ScrapeViz: Hierarchical Representations for Web Scraping Macros

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

Steve Oney
School of Information
University of Michigan
Ann Arbor, MI, USA
soney@umich.edu

***Abstract*—Programming-by-demonstration (PBD) makes it possible to create web scraping macros without writing code. However, it can still be challenging for users to understand the exact scraping behavior that is inferred and to verify that the scraped data is correct, especially when scraping occurs across multiple pages. We present ScrapeViz, a new PBD tool for authoring and visualizing hierarchical web scraping macros. ScrapeViz's key novelty is in providing a visual representation of web scraping macros—the sequences of pages visited, generalized scraping behavior across similar pages, and data provenance. We conducted a lab study with 12 participants comparing ScrapeViz to the existing web scraping tool Rousillon and saw that participants found ScrapeViz helpful for understanding high-level scraping behavior, tracing the source of scraped data, identifying anomalies, and validating macros while authoring.**

***Index Terms*—web scraping, visual representation, automation, programming-by-demonstration**

I. Introduction

User interface (UI) automation macros can save users time and effort by automatically performing digital tasks. Some automation is readily available through virtual assistants or pre-programmed macros (e.g., in iOS Shortcuts [1]). However, for long-tail needs, users need to create custom macros, traditionally by writing code that mimics a user's mouse and keyboard actions. Writing this macro code can be prohibitively challenging, even for experienced programmers [2].

Prior work has leveraged programming-by-demonstration (PBD) [3], [4] to allow users (including non-programmers) to create macros without writing code. With PBD, users provide a few concrete demonstrations of the desired program behavior, and then the system infers a generalized program. Although PBD has made it easier for people to create automation and scraping macros [5]–[16], it comes with several challenges [17], many the consequence of users not understanding how PBD-generated programs work. More modern web automation tools driven by Large Language Models (LLMs) can similarly be inscrutable and difficult to understand [18].

We present *ScrapeViz*, a new PBD tool for creating web scraping macros, with a focus on helping users understand macros' behavior. ScrapeViz is designed for scraping *distributed hierarchical* data [5], which involves parent-child relationships (hierarchical) across multiple pages (distributed). Users navigate the web as they normally would, using familiar actions such as clicks to locate and select the data they want to extract, and ScrapeViz generalizes such actions across similar UI elements and across similar website pages.

The novelty of ScrapeViz is in the tools it offers for understanding web scraping behavior across multiple website pages. Prior systems also allow users to create nested-loop [5]–[8] or parameter-based [9], [12], [13], [16] automation macros. However, there are key limitations to how these systems represent macro behavior to users—either they provide no representation (requiring users to run the macro to understand its behavior), a limited preview of behavior on the next input, or a natural language description of macro steps that is separated from the execution context. ScrapeViz aims to address these limitations through a novel storyboard-like visualization, as Figure 1 shows. This visualization provides a high-level overview of the macro that represents key information for understanding its behavior: the pages visited, how actions generalize across semantically similar pages, relationships between pages, and the data scraped from each page (highlighted in context). As we found in a within-subjects lab study comparing ScrapeViz and Rousillon [5], ScrapeViz's visualization can make web scraping macros easier to understand and debug.

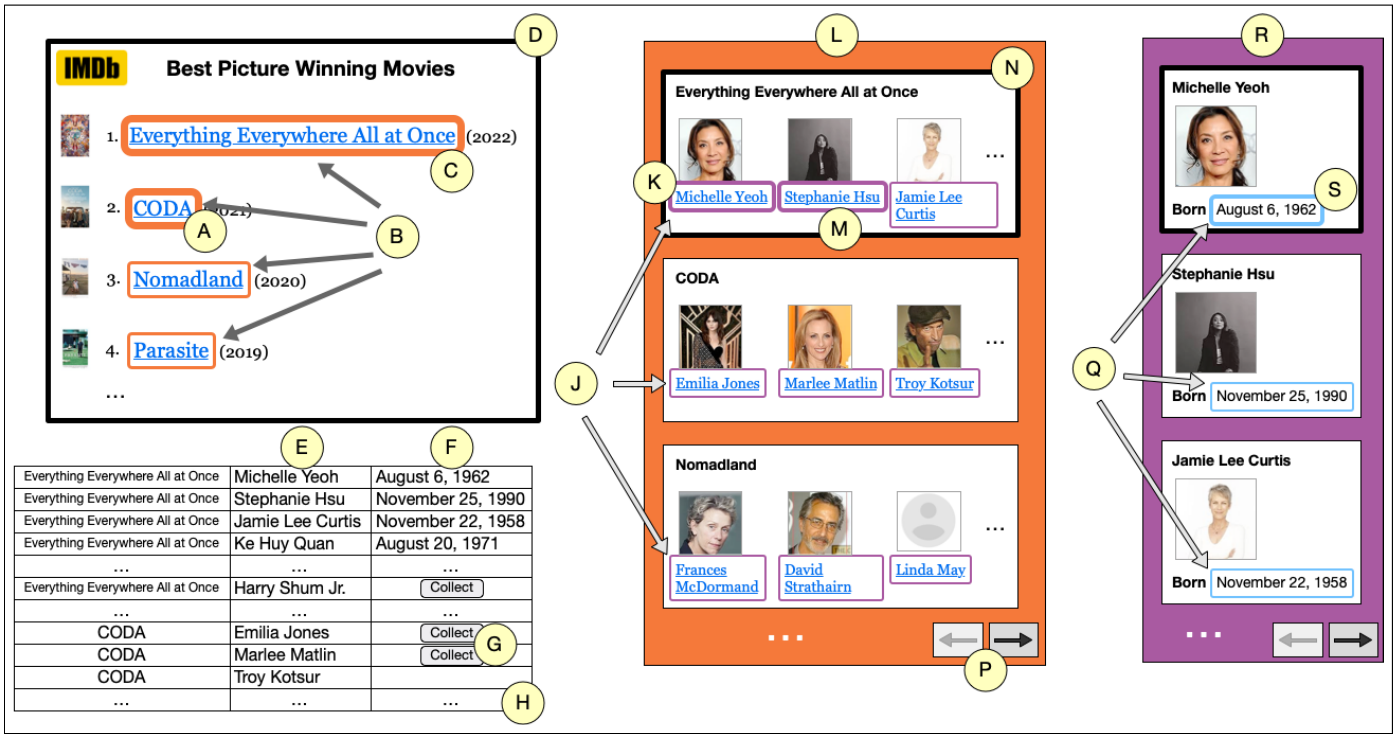
II. Related Work

ScrapeViz builds on prior work in web scraping tools, PBD, and visualizations for understanding automation.

A. Web scraping

Web scraping is the process of extracting data from websites. Traditional web scraping tools such as Beautiful Soup [19] and Selenium [20] require users to write code. Several tools have sought to make web scraping more accessible to a wider audience through no-code or low-code approaches. Marmite [21] offers users a visual data flow interface for scraping and a spreadsheet interface for viewing scraped results, but does not support tasks that involve hierarchical data or multiple pages. Sifter [22] can extract data across multiple pages but is limited to pagination-based navigation and does

This work is supported by NSF Award 2007857.
*The author is currently affiliated with Postman, Inc. This research was done when she was a doctoral student at the University of Michigan.



Everything Everywhere All at Once	Michelle Yeoh	August 6, 1962
Everything Everywhere All at Once	Stephanie Hsu	November 25, 1990
Everything Everywhere All at Once	Jamie Lee Curtis	November 22, 1958
Everything Everywhere All at Once	Ke Huy Quan	August 20, 1971
...	...	...
Everything Everywhere All at Once	Harry Shum Jr.	Collect
...	...	...
CODA	Emilia Jones	Collect
CODA	Marlee Matlin	Collect
CODA	Troy Kotsur	
...	...	...

Fig. 1. An illustration of ScrapeViz's final state after the user has completed the IMDb scraping task in section IV. ScrapeViz gives users a programming by demonstration (PBD) authoring interface and interactive storyboard-like visual representations of their web scraping macro. A user begins with a single website viewport (D) and provides demonstrations (C and A for movie titles, K and M for actor names, S for birthdate) of the scraping and navigation actions they want to perform – ScrapeViz adds colored borders to indicate selected elements, opens new viewports for website pages navigated to, and generalizes these actions across UI elements and pages (using the PBD algorithm described below). During this process, ScrapeViz builds up a hierarchy of pages visited either manually or by PBD generalization – a top-level IMDb page containing a list of movies (D); in column L, a movie page for each of the movies in the top-level list; and in column R, an actor page for each actor in the first movie, "Everything Everywhere All at Once". ScrapeViz uses color-coding to indicate scraping generalizations (e.g., orange for movie titles (B), purple for actor names (J)) and to indicate which UI elements when clicked lead to which website pages (e.g., orange-bordered movie titles (B) when clicked open the movie pages (L)). Users can navigate and inspect pages more closely by clicking directly on a small viewport, browsing sibling pages (P), or by clicking on a cell in the output table (H) to be taken to its source page and location.

not support hierarchical data. In contrast, ScrapeViz supports web scraping for hierarchical data and across multiple pages.

Large Language Models (LLMs) can also assist with web scraping tasks by either writing scraping code or identifying relevant elements in web content. However, LLMs can be opaque in their scraping process, produce results that are plausible but incorrect, and be impractically difficult to debug and fix [18]. We focus on programming-by-demonstration approaches, which can be easier to predict and understand.

B. Programming by demonstration

Programming by demonstration (PBD) enables users to create computer programs without writing code. Users instead demonstrate how the program should behave in a small set of scenarios, and the system then infers a generalized program. Prior work has explored PBD for creating programs for personal task automation [9]–[16] and web scraping [5]–[8].

ScrapeViz leverages PBD approaches similar to prior PBD web scraping systems [5]–[8] which also enable distributed hierarchical scraping. In Rousillon [5], users provide one example for each kind of data they want to scrape for the system to generalize from; in WebRobot [6] and ScrapeViz, users provide two examples. However, unlike these systems, ScrapeViz also provides a visual representation that gives a complete overview of macro behavior in context.

C. Understanding macros

To understand how a macro behaves, a user can manually run their macro on different inputs, but this can be cumbersome. Some PBD systems [6]–[8], [16] give macro creators a preview of how the macro will run on the next input, which helps them understand behavior as they build it. Rousillon [5] uses color coding to highlight the corresponding UI elements on a given page, but only on a single page and not across multiple pages, and only during authoring and not afterward. A key limitation of these approaches is that the user only sees the macro run on a small set of inputs—they do not get an overview of how the macro works broadly and as a result may miss cases where the macro does not behave as desired.

MIWA [8] provides an overview of macro behavior through a step-by-step natural language description of actions, visually highlighting corresponding UI elements on the website page for each action, and proactively pointing out potential anomalies. However, users cannot see an overview of pages visited, cannot see visual correspondence highlighting across multiple pages, and cannot easily determine the page and location that data in the output table came from. Instead, ScrapeViz aims to help users get an overview of macro behavior across different page contexts through a visual representation of pages visited and interactive output table for checking data sources.

III. Design

ScrapeViz has the following key design features:

- *Authoring*
  - *Programming-by-demonstration.* Authors provide two examples for each navigation or scraping action they want to perform. The system then infers the rest of the UI elements to perform that action on, within and across website pages.
  - *Live feedback.* As users author, they are given immediate feedback of actions performed, pages visited, specific data scraped, and generalizations.
- *Visual representation.*
  - *A storyboard-like visual* of action sequences and the hierarchy of resulting website pages, presented through multiple live viewports. Parallel website pages are grouped together to signify their semantic similarity and scraping generalizations across them.
  - *Color-coding* of UI elements identified as parallel based on user demonstrations.
  - *Interactivity* that enables users to zoom in on any given website page to inspect or add actions.
- *Interactive output table* where users click on a cell and are automatically taken to its source page and location.

IV. Sample Usage Scenario

A. Authoring

Susan is analyzing the distribution of actor ages in popular movies and needs to collect data from the IMDb website[1]. For each movie, she needs to collect its name, its actors' names, and the birthdate for each actor[2]. Susan decides to use *ScrapeViz* to create a scraping macro to collect this data.

ScrapeViz starts by showing just a top-level IMDb page (Fig 1-D) which includes a list of movie titles, each a link leading to the individual movie's page. Susan starts by **demonstrating collecting data for one movie and one of its actors**. She does this by clicking the first movie title on the page, "Everything Everywhere All at Once" (EEAAO) (Fig 1-C), which scrapes the movie's title and navigates to the movie's page. Instead of opening the movie page in the current viewport, ScrapeViz leaves the current viewport intact at the top-level page (Fig 1-D) and creates a new smaller viewport (Fig 1-N) next to it to load the movie page in. This allows Susan to keep track of the pages she has visited and revisit them later to make edits or additions. Susan clicks on the smaller viewport to expand it and see the movie page more clearly; this then shrinks the original viewport containing the list of movies. She then clicks the first actor name link on the EEAAO movie page, Michelle Yeoh (Fig 1-K), which scrapes her name and navigates to her page. Finally, she scrapes the birthdate "August 6, 1962" (Fig 1-S) on Michelle Yeoh's page.

At each step of Susan's demonstration, the interface builds up a storyboard-like visualization illustrating the sequence of pages navigated to, places a border around selected UI elements illustrating data scraping and clicking actions performed, and places scraped text into the output table (Fig 1-H). The visualization uses color-coding to indicate element interactions that lead to new UIs. Here, for example, the same color orange border is used to convey that clicking the "Everything Everywhere All at Once" text (Fig 1-C) causes the browser to navigate to page 2 (Fig 1-N).

Next, Susan wants to replicate these extraction steps for the rest of the actors in the cast. To tell ScrapeViz to generalize in this way, **she simply needs to click on a second actor name**, e.g., "Stephanie Hsu" (Figure 1-M). The system then infers that Susan wants to perform the same kind of actions demonstrated for Michelle Yeoh for other actors on the page, illustrated through an updated visualization: a purple border around each actor name on the EEAAO page (Fig 1-N), a new small viewport for each actor page that is visited upon clicking an actor name (Fig 1-R), a blue border around the birthdate on each actor page (Fig 1-Q), and scraped actor names (Fig 1-E) and birthdates (Fig 1-F) added to the output table.

Next, Susan similarly wants to specify that all of the action sequences performed for the EEAAO movie should also be performed for each of the other movies on the page. She does this by clicking "CODA" (Fig 1-A) to give a second movie example, which again results in an updated macro and visualization—orange borders around the movie titles on the top-level page (Fig 1-B) indicate that they will be scraped and clicked like EEAAO, and the resulting movie pages visited are shown in viewports in the middle column (Fig 1-L). The way that actor names were scraped and clicked for the EEAAO movie page will automatically be generalized to the other movie pages, too, as evidenced by the purple borders around actor names on all of the movie pages in the middle column (Fig 1-J). Finally, the way actor birthdates were scraped for EEAAO will automatically be generalized to the rest of the movies and will be performed once those actor pages are rendered. To avoid excess cognitive load, we only show actor pages for the currently selected movie (signified with a thick black border, Fig 1-N) and we only show at most seven actor pages at a time. To view other pages in a group, the user can click to select a different parent viewport or navigate through sibling pages using left and right arrow buttons (Fig 1-P). To collect a particular datum that has not been rendered yet, they can click on its "Collect" button (Fig 1-G) in the output table, which will bring the viewport containing the expected data into view and also bring all of its ancestor viewports into view.

After browsing through the website pages and inspecting the output table, Susan feels confident that the macro is scraping the data she wants. ScrapeViz has allowed Susan to author **nested-loop scraping logic**, and across multiple pages—*for*

[1]https://web.archive.org/web/20230404103018/https://www.imdb.com/search/title/?count=100&groups=oscar_best_picture_winners&sort=year,desc&ref_=nv_ch_osc

[2]IMDb example inspired by Rousillon [5]

each movie, scrape its name and click on it to reach its list of actors; *for each actor*, scrape their name and click on it to reach their birthdate and scrape their birthdate text.

B. Consuming

Imagine Susan shares her macro with a colleague. ScrapeViz's visual representation would allow the colleague to get an overview of the macro's behavior without needing to watch a long and linear execution of the macro, where it may be harder to keep track of which pages were visited and which data was scraped. If she wants to understand exactly what certain data in the table mean and where they came from, she can click on a cell and ScrapeViz will bring its source page into view and specifically scroll to and highlight the data on the page.

V. Implementation

A. Inference

ScrapeViz leverages website structure-based inference methods similar to those in ParamMacros [9]. ScrapeViz is based on two kinds of generalization:

Generalizing across two example UI elements. When the user clicks or scrapes a new UI element, we check if it may generalize with any UI element the user has previously clicked or scraped. We use the approach from ParamMacros that leverages structural patterns in the website DOM [23] to search for a generalized *XPath formula* that matches all specified UI elements (e.g., the newly scraped "Stephanie Hsu" and the previously scraped "Michelle Yeoh" from Figure 1). This formula must have the form `/prefix/index/suffix`, namely, the only difference between the two elements' XPaths being the index of a single node. If a generalized XPath of this form is found, we then use that formula to enumerate the other matching elements on the page. For example, the index-based XPath for "Michelle Yeoh" is `/html/.../div[`**`1`**`]/div[2]/a` and for "Stephanie Hsu" is `/html/.../div[`**`2`**`]/div[2]/a`, so a generalized XPath that matches both of them is `/html/.../div[`**`index`**`]/div[2]/a`.

Generalizing across pages. We also generalize actions across pages. For example, after the user has generalized the macro to scrape all movie titles from the top-level page (Fig 1-B) and open viewports for each movie page (Fig 1-L), ScrapeViz now generalizes to replicate all actions from the original "Everything Everywhere All at Once" page to each of the other movie pages in column L – namely, to scrape all the actor names. ScrapeViz simply applies the same generalized actor XPath formula `/html/.../div[index]/div[2]/a` to each of these other movie pages.

B. Interface

ScrapeViz is implemented as an Electron desktop app [24]. This supports a key requirement—the ability to embed multiple live website viewports at a time. We specifically use Electron's WebView component [25], which is a wrapper around Chromium and enables rendering any target website.

VI. Study Design

We conducted a within-subjects lab study to evaluate the usability and usefulness of ScrapeViz.

A. Participants

We recruited 12 participants (6 men, 6 women; aged 22–52, median 28.5) with varying levels of programming experience: one participant with none, three with less than 1 year, two with 1–2 years, four with 2–5 years, and two with 5–10 years. We compensated participants with a $40 USD Amazon gift card.

B. Protocol

1) Rousillon: We compared ScrapeViz to Rousillon [5], another PBD tool which similarly enables distributed hierarchical web scraping but with differences, e.g., visually illustrates generalization only within a single page and not across multiple pages, only conveys program representation after the user has explicitly ended their demonstration, shows only one website page at a time, does not link output table to source pages.

2) Reading: Participants completed one reading task using ScrapeViz and one using Rousillon. For each tool, we presented participants with a seven minute tutorial video.

Task websites: We chose macros that mostly worked but also exhibited edge cases, to require users to carefully inspect the macros and websites to identify these anomalies.

- WTA tennis[3]: A macro that scrapes players' last names from a top-level page containing a table of player data. The macro also visits each individual player's page and scrapes their age and their coach's last name.
- Wayfair furniture[4]: A macro that scrapes furniture products' names from a top-level catalog page. The macro also visits each individual furniture product's page and scrapes a list of "variations" (e.g., fabrics, colors, orientations).

Task instructions: We asked participants to describe what data were scraped, what anomalies they saw (if any), and why they believed these anomalies occurred.

3) Authoring: Participants completed one authoring task using ScrapeViz and one using Rousillon. First, we had participants complete a brief tutorial task to ensure familiarity with the scraping interaction, demonstration, and generalization.

Task websites and instructions: We asked participants to scrape the following data:

- Google Scholar[5]: researcher names, the titles of their papers, the text of the paper's PDF/HTML link. This data was spread across three levels of pages.
- Yelp[6]: restaurant names, their hours for today, a list of their most popular dishes. This data was spread across two levels of pages.

[3]https://web.archive.org/web/20231012210012/https://www.wtatennis.com/stats

[4]https://web.archive.org/web/20220324013917/https://www.wayfair.com/furniture/sb0/sectionals-c413893.html

[5]https://web.archive.org/web/20230324030019/https://scholar.google.com/citations?view_op=view_org&hl=en&org=8515235176732148308

[6]https://web.archive.org/web/20230429225251/https://www.yelp.com/search?find_desc=Pizza&find_loc=New+York%2C+NY

VII. Results

Participants appreciated ScrapeViz's visual and interactive nature, enabling them to get an overview of macro behavior, quickly understand the source of scraped data and anomalies, and verify in-progress authoring.

A. Reading

1) Understanding what data is scraped: Presenting multiple website pages with scraped elements highlighted helped users discover the pattern of what was being scraped – "from a number of examples I can see...*okay, this is the last name of the tennis player, this is their age*" (P3). This also helped participants understand the high-level actions performed and the hierarchy of pages visited – "It was helpful to see how [pages are] grouped together so I can say that this is what I'm iterating through in the first layer... and so on" (P6).

2) Identifying source of data and anomalies: Participants found clicking on cells in ScrapeViz's interactive table highly useful – "clicking on the value actually takes you to...where that particular data point came from. I think that's great, just being able to investigate, especially if you see something wonky like we did with actually both cases when there were errors" (P9). In contrast, participants noted that with Rousillon, in order to inspect potential anomalies, they had to align the output table with website content and manually navigate.

3) Navigation: ScrapeViz helped participants keep better track of pages as they navigated – "[Rousillon] was opening up a new tab in the browser or you were having to navigate to each page individually, you kind of lost that initial context, whereas I felt like [with ScrapeViz], you still retained some of that context... I could see the path that I was taking through the webpages a little bit more clearly" (P5).

B. Authoring

Rousillon presents scraping results only after the user has stopped recording demonstrations and then run the generated macro. ScrapeViz actually starts scraping and generalizing immediately as the user provides demonstrations. Many participants found this real-time scraping helpful because it gave them feedback that they were giving the correct number of examples and that their generalization worked as intended – "If I want to get all the authors' name and after I select the second one, [ScrapeViz will] highlight all the other authors for me. So I'm pretty sure I did the right thing, but for [Rousillon], because I cannot get visual feedback after I select the first one, so I would be a little bit worried that I did something wrong" (P11).

C. ScrapeViz – multiple website pages

Many participants did comment that the current interface can at times feel overwhelming with too many website pages, and instead suggested showing fewer parallel sibling pages at once and hiding the rest. Some participants also commented on how small the non-active website pages are and how it can be hard to see all of the relevant content; future work should explore how to highlight the most relevant content for users within these small viewports.

D. Threats to validity

ScrapeViz and Rousillon presented anomalies differently for the reading task websites we chose – ScrapeViz presented N/A for missing values; Rousillon either included incorrect values or skipped rows completely. In part due to this, participants found it easier to identify anomalies in ScrapeViz. However, our interviews with participants still highlight other more meaningful differences between the tools, as described above.

VIII. Discussion and Future Work

A. More advanced AI

ScrapeViz uses heuristics for identifying parallel UI elements (i.e., XPath formulas) and when to show a page in the current viewport versus a new one (i.e., a new viewport only for a new URL). It will be important to explore approaches for more robustly understanding UI element and page similarities, e.g., using language or vision [26] based machine learning for comparing the textual content, structure, or appearance of UIs.

Recent large language model (LLM) powered tools (e.g., MultiOn [27], Adept [28]) seek to support users in automating web tasks through natural language alone, but are prone to hallucinations. A visual macro representation like ScrapeViz may help users verify automation produced by such agents, especially if acting on multiple elements or across pages.

B. Visualizing intermediate within-page actions

ScrapeViz only works meaningfully for click actions that navigate to new page URLs. Clicks that perform stateful operations (e.g., deleting an item) or navigate within page (e.g., opening a pane) will all be performed within a single viewport, resulting in the viewport effectively only showing the final page state (and not intermediate ones). Future work should consider how to visualize such in-page action sequences (e.g., storyboard graphic, GIF) and when to show them (e.g., for all items on a page or only one at a time) to avoid information overload and adhere to space constraints. Future work should also explore ways for users to preview generalized stateful operations before they are performed (since erroneously performed stateful operations can be detrimental).

C. Checking for correctness across numerous webpages

Future work may explore helping users more quickly uncover patterns across pages, without needing to check all sibling pages in a group. One approach may be to cluster pages by visual or DOM similarity or scraping results similarity.

IX. Conclusion

We present ScrapeViz, a new programming by demonstration tool for authoring and understanding distributed hierarchical web scraping macros. ScrapeViz provides users a storyboard-like visual representation of their macro, illustrating the sequences of pages visited, UI elements scraped, and generalizations across elements and pages, and links this to an interactive output table. Through a lab study we saw ScrapeViz helped participants get an overview of macro behavior, quickly understand the source of scraped data and anomalies, and verify in-progress authoring.

Acknowledgments

We thank our study participants for their time and effort, and our anonymous reviewers for their feedback which has helped improve the paper. Thank you also to Fei Wu who helped provide usability feedback.

References

[1] "Shortcuts user guide," https://support.apple.com/guide/shortcuts/welcome/ios, accessed: 2022-06-06.

[2] 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.

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

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

[5] S. E. Chasins, M. Mueller, and R. Bodik, "Rousillon: Scraping distributed hierarchical web data," in *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 2018, pp. 963–975.

[6] R. Dong, Z. Huang, I. I. Lam, Y. Chen, and X. Wang, "Webrobot: Web robotic process automation using interactive programming-by-demonstration," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2022.

[7] K. Pu, R. Fu, R. Dong, X. Wang, Y. Chen, and T. Grossman, "Semanticon: Specifying content-based semantic conditions for web automation programs," in *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, 2022, pp. 1–16.

[8] W. Chen, X. Liu, J. Zhang, I. I. Lam, Z. Huang, R. Dong, X. Wang, and T. Zhang, "Miwa: Mixed-initiative web automation for better user control and confidence," in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.

[9] 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, pp. 1–10.

[10] G. Little, T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan, "Koala: capture, share, automate, personalize business processes on the web," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2007, pp. 943–946.

[11] G. Leshed, E. M. Haber, T. Matthews, and T. Lau, "Coscripter: automating & sharing how-to knowledge in the enterprise," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 1719–1728.

[12] 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.

[13] 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.

[14] 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.

[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] K. Pu, J. Yang, A. Yuan, M. Ma, R. Dong, X. Wang, Y. Chen, and T. Grossman, "Dilogics: Creating web automation programs with diverse logics," in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.

[17] T. Lau, "Why programming-by-demonstration systems fail: Lessons learned for usable ai," *AI Magazine*, vol. 30, no. 4, pp. 65–65, 2009.

[18] R. Krosnick and S. Oney, "Promises and pitfalls of using llms for scraping web uis," in *Workshop.(April 2023). https://doi.org/10.1145/nnnnnnn. nnnnnnn*, 2023.

[19] "Beautiful soup," https://www.crummy.com/software/BeautifulSoup/, accessed: 2022-06-05.

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

[21] J. Wong and J. I. Hong, "Making mashups with marmite: towards end-user programming for the web," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2007, pp. 1435–1444.

[22] D. F. Huynh, R. C. Miller, and D. R. Karger, "Enabling web browsers to augment web sites' filtering and sorting functionalities," in *Proceedings of the 19th annual ACM symposium on User interface software and technology*, 2006, pp. 125–134.

[23] "Document object model (dom)," https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/, accessed: 2021-06-29.

[24] "Electron," https://www.electronjs.org/, accessed: 2020-09-18.

[25] "Web embeds," https://www.electronjs.org/docs/latest/tutorial/web-embeds, accessed: 2023-10-21.

[26] 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.

[27] "Multion - your personal ai agent," https://www.multion.ai/, accessed: 2023-10-22.

[28] "Adept - useful general intelligence," https://www.adept.ai/, accessed: 2023-10-22.