Leveraging graphical user interfaces to facilitate selection of elements in web-pages

Paul van Wijk
TU Delft
Delft, The Netherlands

ABSTRACT

Web-based interaction logging is an important concept for understanding user behavior on web-pages. LogUI is a powerful modern framework for logging a user's interactions. Integrating such a framework in web-pages requires the construction of configuration objects to define selectors that indicate which elements on a web-page should be under observation. Writing such configuration object can be a tedious task for researchers who are less experienced in programming. Therefore, this paper addresses the potential of a graphical user interface (GUI) to simplify the creation of configuration objects. An experimental GUI-based tool was devised to aid a user in the process of producing configuration objects in an interactive fashion. Results from conducting a small scale user study show that users are capable of creating configurations utilizing the GUI-based tool with an average accuracy of 70% measured in terms of selector equivalence. User experience evaluation shows that the tool is perceived as efficient.

KEYWORDS

graphical user interfaces, web-based interaction logging, LogUI, selectors

1 INTRODUCTION

With the increase of web-based applications [17] researchers developed an interest in understanding why users follow certain paths in web-based information systems. Therefore, researchers have proposed an *information foraging* model to provide insight into user behavior in information systems [16]. To gather data for analyses such as these, there exist several tools or frameworks which can track a user's interactions on a web-page.

One tool for analyzing such user's behavior is LogUI [14]. As web-pages can become quite complex, it becomes challenging to realize a generic interaction logging framework which can be integrated into any web-page. Therefore, frameworks such as LogUI require a certain amount of configuration which is often done in a text-based manner, for instance by writing out configuration files. This comes with a few disadvantages. Primarily, some comprehension of a web-page's internal structure is required to be able to specify which event should be observed on a particular element. Secondly, The user is responsible for applying the correct syntax and semantics to the configuration. This may be a complex task and may cause the application to fail at runtime if the produced configuration is incorrect or incomplete. These disadvantages can make similar frameworks less accessible to researchers who are less experienced in programming yet still desire to use such a framework.

CSE3000, June 27th, 2021, TU Delft
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/N/A

There have been studies on simplifying human-computer interaction with the help of graphical user interfaces (GUI) or visual programming approaches such as the Alice programming environment [6] which allow tackling a programming problem in a visual way. Such tools make it more convenient for a user to configure a complex framework for their particular use case without writing code. The research in this paper takes a similar approach and goes into the analysis and development of an experimental tool to visually aid researchers in creating configuration objects. A GUI is considered to facilitate the selection of events and elements on a web-page in an interactive "point-and-click" fashion which in turn allows for correct element identification and tracking by the LogUI framework. We aim to answer the following research questions:

RQ1 Can a GUI-based tool be used to facilitate the construction of configuration objects?
RQ2 Can users of the GUI-based tool create an accurate configuration object?
RQ3 How do users perceive the GUI-based tool?

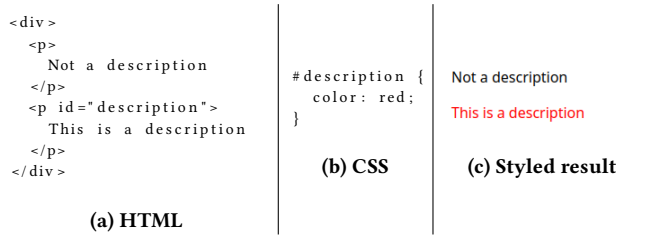
2 RELATED WORK

2.1 Web-page structure

Currently, *HyperText Markup Language* (HTML) is the technology used for developing web-pages (alternatively referred to as documents) as it is standardized by the W3C [1]. These HTML documents are interpreted by web-browsers which yields a tree-like model called the *Document Object Model* (DOM). This model provides the necessary interface to access and modify HTML elements and their associated events.

2.1.1 Selectors. In order to query one or more nodes in the DOM tree, one can rely on *selectors* as a means to express which nodes to select [5]. Selectors are primarily used in *Cascading Style Sheets* (CSS) to bind style properties to the elements matching the pattern defined in the selector. An example of such a selection process is illustrated in Figure 1. The `#description` selector in 1b selects the element in the DOM which has the ID `description` attached, resulting in the second paragraph element being colored red. However, the usefulness of selectors goes beyond style sheets and can be used by any application that requires access to specific DOM nodes.

2.1.2 Specificity. A selector has a *specificity* value based on the amount of ID's, classes (or pseudoclasses) and element names that constitute the selector. Specificity values determine which selector gets preferred over other selectors selecting the same element. The selector with the higher specificity takes the precedence. For instance, the selector `ul#results > li.result:nth-of-type(1)` has a specificity of $\{1, 2, 2\}$ since it contains one ID (`#result`), two



(a) HTML **(b) CSS** **(c) Styled result**

Figure 1: Applying style on the element with the `description` ID using a selector

classes (`.result` and the pseudoclass `:nth-of-type`) and two element names (`ul` and `li`).

Although a selector with a higher specificity value will likely yield a smaller set of selected DOM nodes than a selector with a lower specificity, this is not always the case as it depends on the elements available in the DOM. For instance, the selector `li:nth-of-type(1)` has a specificity of {0, 1, 1}. However, another selector `.active` having a lower specificity ({0, 1, 0}) than the former selector can select less elements depending on the amount of elements in the DOM having a class of `active`. In this regard, specificity does not behave linearly towards the amount of DOM elements selected.

There exist tools which require and manipulate DOM nodes by using selectors, such as browser extensions. One example from the domain of advertisement blocking is *uBlock Origin* [11]. This tool is a widely used browser extension which provides an element picker to aid a user in creating a so-called *cosmetic filter* by enabling them to interactively select elements on the currently loaded web-page. Filters such as these are expressed in selectors and are subsequently used to filter out the selected elements from a page. Additionally, a slider interface facilitates the refining of specificity. The experimental tool developed for this research follows the same philosophy and is heavily inspired by uBlock Origin.

2.2 Web-Based Interaction Logging

Interaction logging is an important concept in the field of web-based *interactive information retrieval* (IIR) systems [14]. An interaction log, generated by capturing a set of events and user interactions with the DOM of a web-based application can be analyzed to provide further insight into user behavior and usability of the application. Therefore, many frameworks and tools have been developed for capturing user interactions such as *Search-Logger* [19], *WHOSE* [10], *YASFIIRE* [22] and *UXJs* [20]. Maxwell and Hauff [14] mention that these tools come with certain drawbacks such as installation of supplementary software, utilization of a proxy server, and integration issues with modern web-applications. LogUI is a recent logging framework which mitigates many shortcomings of the aforementioned frameworks.

2.3 LogUI

LogUI is a logging framework which consists of a client and a server component operating together in capturing a user's interaction with a web-page or web-application [14]. The client is integrated into (web-) applications by completing a series of steps [4], which involves writing out a JavaScript *configuration object*. A minimal representative configuration object is shown in listing 1. This object contains two required configurations, `logUIConfiguration` and `trackingConfiguration`. The former contains basic properties needed to establish a connection to the LogUI server. Furthermore, it contains the `browserEvents` configuration that defines which browser-wide events to capture. LogUI currently supports six browser-wide events which can be tracked such as mouse movement events. The `trackingConfiguration` object defines in detail which events on DOM elements of the web-page should be tracked by LogUI. This is accomplished by specifying and associating event listeners to a set of selectors which will trigger on user interactions with the corresponding elements. Subsequently, this configuration object definition will be utilized by the LogUI client bundle injected into the web-application.

```
configurationObject = {
  logUIConfiguration: {
    endpoint: 'ws://logui/endpoint/',
    authorisationToken: '',
    verbose: true,
    browserEvents: {
      ...
    }
  },
  trackingConfiguration: {
    'searchfocus': {
      selector: '#search-box',
      event: 'focus',
      name: 'SEARCH_FOCUS',
    },
      ...
  }
}
```

Listing 1: LogUI Configuration Object

Completing these steps and correctly formulating the necessary selectors for every experimental interaction logging system can become a tedious and repetitive task for researchers. Therefore, an alternative method to accomplish these tasks could be favorable.

2.4 Graphical user interfaces

Graphical user interfaces (GUI) have become ubiquitous over the last decades and are currently a predominant way to interact with computer applications. A fair amount of GUI's have been developed in academia and have proved to be helpful in assisting novice users in accomplishing complex tasks by providing a visual way of tackling the problem. Examples of such a tool is *EXPGUI* which provides a platform independent GUI-based tool to facilitate interaction with the *Generalized Structure and Analysis Software* [21]. *Deducer* [7] is another example of a GUI which significantly reduces the effort of performing analyses in the statistical programming language R. Finally, *Zbl-build* [15] is a GUI which assists a user in creating configurations for BibLaTeX .

Web-based applications can also be classified as GUIs and have been shown to be effective in aiding users to accomplish complex tasks in a visual way. For instance, [13] have developed a web-based tool for running experiments in the field of molecular dynamics.

2.5 GUI generation

GUI generation is a term commonly applied to techniques which transform some general data model or task description into an interactive GUI and can reduce the time and effort of manually developing a GUI. This has been accomplished in different ways. Jelinek and Slavik [12] have shown a method to generate GUIs by analyzing GUI directives in source code annotations without explicitly defining the GUI itself. Fischer et al. [8] have developed "Brassau, a virtual assistant which can transform natural language commands into GUIs". Various different templates of common assistant tasks were used to generate these GUIs.

For this research, a similar approach is taken and is described in section 3 where the DOM of a web-page is used as an input model to generate an interactive user interface. Such an approach has the potential that the tool can be utilized on any DOM thus being a general tool.

3 LOGUI BROWSER EXTENSION

The most popular desktop web-browsers allow support for *extensions* allowing developers to provide additional functionality to the browser itself and any page content loaded in the browser [3]. To devise a GUI generator, a Chrome browser extension has been developed as a tool to select elements on the currently loaded web-page. The tool serves as a means to facilitate the construction of configuration objects for the LogUI framework by following a point-and-click philosophy. This process is illustrated in figure 2.

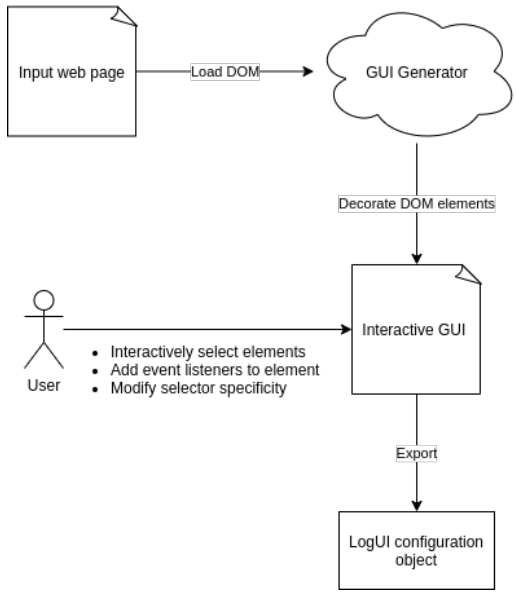


Figure 2: Process of creating configuration objects using a generated GUI

3.1 Overview

A GUI generation approach is accomplished by attaching additional event listeners to the DOM elements of the current web-page enabling an element picking mode. When the cursor is positioned over any element on the web-page, the user is provided with immediate feedback indicating which element will be selected when a mouse-click follows. This is attained by displaying a highlight over the *bounding client rectangle*[1], i.e. "the smallest rectangle containing the entire element". An example usage of the element picker functionality on a Search Engine Result Page (SERP) is shown in figure 3.

Figure 3: Element picker

Succeeding the selection of an element using the picker, the user is presented with an interface serving to complete the tracking configuration object as required by LogUI. An instance of the interface is illustrated in figure 4. In (1), the name of the tracking event is entered which will be utilized by the LogUI framework to label this event in the log. Subsequently, the specificity level of a selector can be adjusted using the slider in (4). Alternatively, if the desired specificity level is not obtainable by using the slider, the user can specify their own selector by entering one in the text field (2). Both approaches provide immediate visual feedback indicating which elements on the page will be selected when the selector is modified. In the example, the `.gs_rs` selector will select every description of search results as is indicated by the highlighted elements (3). Lastly, the desired event listener is associated with the tracker by making a choice from a set of predefined event names (5).

3.2 Basic Configuration Popup

In order to visualize the current state of the configuration object, the extension features a *popup*. This component displays a view of the two configuration models described in 2.3. Furthermore, the popup allows users to modify the properties of the configuration object and presents which values are required in an intuitive way. The basic entries consist of the web-socket endpoint URL, authorization token and verbose mode. Furthermore, six trackable browser-wide events can be enabled or disabled for logging by utilizing on/off switches. Finally, the popup contains buttons to activate the element picker, to delete previously added trackers, and to export the current state of the configuration object into a JavaScript Object Notation (JSON) file.

[1] https://developer.mozilla.org/en-US/docs/Web/API/Element/getBoundingClientRect

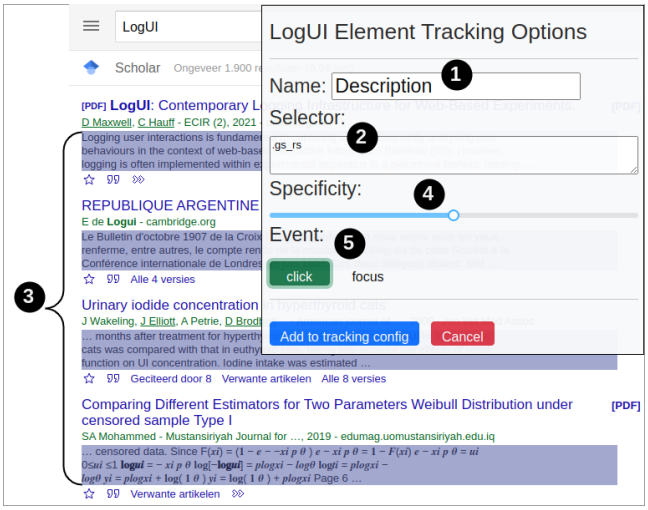


Figure 4: Tracking configuration editor. Numbered items are explained in section 3.1

3.3 Element Selection

The usual method to conceive selectors is by writing these down. This is normally accomplished by individuals who have a good understanding of the web-page's structure. Non-expert users may be less likely to know the correct selector for one or more particular elements. Therefore, the extension simplifies this process by providing an element picker (figure 3). The picker becomes active on the currently loaded page by the user's command. The aim of this picker is to generate a selector based on the element picked by the user. Such a selector can be used in future interaction logging experiments which require logging the selected element.

Selectors are typically used to define which elements on a web-page to select. However, in the case of using an element picker for constructing selectors, this is the reversed process. In this paper, selectors are deduced from a user's selection of one or more elements on a web-page.

3.4 Specificity Levels

Picking a DOM element using the aforementioned element picker is the first step of constructing the selector. However, it might occur that the user does not require solely the picked element but all elements of the same type. One example use case is when a selector is required which selects every list item in an ordered list. This is an indication that the selector resulting from the picker might need refinement. Therefore, *specificity levels* are introduced. Each level defines a strategy used to decide the structure of the resulting selector.

The extension provides a set of six predefined levels of specificity which are available for selection with the slider (4) shown in figure 4:

(1) The selected element's ID. This is the most specific type of selector as it exclusively selects the element on the page which has the ID attached.
(2) The structural location in the DOM tree. If an element has no ID or classes attached, a selector for this particular element can be resolved by traversing up the ancestors in the DOM tree. When this traversal reaches an ancestor that has an ID attached, this ancestor can be used as a "checkpoint". While traversing further up the DOM tree can increase the specificity of the resulting selector, this is redundant as the selector will select the same element due to the ID contributing significantly to the specificity. Selectors generated in this level are composed as a path-like structure from the checkpoint to the element.
(3) Sibling types. This is useful in scenarios when the user requires selecting siblings of the same element type, such as all `li` elements in a `ul`.
(4) Explicit classes. It can occur that the selected element has attached classes.
(5) Element types. It may be required to select all elements in the DOM which are of the same element type, for instance, every anchor (<a>) element.
(6) The universal selector `*`. This selector has the lowest specificity value and selects every element. This may be useful in cases where it is required to attach an event listener to every element in the DOM.

For other use cases in which users are not able to reach the desired level of specificity, it is possible to enter the selector manually in the text box (2) shown in figure 4.

3.5 Limitations

Contemporary web-pages can convey different appearances and behaviors across browsers due to the implementation of multiple different standards. For this research, exclusively the desktop versions of the Chrome and Chromium browsers are considered.

The specificity levels defined in 3.4 are primarily based on heuristics. There can be special use cases where the user requires a complex selector which can not be formed using any of the predefined strategies. Finding if the current amount of specificity levels is either sufficient, insufficient or superfluous is a topic open for future research.

The prototype of this tool does not cover every aspect of LogUI. A configuration object with the minimum required settings can be obtained using the tool whereas LogUI offers significantly more features which can be defined in the configuration object. For instance, *application specific data* and additional *metadata* per tracker may be specified for logging.

4 METHOD

To answer RQ1, the tool in chapter 3 was developed as a prototype and superficially tested on arbitrary web-pages. To gain further insight in the usage of the tool and to answer RQ2, a small scale user study was set up to examine selectors created by research participants. Furthermore, to address Q3, the user experience is studied.

4.1 Participants

To conduct the user study, five participants committed to the experiment. The participants are PhD students and experts in the field

of *web information systems* which makes them possible candidates to use LogUI. Three participants did not have previous experience with the LogUI framework and were given a brief explanation of the configuration object prior to the experiment. All participants were given instructions to install the extension prototype in their own Chrome or Chromium browser. For each participant, a short fifteen minute online interview was conducted.

4.2 Task Description

During the experiment, participants were given a set of tasks to perform by interacting with the extension. A simple web-page [2] was conceived for participants to select elements from using the tool.

The first task involved entering the basic LogUI configuration values. A short requirement story was given from which the participant can extract the necessary entries. Successful entry of these items constitute a well-formed `logUIConfiguration` object.

Subsequently, in order to cover a fair amount of selector types in the experiment, participants were given the task of including six diverse tracking configurations in their configuration object. Each task was formulated as a natural language phrase:

(1) Add a *click listener* to the third result. Give it the name `thirdresult`.
(2) Add a *click listener* to the search box. Give it the name `searchbox`.
(3) Add a *click listener* to all search results. Give it the name `searchresult`.
(4) Add a *click listener* to the first advertisement. Give it the name `firstad`.
(5) Add a *focus listener* to any advertisement. Give it the name `adfocus`.
(6) Add a *focus listener* to the search button. Give it the name `searchfocus`.

The reason for using natural language in the task description is that this approach eliminates the need for thinking in programming terms such as "selectors" or "elements" and diverges more into the semantics of the task. After fully constructing the configuration object, participants were asked to export it to a JSON file and submit this file for further analysis.

4.3 Responsible Research

For every participant it was explicitly requested if the session may be recorded prior to the start of the recording. It was made clear which data will be collected for the research, which were the configuration object produced by the participant and their questionnaire response which were both anonymized prior to their analysis. The required tasks did not put the participant at risk. The source code for the experimental tool is made available on GitHub [3] and can be used to reproduce the results of the conducted experiment. However, due to human unpredictability, exact results may differ.

There were no conflicts of interest conducting this research.

[2] https://pjvanwijk.github.io/fakesearch.html
[3] https://github.com/pjvanwijk/LogUI-config-builder

4.4 Metrics

Two metrics are captured in the user study. Namely, the *accuracy* and the *user experience*. Both are described in this section.

4.4.1 Accuracy. To perform accuracy measurements of a selector generated by a participant, it is necessary to define suitable metrics that can give insight towards the usefulness of employing a graphical user interface for creating selectors. Geneves et al. [9] present a formalization for style sheets. Though we are less concerned with style sheets, the analysis provided in this paper is useful for comprehending the workings of selectors. A similar approach is taken when reasoning about the equivalence of selectors later in this section.

Firstly, a predefined calibration standard, henceforth referred to as the *gold standard* configuration object is an object of which the selectors are guaranteed to select the elements as required in the task description. It is necessary that participants have no prior knowledge of the gold standard when conducting the experiment as such knowledge could influence the decisions made. The selectors generated by the participants are compared against the gold standard based on two accuracy metrics, particularly the *selection accuracy* and *specificity accuracy*. The former metric is necessary to acknowledge that the user-generated selector selects the same elements on the experimental web-page as the accompanying selector defined in the gold standard. The latter metric is necessary to guarantee that the user-generated selector does not select more elements than is required by the task description.

Consider the following scenario; it may be desired to obtain a selector which selects a particular button, for instance a search-button on the web-page. However, it might occur that this is the only button on the page. In this case, a user can create the `button` selector which will yield the correct resulting element set for the current page and therefore is considered accurate in terms of selection accuracy. However, such a selector will select *all buttons* on the web-page and is hence considered insufficiently specific. For a completely accurate selector, both the selection accuracy and the specificity accuracy need to be tested for correctness. A selector declining either metric is considered to be inaccurate.

Hence, to measure the accuracy of a selector, a comparison is made between the user-generated selector and its corresponding gold standard value. The method is based on the "equivalence of selectors" test from [9]. This is implemented by applying both selectors to the experimental web-page and verify that the resulting selections of elements are equal. Finally, the specificity value of both selectors are calculated and compared to verify that the user-generated selector has a specificity value equal to or higher than the gold standard selector's specificity.

4.4.2 User Experience. The user experience is measured using the *User Experience Questionnaire* (UEQ) [18]. Such a questionnaire is useful for obtaining an immediate insight into user experience by analyzing feedback from the user of the tool. Participants were requested to fill in the questionnaire after they fulfilled the task of constructing a configuration object. The questionnaire contained 26 questions concerning the attractiveness, perspicuity, efficiency, dependability, stimulation and novelty aspects of the experimental tool. Four additional open-ended questions were appended to the

questionnaire regarding the functionality of the tool. This was done particularly to gain insight in the usability of the predefined specificity levels. However, it was not mandatory to answer these questions. The following questions were asked:

(1) What type of selectors do you consider important but were not possible to select using the specificity slider?
(2) Does the specificity slider provide too much, too few or enough levels of specificity to choose from?
(3) Were there any tasks you particularly struggled with when constructing the configuration object?
(4) Do you have any other remarks or suggestions?

5 RESULTS

In this section, results obtained from the user study are presented. All five participants completed the tasks described in the previous section. Out of the five participants, four have responded to the user experience questionnaire. The results are presented relating to the research question they contribute to.

5.1 RQ2. Accuracy

An aggregated visualization of the resulting selectors produced by the participants is shown in figure 6. Highlights are positioned over the bounding client rectangle of the selected element, captions represent the name of the selector as requested in the task description. A higher intensity of highlighting (darker) indicates that more participants created selectors that select the highlighted element.

To examine the accuracy of the selectors created by participants, the following table 5 shows a summary of the obtained results from running the experiment with five participants. The amounts in the *# Selection* column represent the number of participants that created a selector which selects an equal set of elements as the corresponding selector defined in the gold standard. Likewise, the *# Specificity* column represents the number of participants that created selectors with equal to or higher specificity than its relating gold standard.

The minimum of selection and specificity $min(selection, specificity)$ is exactly the amount of participants that was able to constitute a completely accurate selector for the corresponding tracking configuration, i.e. a selector which selects the required elements and has the required specificity. The overall accuracy of the tracker is defined by dividing $min(selection, specificity)$ by the number of participants.

It can be observed that participants are generally able to accurately create selectors with an average score of 70%. However, it is necessary to criticize the methodology used to obtain this result. Since the tasks carried out by the participant are described in natural language phrases, there may be semantic ambiguities when describing a selector as such a phrase. A participant may interpret the task wrongly which could contribute to inaccurate selectors.

Ambiguities may also arise from the composition of elements in the web-page. In some cases, web-pages contain groups of elements which serve a common purpose. For instance, a "search result" on a search engine result page can be interpreted in multiple ways. Questions might emerge such as "is the description also part of the search result?". It might be challenging to clearly indicate which element contributing to such a component should be selected. The same applies to list-like structures. The natural language description "list of results" can be interpreted to mean either the list element containing all results, or all result elements that are contained by the list element. Such different interpretations will lead to distinct selectors. A similar misinterpretation can be observed in the accuracy of `adfocus`. The term "any advertisement" could be understood to mean one or more advertisements.

Tracker	# Selection	# Specificity	Accuracy
`thirdresult`	5	5	1.0
`searchbox`	4	4	0.8
`searchresult`	3	5	0.6
`firstad`	4	5	0.8
`adfocus`	1	4	0.2
`searchfocus`	4	4	0.8

Figure 5: Resulting selector accuracy for 5 participants. Values represent the amount of participants who created an accurate selector. Undefined selectors are considered inaccurate.

5.2 RQ3. User Experience

The results from the conducted user evaluation questionnaire are analyzed by creating a benchmark allowing us to draw a conclusion about the tool's user experience in general. To accomplish this, the summarized data is compared against a benchmark data set provided by [2]. This particular data set contains data from 21175 individuals originating from 468 studies concerning general products such as "business software, web-pages, web-shops and social networks." The resulting benchmark is shown in figure 7.

It can be observed that the mean score of the questionnaire is excellent in all aspects. However, caution must be applied when interpreting this score by the fact that the sample size of representative participants was very small ($n = 4$). Another observation that can be made is that the mean score shows a peak in the efficiency aspect. This might be an indication that the GUI tool presented in this research could be beneficial for researchers.

As previously stated, the participants were asked additional open-ended questions so that the user's opinion on the heuristics applied to the specificity levels can be evaluated. To (1), half of the responses stated that the slider generated appropriate selectors for the tasks. Three suggestions were given regarding selector types which are not obtainable by using the slider. Namely, the click selector, selectors containing class names and selectors testing for substrings in text. To (2), the general answer was that there is a sufficient amount of specificity levels. Two responses mention that it depends on the web-page. To (3), two participants indicated to experience difficulties in creating the configuration object. Particularly, one response mentioned a difficulty selecting more than one (element) at once. Another stated a difficulty producing the selector for the `adfocus` tracker which involved the usage of the slider being counter-intuitive.

In the next section, the obtained results will be discussed and evaluated to draw conclusions and obtain answers to the research questions.

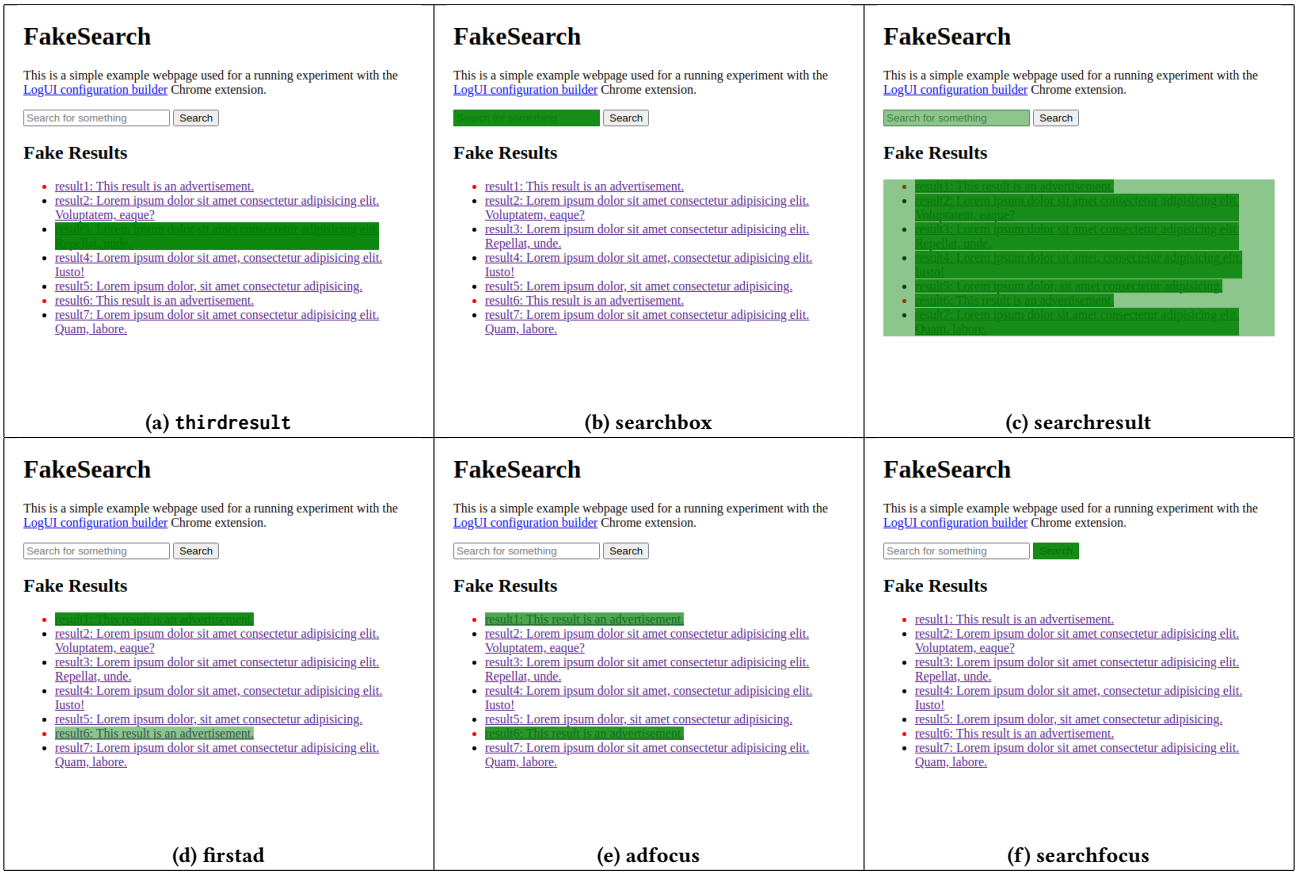


Figure 6: Overview of selectors created by participants. Green highlights indicate the chosen elements for the corresponding selector

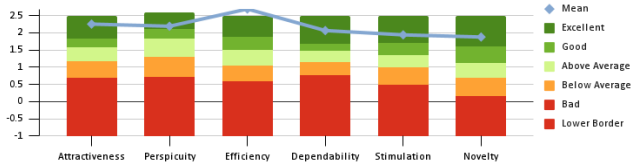


Figure 7: Benchmark of the tool based on the UEQ responses of 4 participants

6 DISCUSSION, CONCLUSIONS AND FUTURE WORK

RQ1. By the results shown in the previous section it has been shown that users are able to create configuration objects which involve selectors by using a graphical user interface. However, in practice the prototype tool is not perfect. Superficial manual testing the tool on several well-known web-pages such as the DuckDuckGo[4] search engine and the BBC News[5] pages has shown that there are various challenges that need to be addressed when generating a GUI from the DOM. Due to the dynamic nature of the web-page it may be difficult to reach certain elements using an element picker interface. For instance, elements in a drop-down menu are not possible to pick as doing so would require clicking on it and thus conflicts with the click event of the picker.

Another issue is that there exist multiple ways to implement event listeners on DOM elements, making it more difficult to allow the element picker's events to take precedence over the ones already associated with the element.

A final major issue that was discovered during one of the interviews was that other extensions may conflict with the experimental tool. For instance, there are extensions such as Grammarly [6] which add class names to elements of the currently loaded DOM. These classes may interfere with the generation of selectors as these classes can then appear in the selector whilst they should not.

RQ2. With the results it has been shown that users are able to create selectors with an average accuracy of 70%. However, there are some flaws in the methodology regarding ambiguities in the task descriptions. This is worth to consider when performing user studies in the future that involve natural language tasks.

[4]https://duckduckgo.com
[5]https://www.bbc.co.uk/news
[6]https://www.grammarly.com

RQ3. The results of the UEQ have shown that users generally find the tool useful in terms of user experience. However, as the study conducted with a small amount of participants, it might be convenient to conduct further research on the user experience of the experimental tool presented in this paper.

6.1 Limitations

There are a few limitations to this work. The GUI described in this research was developed as an extension for the Chrome and Chromium browser. However, it could be extended to work in other browsers or applications as well. Furthermore, the tool supports construction of bare minimum versions of LogUI configuration objects and does not account for some optional configurations such as logging additional metadata for specific selectors.

REFERENCES

[1] [n.d.]. HTML Standard. https://html.spec.whatwg.org/multipage/.
[2] [n.d.]. User Experience Questionnaire (UEQ). https://www.ueq-online.org/.
[3] [n.d.]. What Are Extensions? https://developer.chrome.com/docs/extensions/mv3/overview/.
[4] 2021. Logui-Framework/Client. The LogUI Framework.
[5] Tantek Çelik, Elika J. Etemad, Daniel Glazman, Ian Hickson, Peter Linss, and John Williams. 2018. Selectors Level 3. https://www.w3.org/TR/2018/REC-selectors-3-20181106/.
[6] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: A 3-D Tool for Introductory Programming Concepts. *Journal of Computing Sciences in Colleges* 15, 5 (April 2000), 107–116.
[7] Ian Fellows. 2012. Deducer: A Data Analysis GUI for R. *Journal of Statistical Software* 49, 1 (June 2012), 1–15. https://doi.org/10.18637/jss.v049.i08
[8] Michael Fischer, Giovanni Campagna, Silei Xu, and Monica S. Lam. 2018. Brassau: Automatic Generation of Graphical User Interfaces for Virtual Assistants. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3229434.3229481
[9] Pierre Geneves, Nabil Layaida, and Vincent Quint. 2012. On the Analysis of Cascading Style Sheets. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. Association for Computing Machinery, New York, NY, USA, 809–818. https://doi.org/10.1145/2187836.2187946
[10] Daniel Hienert, Wilko van Hoek, Alina Weber, and Dagmar Kern. 2015. WHOSE – A Tool for Whole-Session Analysis in IIR. In *Advances in Information Retrieval (Lecture Notes in Computer Science)*, Allan Hanbury, Gabriella Kazai, Andreas Rauber, and Norbert Fuhr (Eds.). Springer International Publishing, Cham, 172–183. https://doi.org/10.1007/978-3-319-16354-3_18
[11] Raymond Hill. 2021. Gorhill/uBlock.
[12] Josef Jelinek and Pavel Slavik. 2004. GUI Generation from Annotated Source Code. In *Proceedings of the 3rd Annual Conference on Task Models and Diagrams (TAMODIA '04)*. Association for Computing Machinery, New York, NY, USA, 129–136. https://doi.org/10.1145/1045446.1045470
[13] Sunhwan Jo, Taehoon Kim, Vidyashankara G. Iyer, and Wonpil Im. 2008. CHARMM-GUI: A Web-Based Graphical User Interface for CHARMM. *Journal of Computational Chemistry* 29, 11 (Aug. 2008), 1859–1865. https://doi.org/10.1002/jcc.20945
[14] David Maxwell and Claudia Hauff. 2021. LogUI: Contemporary Logging Infrastructure for Web-Based Experiments. In *Advances in Information Retrieval*, Djoerd Hiemstra, Marie-Francine Moens, Josiane Mothe, Raffaele Perego, Martin Potthast, and Fabrizio Sebastiani (Eds.). Vol. 12657. Springer International Publishing, Cham, 525–530. https://doi.org/10.1007/978-3-030-72240-1_59
[15] Guido Milanese. 2015. Zbl-Build: A GUI Interface for Biblatex. (2015), 4.
[16] Peter Pirolli and Stuart Card. 1999. Information Foraging. *Psychological Review* 106, 4 (1999), 643–675. https://doi.org/10.1037/0033-295X.106.4.643
[17] Gustavo Rossi, Matias Urbieta, Damiano Distante, Jose Matias Rivero, Sergio Firmenich, Gustavo Rossi, Matias Urbieta, Damiano Distante, Jose Matias Rivero, and Sergio Firmenich. 2016. 25 Years of Model-Driven Web Engineering. What We Achieved, What Is Missing. *CLEI Electronic Journal* 19, 3 (Dec. 2016), 5–57. https://doi.org/10.19153/cleiej.19.3.1
[18] Martin Schrepp, Andreas Hinderks, and Jörg Thomaschewski. 2014. Applying the User Experience Questionnaire (UEQ) in Different Evaluation Scenarios. In *Design, User Experience, and Usability. Theories, Methods, and Tools for Designing the User Experience (Lecture Notes in Computer Science)*, Aaron Marcus (Ed.). Springer International Publishing, Cham, 383–392. https://doi.org/10.1007/978-3-319-07668-3_37
[19] Georg Singer, Ulrich Norbisrath, Eero Vainikko, Hannu Kikkas, and Dirk Lewandowski. 2011. Search-Logger Analyzing Exploratory Search Tasks. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11)*. Association for Computing Machinery, New York, NY, USA, 751–756. https://doi.org/10.1145/1982185.1982350
[20] Jaime Solís-Martínez, Jordan Pascual Espada, Rubén González Crespo, B. Cristina Pelayo G-Bustelo, and Juan Manuel Cueva Lovelle. 2020. UXJs: Tracking and Analyzing Web Usage Information With a Javascript Oriented Approach. *IEEE Access* 8 (2020), 43725–43735. https://doi.org/10.1109/ACCESS.2020.2977879
[21] Brian H. Toby. 2001. EXPGUI , a Graphical User Interface for GSAS. *Journal of Applied Crystallography* 34, 2 (April 2001), 210–213. https://doi.org/10.1107/S0021889801002242
[22] Xing Wei, Yinglong Zhang, and Jacek Gwizdka. 2014. YASFIIRE: Yet Another System for IIR Evaluation. In *Proceedings of the 5th Information Interaction in Context Symposium (IIiX '14)*. Association for Computing Machinery, New York, NY, USA, 316–319. https://doi.org/10.1145/2637002.2637051