Providing Programmers Feedback and UI Context When Developing Web Automation Scripts

Rebecca Krosnick[1], Steve Oney[2,1]
[1]*Computer Science & Engineering*, [2]*School of Information*
University of Michigan | Ann Arbor, MI USA
{rkros, soney}@umich.edu

***Abstract*—To more efficiently and effectively perform frequent, tedious, or inaccessible web tasks, web users can leverage web automation macros to programmatically click, type, and perform other page operations. Developers use automation libraries such as Selenium to write custom web automation scripts, but this comes with challenges. A developer must identify robust navigation and element selection logic that works across a variety of user inputs and website pages. Choosing meaningful and robust CSS selectors to query the appropriate UI elements from the DOM can be particularly challenging. We believe developers need better tools for understanding the effects their script has on a website. We built a prototype web automation IDE that embeds UI snapshots and provides live feedback on CSS selectors, and presents these across different scenarios so the programmer can assess script robustness. We plan to demo our prototype to get feedback and to discuss the challenges and opportunities of UI automation.**

I. Introduction

Certain web tasks can be obtrusively frequent (e.g., scraping data), tedious (e.g., navigation and form filling), or inaccessible (e.g., to a blind person) for someone to complete by hand. Web users can offload such tasks by running web automation macros (e.g., via a browser extension or keyboard shortcut) that programmatically click, type, and otherwise interact with website UIs. To create custom web macros, developers most commonly write scripts, for example using Selenium [1], Puppeteer [2], or Cypress [3]. Writing a web automation script involves querying the Document Object Model (DOM) [4], e.g., using CSS [5] or XPath [6], to select desired UI elements and then interacting with them as appropriate. However, it can be challenging to write robust scripts that work across different user inputs (e.g., dates to book a flight), page states (e.g., logged in or not), page content (e.g., different data across semantically similar Wikipedia articles), and page DOMs (e.g., randomly generated UI element IDs). First, the developer must be aware of such differences and identify appropriate navigation and element selection logic. Then, the developer must choose robust and meaningful CSS or XPath selectors that select the elements they intend. These challenges are exacerbated by the fact that developers typically do not own the websites they are trying to automate, and therefore must learn the DOM structure on the fly and also cannot anticipate future changes to the website DOM or content.

We believe developers creating web automation scripts need more resources than current environments offer. We propose providing UI context and element selection feedback within the web automation developer's scripting environment. We built a prototype that provides snapshots of the website's UI state per line of code and feedback on the validity of CSS selectors. This UI context and feedback is provided in one place across user inputs and loop iterations to help developers assess script success and discrepancies across scenarios.

II. Prototype

Our prototype web automation IDE embeds UI context and element selection feedback within the web automation developer's scripting environment. The programmer writes their script in the editor in the left pane (Figure 1A), and can inspect (using Chromium dev tools) and interact with the live website in the right pane (Figure 1B).

UI snapshots. When the programmer runs their script, snapshots of the website's UI state before and after each line of code are captured and rendered in the right pop-out pane (Figure 2A). The programmer can view the snapshots for a given line by clicking on that line in the editor or using the form widget on the right. Developers can use these UI snapshots to explore the effect a particular line of code has on the website UI and whether this matches the developer's intentions.

CSS selector feedback. As the programmer writes a CSS selector in the editor, elements matching the selector are highlighted in blue in the live website view in real-time (Figure 1C). This offers the programmer live feedback on which elements their selector is matching, and allows them to iteratively refine the selector inline and evaluate whether it matches the desired element(s). The editor also provides inline feedback on CSS selectors, indicating with a squiggle and tooltip message whether a selector is valid and unique (Figures 1D, 2B). This CSS validity check is performed each time the programmer runs their code. The CSS validity check may also be provided live as the user edits their code, if the UI "before" snapshot for the given line of code is still valid (i.e., is not stale due to code changes earlier in the script).

Across scenarios. The prototype IDE enables programmers to see the above UI snapshots and CSS selector feedback across different runs of a given line of code. This means that when the script contains a loop, for example for scraping data from multiple pages on a website, the programmer will be able to see UI snapshots and CSS validity checks in the context of each loop iteration (e.g., article page) the code was run

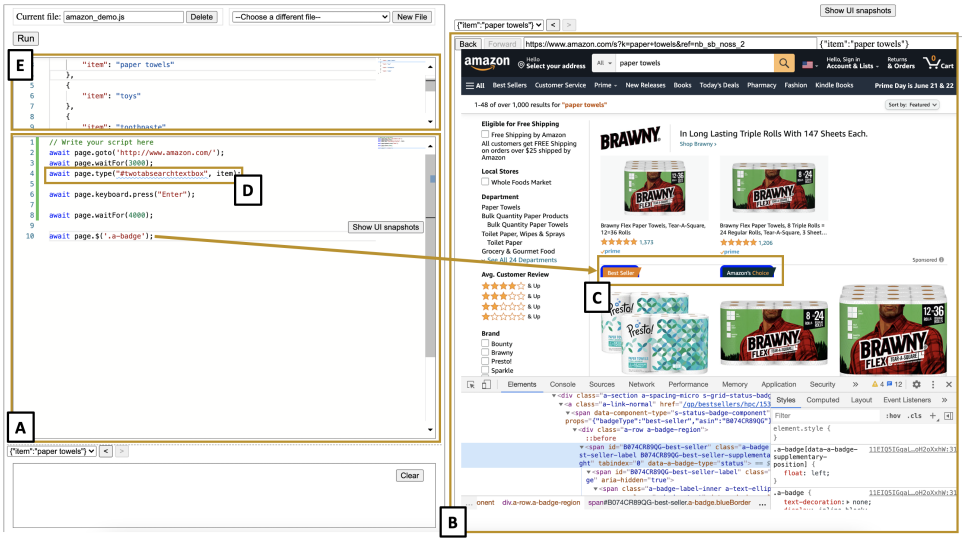

Fig. 1. Our prototype web automation IDE. As programmers write their script (A), they can inspect and interact with the target website (B) within the IDE to identify desired UI elements. When the programmer types a CSS selector on line 10, the matching UI elements on the website are highlighted in blue (C). Feedback on CSS selectors also appears inline – the blue squiggle under #twotabsearchtextbox indicates it is found and unique (D). The programmer can also run and test their script on different user inputs (E).

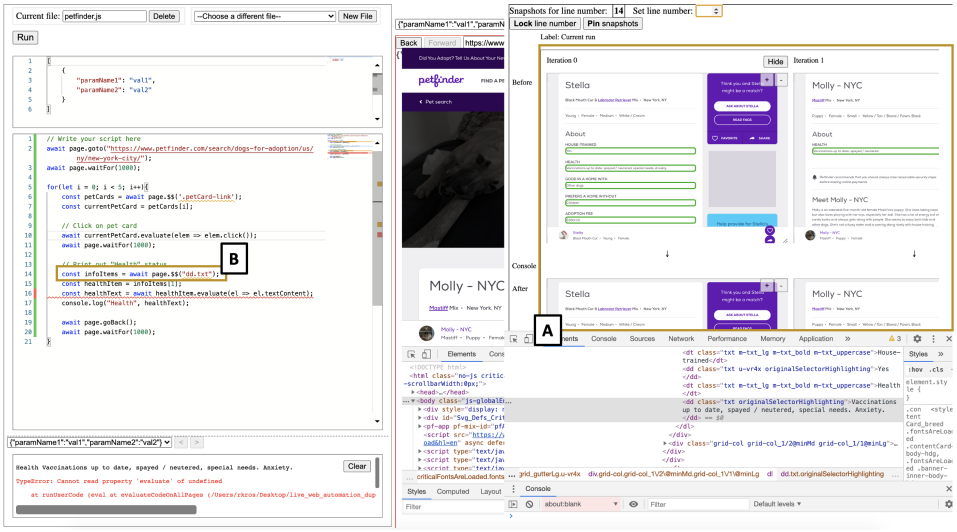

Fig. 2. The prototype lets users view UI snapshots for each line of code. Here, the script has failed in the i=1 iteration of the loop, and the user can inspect the UI snapshots (A) to try to understand why. The UI snapshots for line 14 indicate that Stella has 5 info elements matching selector “dd.txt” (B) whereas Molly only has 1, which explains why the infoItems[1] indexing on line 15 failed for Molly’s page.

on. The prototype IDE also lets users enter multiple sets of user inputs to run their script on (Figure 1E), enabling them to see UI snapshots and CSS validity checks across different inputs. UI snapshots across different runs are shown side by side in the right pop-out pane, enabling users to compare snapshot appearances and DOMs, and identify discrepancies (Figure 2A).

The prototype is implemented as an Electron app [7], and uses the Puppeteer automation library [2] for performing page operations and rrweb-snapshot [8] for capturing DOM snapshots at runtime. The prototype lets users run scripts on any real website.

III. Presentation

We will present a live demo of our prototype during the showpieces track, having a few in-progress and completed scripts ready to run or further edit. We have three goals in presenting our demo as part of the showpieces track: 1) to get feedback on our prototype, 2) to further discuss a paper we submitted to the VL/HCC 2021 main program which studies the challenges programmers face in writing web automation scripts (this prototype is one of the environments participants tested), and 3) to discuss in general the challenges and opportunities in UI automation and in integrating UI context into development environments.

REFERENCES

[1] "Selenium," https://www.selenium.dev/, accessed: 2020-09-11.
[2] "Puppeteer," https://pptr.dev/, accessed: 2020-09-18.
[3] "Cypress," https://www.cypress.io/, accessed: 2021-03-19.
[4] "Introduction to the dom," https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction, accessed: 2021-06-11.
[5] "Css selectors," https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors, accessed: 2021-03-19.
[6] "Xpath," https://developer.mozilla.org/en-US/docs/Web/XPath/, accessed: 2021-03-20.
[7] "Electron," https://www.electronjs.org/, accessed: 2020-09-18.
[8] "rrweb-snapshot," https://github.com/rrweb-io/rrweb-snapshot, accessed: 2020-09-18.