Tools for Creating UI Automation Macros

Rebecca Krosnick
Computer Science and Engineering
University of Michigan
Ann Arbor, MI USA
rkros@umich.edu

I. Introduction

Automation macros enable users to perform digital tasks programmatically to save time or support hands-free interaction. For example, macros can be used to perform web scraping for a research project (e.g., scraping articles from a news site) or personal task automation via natural language (e.g., ordering food for delivery). Some kinds of automation are built into our devices and are readily available (e.g., via Siri [1] or Alexa [2]), but this set is limited and often will not support a user's niche or complex needs. Users can create their own custom macros, but traditionally this requires writing program code which involves a significant amount of effort for programmers and is infeasible for non-programmers.

In my PhD work, I have studied and designed tools for developers and end-users to more intuitively create user interface (UI) automation macros. First, I studied the challenges and needs of programmers writing web automation scripts [3], both in a traditional text editor and in richer environments, including a prototype IDE I built that provides context about the target UI and feedback on element selection logic. Next, I designed a programming by demonstration (PBD) environment that enables end-users to create custom macros for answering questions on the web without needing to write code [4].

So far my work has focused on helping users create macros that perform a desired web scraping or automation task on a single website. However, it makes sense that users might also want to perform the same kinds of automation on semantically similar websites. For example, a user might want to create an automation macro that can order food not only from the DoorDash website, but also from the GrubHub and Uber Eats websites. Another user might want to create an automation macro that scrapes data from staff directories across different department and university websites. In both cases, the websites' content will be similar and the macro should perform the same high-level actions, but the websites' exact visual appearance, widgets, and underlying implementation will vary. Currently users would need to create separate macros from scratch for each website.

In my future work, I plan to help users create *semantic macros* -– macros that are capable of performing a given high-level semantic task across different websites – without needing to create new automation logic from scratch for each new website.

II. Developer environments for creating macros

Traditionally, users need programming experience to create an automation macro. This involves writing program code that performs a sequence of interaction events on a UI, mimicking user interactions – for example, clicking on buttons and typing into text fields. This requires programmers to understand the UI they are trying to interact with and the effect their automation code has on the UI. Historically most libraries for writing automation macros have been entirely text-based, e.g., Selenium [5] and Puppeteer [6], where the programmer must reason about two related but disconnected environments – the code in their editor/IDE, and the UI in the web browser. Newer frameworks like Cypress [7] now allow users to see their automation logic and target UI in an integrated environment.

In my prior work, I studied developers as they used these environments to write automation macros [3]. First, I studied programmers using traditional text editor environments, writing in Puppeteer. Participants faced a number of challenges, ranging from identifying unique and robust element selection logic, to appropriately interacting with complex widgets like calendars, to understanding the source of automation errors.

Next, I studied programmers using richer UI automation environments that provide UI context and feedback, namely Cypress and a prototype IDE I built. We asked participants to write scripts to scrape data about articles on a blog website and pets on a pet adoption website. This involved appropriately navigating across multiple pages on a given website. Participants appreciated that these environments offered feedback on each piece of element selection logic, allowing them to visually see which elements were selected and confirm if this matched their intent. However, even with the help of UI context and feedback, participants still experienced challenges in constructing element selection logic that appropriately generalizes. For example, it was challenging to construct a generalized CSS selector [8] for UI elements that had different class names across different pages.

III. End-user environments for creating macros

To make creating UI automation macros feasible for non-programmers, many researchers have explored using programming-by-demonstration (PBD) approaches. With PBD [9], [10], an end-user provides a demonstration of how their desired program should behave in a given scenario, and

978-1-6654-4214-5/22/$31.00 ©2022 IEEE

the PBD inference engine then tries to infer a generalized program. A key challenge of PBD is correctly inferring the user's intent. Sugilite [11]–[13], VASTA [14], and AutoVCI [15] have used PBD to support end-users in building intelligent agents (IAs) and macros that perform digital tasks in response to natural language requests.

In my recent work [4] I have designed a PBD system that enables end-users to create custom automation macros that answer formulaic questions about content on a website. Our key insight is that if a user provides a single demonstration of how to answer a specific question, then if that question and the content on the website follow a structural pattern we can infer a program for answering variations of that question. An end-user starts with a concrete question they have about a particular website, e.g., "How many home runs did Vladimir Guerrero Jr. have?" on the Major League Baseball (MLB) statistics website[1]. Next, the PBD system asks the user to identify how the question can generalize through *parameters* and *alternative values*. For example, the user might generalize the above question to "How many <*statistic*> did <*player*> have?". Finally, the user chooses a specific set of parameter values (e.g., <*statistic* = *"hits"*> and <*player* = *"Rafael Devers"*>) and demonstrates the correct answer for that question and the necessary interaction events on the website UI to find that answer. The PBD system then infers a generalized program that answers questions of the form "How many <*statistic*> did <*player*> have?" by leveraging the user-provided parameters and values to understand relevant structural patterns in the website UI's Document Object Model (DOM) [16].

IV. Future Work

In my future work, I plan to support users in creating *semantic macros* -– macros that perform a given high-level semantic task (e.g., booking a flight) and are capable of doing this across different websites (e.g., Delta, Southwest, JetBlue). Currently to create automation that works across multiple websites, a user would need to create a macro from scratch per website (e.g., by writing program code, or using a PBD system capable of the desired inference). However, this seems unnecessarily tedious, as these websites are semantically similar and the automation macros should be performing the same high-level actions. For example, even though each airline website has its own unique visual appearance, booking a flight involves the same set of actions across websites: choosing a departure location, arrival location, and flight date.

To reduce duplicate effort in creating common automation logic, I aim to help users leverage an existing macro they have (e.g., for booking a flight on Delta) and adapt it to a new website (e.g., Southwest), so they do not need to start from scratch. The high-level approach would be to take the sequence of programmatic interaction events from the initial macro and find the corresponding sequence of programmatic interaction events on the new website. I hope to explore how much of this adaptation we can do automatically, versus what input and assistance we would need from the user.

I expect that the primary challenge in automatically adapting a macro from one website to another will be in identifying corresponding UI elements between the websites, e.g., that the "Depart" field on the Southwest website corresponds to the "Origin" field on the Delta website. There are multiple reasons this is hard. First, different natural language labels may be used to describe the same kind of UI element (e.g., "Depart" vs "Origin"). We cannot rely on finding the exact same natural language strings, and will probably want to use WordNet [17] or knowledge graphs to look for related words and phrases. Additionally, corresponding UI elements will not necessarily have the same widget type. For example, selecting "Round Trip" versus "One Way" is done through a dropdown menu on the Delta website, but through radio buttons on the Southwest website.

Even if we do find a candidate UI element on the new website with the exact same text and widget type as the original website, this is not necessarily the right match – e.g., at purchase time, there could be multiple text fields with the label "Email" for inputing email addresses (e.g., one for the buyer, one for sharing your itinerary with family). Rather than naively identifying corresponding UI elements by just text and widget type, we may also want to consider their spatial proximity to other relevant UI elements, e.g., noting that the email text field closer to the user's billing address is likely the most relevant one. We may consider building knowledge graphs like in Appinite [12] to represent the semantic and spatial relationships between UI elements, here using them to find similar semantic and spatial relationships across websites.

We may also want to consider machine learning and computer vision approaches for identifying corresponding UI elements. Prior work [18]–[23] has collected large datasets of mobile user interfaces, interaction traces, and crowd worker annotations to train machine learning models that can predict UI properties (e.g., alt text, hierarchical structure) and UI similarity from screenshots.

So far we have assumed all the necessary UI elements will appear on the same page and immediately in view. However, websites are typically dynamic multi-page apps. A desired UI element might be hidden in a collapsed pane or may even appear on the next page of the website. Our automated adaptation algorithm may need to insert or remove interaction events in order to bring the website into the correct UI state to find desired UI elements. We may want to adopt approaches from prior work on visual web test repair [24] that programmatically interact with a website to explore different UI states and search for desired UI elements.

Finally, our automated approach for adapting macros may not always work. It will be important to understand the limitations of our approach and to explore low-effort meaningful ways to ask the end-user for help. For example, we may want to ask the user to confirm a candidate corresponding UI element we have low confidence in, or to perform a micro-demonstration of how to interact with a unique UI element.

[1] https://web.archive.org/web/20220201043626/https://www.mlb.com/stats/

REFERENCES

[1] "Siri," https://www.apple.com/siri/, accessed: 2022-04-06.

[2] "Amazon Alexa Voice AI," https://developer.amazon.com/en-US/alexa, accessed: 2022-04-06.

[3] R. Krosnick and S. Oney, "Understanding the Challenges and Needs of Programmers Writing Web Automation Scripts," in *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2021, pp. 1–9.

[4] R. Krosnick and S. Oney, "ParamMacros: Creating UI Automation Leveraging End-User Natural Language Parameterization," in *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2022.

[5] "Selenium," https://www.selenium.dev/, accessed: 2020-09-11.

[6] "Puppeteer," https://pptr.dev/, accessed: 2020-09-18.

[7] "Cypress," https://www.cypress.io/, accessed: 2021-03-19.

[8] "CSS selectors," https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors, accessed: 2021-03-19.

[9] A. Cypher and D. C. Halbert, *Watch what I do: programming by demonstration*. MIT press, 1993.

[10] H. Lieberman, *Your wish is my command: Programming by example*. Morgan Kaufmann, 2001.

[11] T. J.-J. Li, A. Azaria, and B. A. Myers, "SUGILITE: Creating Multimodal Smartphone Automation by Demonstration," in *Proceedings of the 2017 CHI conference on human factors in computing systems*, 2017, pp. 6038–6049.

[12] T. J.-J. Li, I. Labutov, X. N. Li, X. Zhang, W. Shi, W. Ding, T. M. Mitchell, and B. A. Myers, "APPINITE: A Multi-Modal Interface for Specifying Data Descriptions in Programming by Demonstration Using Natural Language Instructions," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2018, pp. 105–114.

[13] T. J.-J. Li, M. Radensky, J. Jia, K. Singarajah, T. M. Mitchell, and B. A. Myers, "PUMICE: A Multi-Modal Agent that Learns Concepts and Conditionals from Natural Language and Demonstrations," in *Proceedings of the 32nd annual ACM symposium on user interface software and technology*, 2019, pp. 577–589.

[14] A. R. Sereshkeh, G. Leung, K. Perumal, C. Phillips, M. Zhang, A. Fazly, and I. Mohomed, "VASTA: a vision and language-assisted smartphone task automation system," in *Proceedings of the 25th international conference on intelligent user interfaces*, 2020, pp. 22–32.

[15] L. Pan, C. Yu, J. Li, T. Huang, X. Bi, and Y. Shi, "Automatically Generating and Improving Voice Command Interface from Operation Sequences on Smartphones," in *CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–21.

[16] "Document Object Model (DOM)," https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/, accessed: 2021-06-29.

[17] G. A. Miller, "WordNet: a lexical database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[18] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, 2017, pp. 845–854.

[19] T. J.-J. Li, L. Popowski, T. Mitchell, and B. A. Myers, "Screen2Vec: Semantic Embedding of GUI Screens and GUI Components," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–15.

[20] X. Zhang, L. de Greef, A. Swearngin, S. White, K. Murray, L. Yu, Q. Shan, J. Nichols, J. Wu, C. Fleizach *et al.*, "Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–15.

[21] J. Wu, X. Zhang, J. Nichols, and J. P. Bigham, "Screen Parsing: Towards Reverse Engineering of UI Models from Screenshots," in *The 34th Annual ACM Symposium on User Interface Software and Technology*, 2021, pp. 470–483.

[22] S. Feiz, J. Wu, X. Zhang, A. Swearngin, T. Barik, and J. Nichols, "Understanding Screen Relationships from Screenshots of Smartphone Applications," in *27th International Conference on Intelligent User Interfaces*, 2022, pp. 447–458.

[23] J. Chen, A. Swearngin, J. Wu, T. Barik, J. Nichols, and X. Zhang, "Towards Complete Icon Labeling in Mobile Applications," in *CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–14.

[24] A. Stocco, R. Yandrapally, and A. Mesbah, "Visual web test repair," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 503–514.