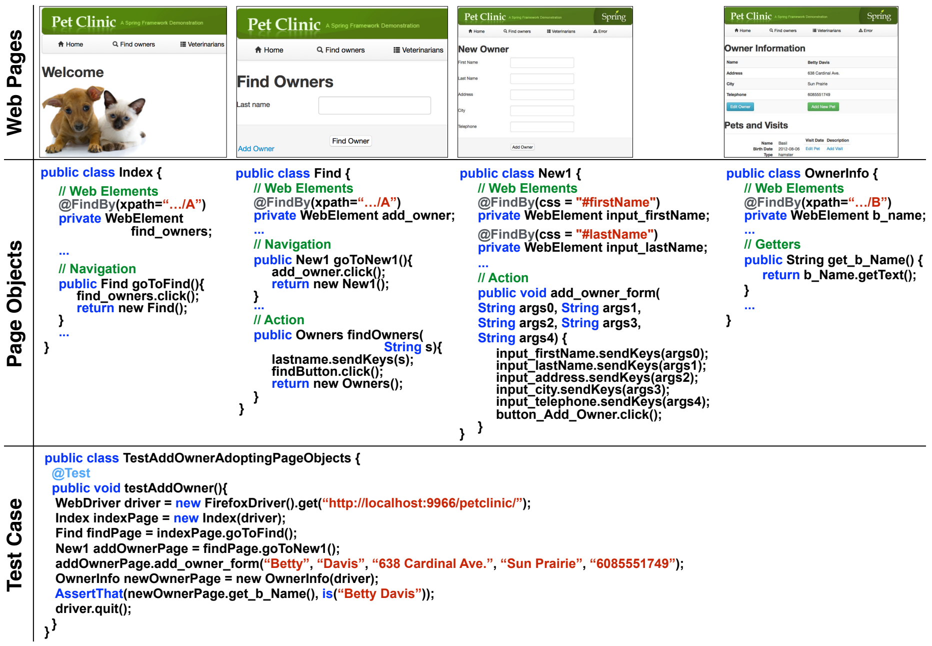
However, the manual inspection of such graph was challenging. Indeed, the high number of dynamic states (26) and transitions (105) made the visualisation of the graph quite tangled, definitely undermining its understandability and reducing the effectiveness of the automated page object creation. For 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 [6]. In the case of PetClinic, 10 clusters were found and displayed by CVE **(3)**. We manually inspected such clusters. The Clusterer was able to find the best page-to-cluster assignment automatically, thus no manual adjustments were necessary. It is worth to mention that, by disabling 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)**, and the Code Generator **(5)** ran to completion and automatically generated 10 Java page objects for PetClinic.

Fig. 2 shows a Selenium WebDriver 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.

4 Conclusions and Future Work

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

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



```
public class Index {
    // Web Elements
    @FindBy(xpath="…/A")
    private WebElement
                find_owners;
    ...
    // Navigation
    public Find goToFind(){
        find_owners.click();
        return new Find();
    }
    ...
}
```

```
public class Find {
    // Web Elements
    @FindBy(xpath="…/A")
    private WebElement add_owner;
    ...
    // Navigation
    public New1 goToNew1(){
        add_owner.click();
        return new New1();
    }
    ...
    // Action
    public Owners findOwners(
                        String s){
        lastname.sendKeys(s);
        findButton.click();
        return new Owners();
    }
}
```

```
public class New1 {
    // Web Elements
    @FindBy(css = "#firstName")
    private WebElement input_firstName;
    @FindBy(css = "#lastName")
    private WebElement input_lastName;
    ...
    // Action
    public void add_owner_form(
    String args0, String args1,
    String args2, String args3,
    String args4) {
        input_firstName.sendKeys(args0);
        input_lastName.sendKeys(args1);
        input_address.sendKeys(args2);
        input_city.sendKeys(args3);
        input_telephone.sendKeys(args4);
        button_Add_Owner.click();
    }
}
```

```
public class OwnerInfo {
    // Web Elements
    @FindBy(xpath="…/B")
    private WebElement b_name;
    ...
    // Getters
    public String get_b_Name() {
        return b_Name.getText();
    }
    ...
}
```

```
public class TestAddOwnerAdoptingPageObjects {
  @Test
  public void testAddOwner(){
   WebDriver driver = new FirefoxDriver().get("http://localhost:9966/petclinic/");
   Index indexPage = new Index(driver);
   Find findPage = indexPage.goToFind();
   New1 addOwnerPage = findPage.goToNew1();
   addOwnerPage.add_owner_form("Betty", "Davis", "638 Cardinal Ave.", "Sun Prairie", "6085551749");
   OwnerInfo newOwnerPage = new OwnerInfo(driver);
   AssertThat(newOwnerPage.get_b_Name(), is("Betty Davis"));
   driver.quit();
  }
}
```

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

web applications, because it is the first solution providing a high degree of automation. As future work, we plan to experiment with case studies involving human subjects to measure the efficacy in supporting the development of web test suites. The maintainability of the generated page objects can also benefit from robust web element localisation techniques [2, 3]. At last, we plan to enhance the level of automation, by employing MDE techniques as, for instance, templates.

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. M. Leotta, A. Stocco, F. Ricca, and P. Tonella. Using multi-locators to increase the robustness of web test cases. In *Proceedings of 8th International Conference on Software Testing, Verification and Validation*, ICST, pages 1–10. IEEE, 2015.
3. M. Leotta, A. Stocco, F. Ricca, and P. Tonella. ROBULA+: An algorithm for generating robust XPath locators for web testing. *Journal of Software: Evolution and Process*, 28(3):177–204, 2016.
4. 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.
5. A. Stocco, M. Leotta, F. Ricca, and P. Tonella. Why creating web page objects manually if it can be done automatically? In *Proceedings of 10th International Workshop on Automation of Software Test*, AST, pages 70–74. IEEE, 2015.
6. A. Stocco, M. Leotta, F. Ricca, and P. Tonella. Clustering-aided web page object generation. In *Proceedings of 16th International Conference of Web Engineering*, ICWE. Springer, 2016.