Course name & code: Bachelor's Thesis (CS498)
Georgios Gerasimos Leventopoulos (csd4152)
Undergraduate Student at University of Crete & Tampere University
Undergraduate Research Assistant - DiSCS - ICS - FORTH
Thesis Title: “Automated, large-scale website interaction through the generation of user action templates”
Supervisor Professor: Polyvios Pratikakis
Advisor Professors: Sotiris Ioannidis, Yannis Tzitzikas

1. Introduction

Automatically interacting with websites can be of significant importance for a variety of research studies, e.g., when testing for security-sensitive functionalities or when carrying out a privacy-related study. However, due to websites' heterogeneity, we cannot be aware of the different functionalities each one offers and how to properly exercise them. For instance, an e-commerce site may allow users to add items to their cart, purchase them or add reviews to products; automatically triggering each of these in a website agnostic manner is a non-trivial task. To aid in overcoming this problem to a certain extent and enable the automated execution of site-specific functionalities on a larger scale, we built a system that relies on common characteristics among websites that have been developed using the same content management system (CMS). The intuition behind this approach is that such CMS' will generally provide a set of well-defined steps to execute a certain functionality, e.g. visit page A, click button B, submit form C, etc. Accurately modeling these steps can allow us to exercise a given functionality in virtually any website that was built with that CMS.

2. Methodology

Before describing the methodological details of our approach, we need to define what exact functionalities, i.e. *tasks*, we want to model and eventually be able to exercise. After inspecting several websites from multiple CMS, we identified and decided to support the following functionalities:

- *Detect*: Detect if a website is made using the corresponding CMS.
- *Signup*: Locate the signup form and create a new user account.
- *Login*: Locate the login form and log into the website.
- *Locate item*: Locate an item/product on the website
- *Add item*: Add a product to the shopping cart.
- Remove item: Remove an item from the shopping cart.
- *Search*: Search for a term with the search bar.
- *Subscribe newsletter*: Fill the subscribe newsletter form with the user email.
- *Account settings*: Change user credentials and/or misc settings

To model the necessary steps to trigger each of these functionalities, we performed a manual process to create individual *user action templates* for each CMS. Specifically, for every functionality we needed to automate, we inspected the HTML code and URL structure in the 50 most popular websites for the corresponding CMS. We then tried to find common characteristics among them, such as common HTML attributes and URL paths. For example, regarding logging into Prestashop websites, we observed that most websites built with it annotate the corresponding login form with a *class=”login”* HTML attribute. Similarly, for the search task in Prestashop websites, we identified that the majority of them are using the *id=“search” HTML attribute,* while for the newsletter subscription they use the *id=”newsletter”* attribute. However, websites built with the same CMS are not restrained to using the same HTML attributes and values; instead, there can be different values across websites. For instance, when the user wants to add an item to their cart in Magento, we

observe that the corresponding item element in the page is either a *li* element, having a *class* attribute that contains the *'product-item'* substring or an *anchor* tag with a class that contains the "*product"* substring. Moreover, the user then needs to click a button with *class="add-to-cart"* to add the item to the cart.

Moreover, each task is accompanied by a *verification* procedure, which incorporates a series of steps to ensure the task at hand succeeded. For instance, in the login task, we ensure that the login happened by checking whether an error message was returned. Typically, this message is encapsulated in an element with a predefined HTML attribute, e.g. *id="create_account_error"*, or contains a specific text string. On the *add item* task, we checked that the system locates products on the website and also that the number of products that we found is more than the number of products that the user requests. Furthermore, for the subscribe newsletter task, we check that we do not have an error message (an error appears when we see the *class = 'newsletter_error'* HTML attribute). We followed similar approaches to ensure the successful execution of the remaining tasks.

We performed this process and created action templates for three CMS, namely Prestashop, Magento, and Drupal. Our action templates are realised as Python scripts that leverage the Selenium browser-automation module.

3. Evaluation

Regarding the experimental setup for the evaluation of our system, we used two sets of websites. The first set, referred to as the *benchmark set*, includes the 50 most popular websites for each CMS the system supports, which have a significant userbase, and aims to serve as a representative sample on how each CMS implements each of the offered functionalities. This set was used as the basis for the design and implementation of the action templates. The second set, referred to as the *evaluation set*, included 10 randomly selected websites from each CMS, to measure the effectiveness of the system on an unknown input. Then, we proceeded to execute the corresponding action template for each website and measured which tasks were executed successfully. We provide the detailed results for both sets in Table 1 and 2 respectively.

Table 1: Number of successful tasks per CMS for the *benchmark* websites

	Prestashop	**Magento**	**Drupal**
Detect	40/50	35/50	32/50
Login	45/46	0/50	0/50
Signup	0/46	0/50	0/50
Locate item	45/47	40/47	0/23
Add item	41/47	38/47	0/23
Remove item	0/47	0/47	0/23
Search	44/48	40/47	38/41

Subscribe newsletter	40/43	40/44	21/25
Account settings	0/46	0/50	0/50

Table 2: Number of successful tasks per CMS for the *evaluation* websites

	Prestashop	**Magento**	**Drupal**
Detect	7/10	8/10	5/10
Login	8/10	0/10	0/10
Signup	0/10	0/10	0/10
Locate item	8/10	6/9	0/6
Add item	5/10	6/9	0/6
Remove item	0/10	0/10	0/6
Search	7/10	8/10	6/7
Subscribe newsletter	6/7	6/8	5/7
Account settings	0/10	0/10	0/10

As can be seen, the generated templates are successful in a lot of the websites both in the *benchmark*, as well as in the unknown *evaluation* set of websites. We believe this verifies our initial assumption that the common characteristics among websites of the same CMS would allow us to automate site-specific functionalities. Regarding the failed tasks, we inspected each website separately to gain insights into why they failed. For the failed *login* tasks, we find that one of the root causes is that we cannot find the login page. This happens because sometimes the page is loading and we cannot click the element before it loads to the webpage. Furthermore, we tried to recognize the CMS that a website is made, by using the “detect” task. A lot of tasks are failing there because we did not find an absolute way to do this. We looked inside the source code for keywords such as “Magento” for websites that are built using Magento CMS and “Prestashop” for websites using Prestashop.

Moreover, on a large scale, we recognized that the majority of login and signup forms require a “captcha”, so we did not find a way to implement this using code. Additionally, our system could not accomplish some of the tasks that are using a Drupal CMS. While we consider find a better alternative to solve these problems, it is left out of scope for this thesis and leave it to future work. In some other cases, the system was unable to find the correct element based on the given attributes or at the given URL. This can be attributed to the page not being fully loaded when parsing it, e.g. due to slower response time or heavy use of Javascript and dynamically generated content. In a few other cases, the element that the system was trying to interact with, e.g. click it, was covered by another, overlapping element. Some websites do not have the support for some of the tasks. For example, some websites do not have a search bar or some websites do not have a subscribe newsletter form. In these cases, the methods return False (they identify that the search bar does not exist), so we count this as correct.

4. Future work

One of the limitations of this work, is that the action templates have to be generated manually, which requires a significant amount of time to inspect the different websites, locate meaningful similarities, identify different functionalities, etc. As part of our future work, we plan to develop techniques to reduce the required manual effort. Moreover, we plan to add support both for more content management systems, as well as more functionalities but also to improve the existing functionalities and correct mistakes that we might have. Finally, to increase the effectiveness of the system, we plan to explore existing solutions for the tasks that failed in our system.

5. Conclusion

To sum up, in this work we implemented a system, to do automated tasks on websites that have been developed using the same content management system (CMS). This system focuses on finding the same properties among these websites and using them to accomplish automation functions on a scale. Our experimental evaluation showed while our system is successful on a lot of websites, there might always be some edge cases that we cannot predict, that cause our system to fail. Also, as part of our future work, we plan to work on the limitations we might have. Specifically, we want to add extra capabilities to our system and also to work on existing limited features and make them better and more efficient. Another concept that we want to cover, is to decrease the amount of time in manually searching for similarities on the websites.