Developing Contemporary Web-Based Interaction Logging Infrastructure: The Design and Challenges of LogUI

David Maxwell[1], Claudia Hauff[1]

[1]*Delft University of Technology, The Netherlands*

Abstract

Studies involving user interfaces typically involve the capturing and recording *(logging)* of key user interactions between the user and the system being examined. However, anecdotal evidence suggests that researchers often implement their own logging infrastructure—sometimes in a piecemeal fashion—which can lead to numerous implementation mistakes (due to misunderstanding or ignoring differences between *web browsers*, for example). While efforts have been made to develop interaction logging solutions for experimentation and commercial use, many solutions either use obsolete technology, are prohibitively expensive, are complex to use (and require extensive programming knowledge), or have no source code available. To address these issues, we have developed LogUI, an easy-to-use yet powerful interaction logging framework that can capture virtually any user interaction within a web-based environment. LogUI has been successfully used in several user studies since its launch. This paper provides an in-depth discussion into how we have designed LogUI, and provides narrative on the key challenges that we are looking to address moving forward.

1. Introduction

Web applications [17, 26] are commonplace in our daily lives. The interplay that takes place between a user and a web application are in essence a series of interactions with the *elements* that make up the *webpage.* This webpage is rendered by the user's *web browser.* Within the web browser, the elements are represented internally as part of the *Document Object Model (DOM)* [12, 34]. A user of a contemporary web application will typically make hundreds or thousands of interactions (such as clicking on an element) with the rendered webpage's DOM during its lifespan. For example, a recent user study reported by Roy et al. [52] observed 799 recorded interactions (e.g. mouse clicks) on average, over 120 participants. This average value was observed over a prescribed ten minute period, where participants interacted with a simple *Search Engine Results Page (SERP).*

In order for researchers to gather insights into how users make use of a given web application's interface, we need a means to *capture and record* (or *log*) the aforementioned interactions that take place [1]. This requirement is a cornerstone of most forms of *usability research* [32], and is indeed important for the *Interactive Information Retrieval (IIR)* community. In order to accurately and reliably capture a user's interactions, we require the use of *web-based interaction logging software*—something

which is difficult to implement correctly without extensive knowledge of the many developmental nuances.

Anecdotal evidence (at least within the IIR community) suggests that researchers often work on developing their own web-based interaction logging software. This is often achieved in tandem with the implementation of their main experimental apparatus [18, 46, 47, 61]. However, we argue that this is highly undesirable. Lessons we have learned from our own experiences—and observations from others in the community—suggest that developing logging infrastructure is non-trivial, with researchers bypassing issues such as network latency (for sending captured events to a server); misunderstanding specific implementation details between different browsers (such as how they interpret events in the DOM) [53]; or simply forgetting to capture key interactions. Ultimately, these pitfalls may lead to low quality *interaction logs,* with missing and/or noisy (or *"raw"* [23]) data. From this, inevitable *post-hoc* frustrations will arise. While a series of separate interaction logging solutions have been developed, we argue that we lack existing, easy-to-use, decoupled, affordable, and up-to-date implementations that are readily available for researchers. Such a solution must abstract away the inevitable complexities for generating high quality interaction data.

To this end, we have implemented LogUI, a complete solution for contemporary web-based interaction logging over *any* web-based application. From our initial presentation of LogUI at *BCS-IRSG ECIR 2021* [36], researchers have successfully used LogUI in several user studies (e.g. the study reported by Roy et al. [52]). We have acquired feedback on how to improve the infrastructure further, and are excited about many future development chal-
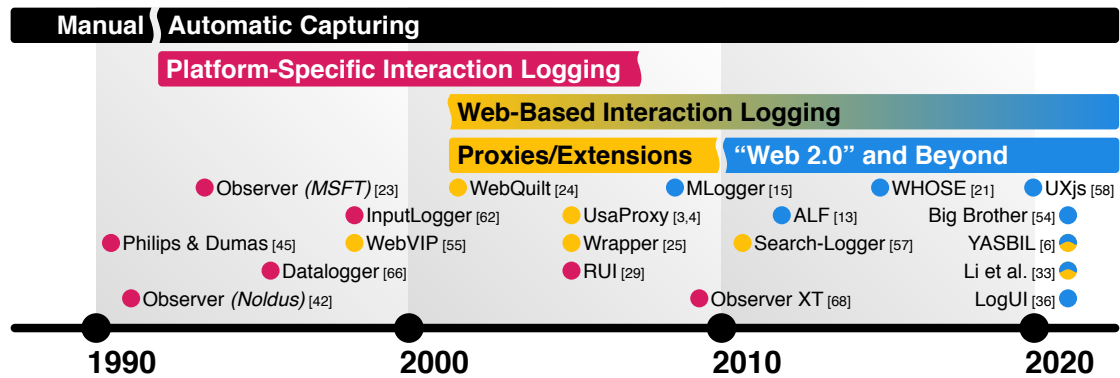
Figure 1: A timeline of user interaction logging solutions proposed—complete with *select* implementations. Time span cutoffs are approximate. Colours correlate to the different *types* of solution, which in turn roughly correlate to a given *era*.

lenges to extend its functionality. This paper discusses these challenges—both past and future—along with an overview of existing interaction logging systems. LogUI is open source, includes detailed documentation, and is available for use at https://www.logui.net.

2. Interaction Logging History

Interaction logging software has a long history in computing science literature, with pioneering implementations appearing long before the establishment of the *World Wide Web (WWW)* [5] as the *de facto* communications medium. Early discussions on the development of interaction logging software date back as far as the early 1990s [23, 45], when *Graphical User Interfaces (GUIs)* were becoming commonplace, with competitors catching up to the benchmark of the classic *Apple Macintosh System Software* [67]. GUIs result in more complex interface designs—and were increasingly *modeless*[1] in nature [23]. This section provides a high-level overview of the seminal attempts undertaken in this research area, leading up to the fully automated and web-based interaction logging solutions of today.

2.1. From Manual to Automatic

Increasing interface complexity means that capturing what users do becomes more of a challenge. Before interaction logging software were established, the only realistic way to perform *usability testing* [32] was for a researcher to record a participant's interactions with a pen and paper [23]. This is known as *manual capturing* [10]. Paper records were often supplemented with

---

[1]*Modeless* interface designs contain a number of components (e.g. dropdown menus, multiple windows, dialog boxes) instead of more simplistic, procedural, state-based interfaces—which were typically text-based.

video recordings of participants interacting with the system in question [19, 50], permitting researchers to replay interactions *post-hoc* [66]. Hoiem and Sullivan [23] however note several drawbacks to this approach: keeping pace with interactions was difficult—as was searching through them. The *Hawthorne effect* [28] could also influence how participants used the system with observers watching. Issues were compounded with the need to turn around analyses in ever shorter time frames [11].

These concerns led to the first generation of interaction logging systems, as shown in Figure 1 around the early 1990s. These were *automatic* in nature [56], whereby a system not only supported the running of some system, but could also log the raw user events taking place—such as key presses on a keyboard, or movement/clicks of a mouse (where used). In reference to their experiences of building such software, Philips and Dumas [45] argued that interaction logging systems should: *(i)* allow researchers to determine exactly *what* to capture; *(ii)* capture and record so-called *discrete* events (such as when a session starts, or the press of a key); and *(iii)* be able to adequately *code* the event. These early recommendations—along with those reported by other researchers—are used as motivations for the functionality of LogUI.

2.2. 1990s-2000s: Platform-Specific Solutions

The following decade saw a number of *platform-specific* implementations. These captured a user's interactions with (sometimes specific) *native applications* running on a platform—with solutions developed for *MS-DOS, Microsoft Windows,* and *Apple's Mac OS X/macOS.*

In addition to the work by Philips and Dumas [45], Hoiem and Sullivan [23] outlined their implementation—an internal project developed at *Mi-*

crosoft.[2] They discussed how their solution evolved over a number of iterations. Separate work by Noldus [42] reported on a MS-DOS solution that could capture and save keyboard interactions to a plaintext file. Both solutions were confusingly called *Observer*, with the latter still in development today as *Observer XT* [68]. Observer XT captures mouse and keyboard events, and has been used by several researchers [14, 59, 63]. Other select solutions include *Datalogger* [66], *InputLogger* [62], *Recording User Input (RUI)* [29], and *AppMonitor* [1]. Aforementioned solutions recorded fine-grained, *"raw"* data [23]—something that, without proper context (i.e. missing identifying *names* like CLICK_SEARCH_BUTTON), is highlighted as *too detailed* for researchers [23, 45].

Towards the start of the *noughties,* researchers were beginning to investigate how the Internet could be used to send interaction logs from native applications to a central server [22]. This *client-server* approach to logging profoundly increased the volume of logs that could be captured. Advancements were made possible thanks to improving networking capabilities at the time [44].

2.3. The WWW and DOM

The noughties saw the rise of the WWW [5]. A number of web-based interaction logging solutions have been proposed over the years, with early solutions such as *Listener* [16] and *NIST*-sponsored *WebVIP* [55] solutions focusing more on navigation logs *(what pages did a user visit?)* rather than interaction logs *(what did they do on each page?).* Vendors of web browsers now broadly follow *web standards,* as dictated by the *W3C*—although minor differences do exist between implementations (which still cause issues for developers). Web browsers use now ubiquitous technologies, such as *HyperText Markup Language (HTML)*, *Cascading Style Sheets (CSS)*, and *client-side scripting (ECMAScript, or JavaScript).* One such technology that is core to how browsers interpret webpages is the *Document Object Model (DOM).*

The DOM, as outlined by Dogac et al. [12], is a tree-like, object-oriented *Application Programming Interface (API)* that is is used by programs that interpret HTML documents [34]. The tree-like structure is derived from each document's source, and DOM objects—called *elements*—are linked together through inheritance. Each element contains a number of properties, such as `class` names. *CSS selectors* can be used to apply styling to change the appearance of elements as rendered within the browser's *viewport*; JavaScript engines interpret code that programmatically manipulates the DOM over the lifespan of a webpage in the browser. Following the *event-driven programming* paradigm, developers track

---

```
#box.addEventListener("click", function() {
    alert("The blue box was clicked.");
});
```
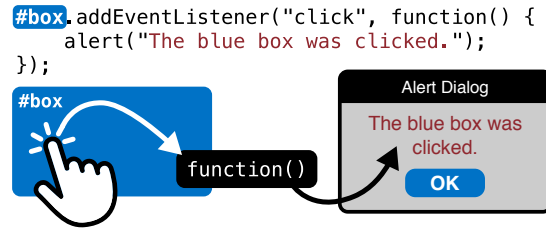

Figure 2: A DOM `EventListener` example. Code binds the `#box` element and the `click` event together. When this event occurs, the associated `function()` is executed.

interactions between users and webpages by *binding* one or more `EventListeners` to DOM elements, which in turn are triggered when a given interaction takes place—as illustrated in Figure 2. Besides standard keyboard and mouse events, new DOM events are frequently added to support ever more diverse input means and device functionalities. Examples include *screen orientation* and *touch events; web APIs* also include viewport recording capabilities, going full circle back towards aforementioned video recordings of interactions [1, 23].

Despite clear advancements in web technologies making interaction logging possible, there are still many nuanced differences between browsers that trip up developers of such software. Events can be interpreted in different ways; developers themselves can misunderstand (or simply miss!) key implementation details, such as the subtle differences between `mouseover` and `mouseenter` events. Further challenges are outlined throughout §3.

2.4. 2000s-Today: Web-Based Solutions

As WWW technologies began to mature, it became feasible to run user experiments through a web browser. Early web-based interaction logging solutions captured interactions on a webpage through the approach outlined in §2.3, but recording them was a challenge—asynchronous approaches for connecting to a server (like *AJAX*) were not widely available until the mid-noughties. Instead, an intermediary *proxy server*, sitting between the client (hosting the user's browser) and server (hosting the web application) was used. The proxy server would inject JavaScript into the webpage as it was served to the client, with the first recorded example using this approach being *WebQuilt* [24]. For many years, this was the preferable solution—a user's navigation history could be logged to the same server (the proxy server), avoiding *Cross-Site Scripting (XSS)* [49] issues. The seminal *UsaProxy* [3, 4] then followed, this time using AJAX to record interactions *within* a page. This approach was used in a number of studies [2, 8, 9, 30]. Further proxy-based solutions included *PooDLE* [7], *RWELS* [56], and *YASFIIRE* [65]. Proxy-based solutions were preferred over the use of
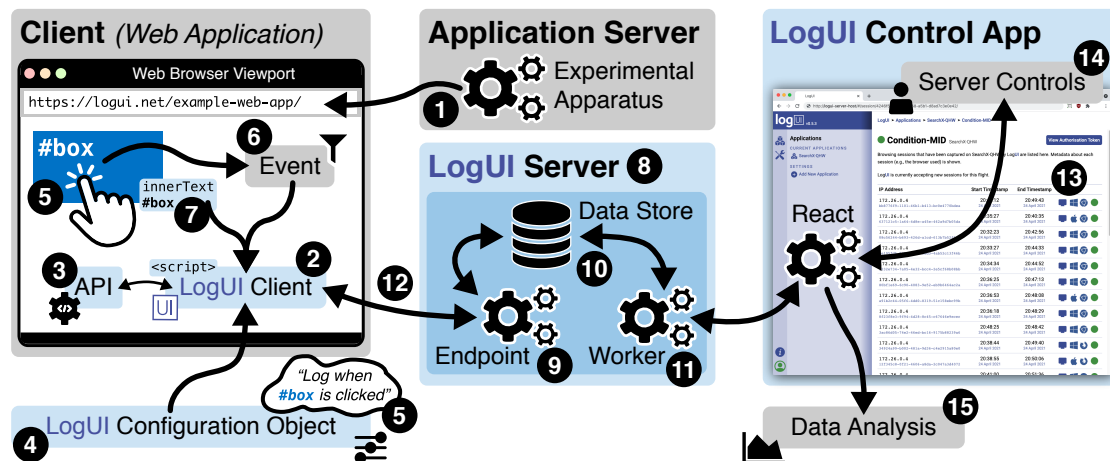
Figure 3: Architectural diagram of LogUI. Blue boxes denote components specific to LogUI; refer to §3.2 for more information. Circled numbers are used throughout §3 to highlight certain concepts; these pertain to the numbers in the illustration above.

browser extensions. Users would often be reluctant to use these thanks to their historical reputation of being vectors for malware delivery [60]. Despite this, notable examples of solutions using this second approach include *Wrapper* [25], *Search Logger* [57], and the *Lemur Toolbar*.

The most recent phase of web-based interaction logging software follows UsaProxy [3, 4] in making use of so-called *Web 2.0* [43] technologies, such as AJAX and advancements in client-side scripting. Feature-rich APIs and improved client-server connectivity eliminated the need for proxy- and extension-based solutions. Examples of these solutions include *MLogger* [15], *WHOSE* [21], *SurfLogger* [20], and *ALF* [13]. However, the pace of development is such that many of these solutions are now themselves dated. This is compounded with source code no longer being available for many.

Nonetheless, further solutions have recently been developed such as *UXJs* [58] and *Big Brother* [54]. These solutions work well, but capture *all* events occurring on a page without filtering, which may result in interaction logs that are too *"raw"* [23]. *YASBIL* [6] and the solution by Li et al. [33] are further recent examples, using contemporary technologies within a browser extension to record more longitudinal experimental data. As an extension however, support depends on the browser being used. Questions also remain over their ability to work with client-side web applications that manipulate the DOM extensively (even reworking the loaded DOM to the point that it appears as if an entirely new page has been loaded!)—such as those written with *React*.

This concern is also present with commercially available tools, such as *Google Analytics*, *Hotjar*, or *Matomo*. Indeed, Google Analytics is largely designed for the cap-

ture of *coarse-grained interactions*, such as page viewing history [48, 58] (as opposed to fine-grained interactions, such as mouse clicks on a specific page). In addition, Matomo requires elements one wishes to capture interactions for to be given a specific `class` [58]. In essence, one can argue that this increases coupling between the logging infrastructure and apparatus.

3. LogUI

LogUI is designed to offer a complete, end-to-end solution for capturing and recording a user's interactions with a given web application. It abstracts many of the complex design decisions to the point that researchers and developers can simply embed LogUI into their apparatus, and start using it. LogUI comes with both a *client* and *server,* each with their own responsibilities. The design decisions behind LogUI have been motivated from both our experiences and the recommendations of other researchers [23, 45, 56]. Note that circled numbers ❶ refer to Figure 3 (illustrating the architecture); squared letters Ⓐ refer to Figure 4 (illustrating a log entry).

3.1. Key Features of LogUI

The fundamental principle of LogUI is that of *less is more.* From experience, we have found that if *too much data is captured, it can be overwhelming.* Our principle here (as opposed to other solutions like Big Brother [54] or UXJs [58]) is to enable researchers to capture *only what they need* (at the expense of greater complexity)—and *provide event context* (through event coding [45]). As previously mentioned, Hoiem and Sullivan [23] highlighted

this in the context of data being too *"raw"*—such that data were so fine-grained that it were not immediately useful for analysis. This problem (amongst others) is addressed via a number of novel constructs within LogUI.

Configuration Object   Central to our attempts of tackling the above problem is the *configuration object* ❹. This provides *custom event coding functionality* [45], and is central to the LogUI client. The configuration object is comprised of a series of rules which state exactly *what should be captured, and when.* CSS selectors and *CSS specificity rules* are used to determine what listeners are *bound* to what elements (e.g. ❺ uses element #box and the click event). Names are used for bindings, so that eventual logs report these names for easy identification of a logged event Ⓐ.

Metadata   One of the limitations of decoupling interaction logging software from the main experimental apparatus is that researchers lose the ability to record the *internal state* of the said apparatus with captured events. LogUI provides a solution to this problem with so-called *metadata*. Metadata is unique to the element being interacted with, and can be appended to the captured event with ease. As an example, Ⓒ shows the value of an input box, which represents query terms entered by a user. We can derive data from a number of *sources,* such as the corresponding DOM element's attributes and/or its properties, or even from the *state* and/or *props* of the underlying React component ❼ (if React is used—support for other frameworks can be added in the future).

Application-Specific Data   Similar to metadata defined above, we also provide researchers with the ability to append *application-specific data* to *all* captured events. These data are global to the context in which a web-based application is run, and has been used for capturing data such as the *user ID, group ID,* and *interface variant,* as shown by Ⓑ. These data (as well as metadata) can then be easily filtered *post-hoc* to report statistics, for example, over a given experimental condition.

Event Groupings   While researchers are free to use whatever DOM events they wish (e.g. click), we also provide a number of so-called *event groupings.* These abstract from native, browser-level events—and provide additional logic to remove much of the complexity of reliably and accurately capturing certain events. For example, we have implemented a mousehover grouping, which internally listens for mouseenter and mouseleave events, and logs them separately. Furthermore, a scrollable grouping listens for scroll events on a page and/or element, reporting only *start* and *end*

events. Existing solutions may record a number of superfluous, confusing intermediary scroll events, leading to noisy (and potentially confusing) data.

Browser Events   Contrasting to events which are captured from interactions with DOM elements, LogUI also supports a number of so-called *browser events.* Not pertaining to a specific element, events include the resizing of the browser's viewport, a gain/loss of *focus* to the viewport window, and a change in the URL in the address bar. More are detailed in the documentation.

3.2. Architecture Overview

LogUI utilises the *client-server architecture*—with the client responsible for *capturing* interaction data, and the server currently responsible for *receiving and storing* it for later access by researchers.

3.2.1. LogUI Client

The LogUI client can be used within any web-based application, hosted on some web application server ❶. Built in a highly modular way, the LogUI client uses the contemporary Node.js JavaScript ecosystem to create a *Browserified* bundle. This bundle is then dropped into the target application via a <script> tag ❷.

Source components of the LogUI client each have their own distinct responsibilities; a noteworthy example is the Binder component, which automatically binds EventListeners to the elements in the DOM, as selected by researchers.[3] The component has a bespoke algorithm which parses the configuration object to work out what DOM elements should be targeted. In addition, it also uses the contemporary MutationObserver web API to listen for changes to the DOM, and automatically appends listeners to new elements as necessary. As such, this one feature alone makes LogUI an ideal fit for contemporary web frameworks like React—and as it considers the DOM directly, it is also completely *framework agnostic.* The client uses the principle of *unobtrusive capture* [56]; LogUI functions silently, with users unaware that it is recording their interactions.

The LogUI client is controlled through a simple API ❸ that provides functionality for researchers to start and stop interaction logging as and when required (amongst other functionality). The only requirement is for a configuration object to be present ❹. When active, interactions on DOM elements that were bound to ❺ are interpreted by the browser ❻ and passed to the LogUI client. Metadata and application-specific data are then appended to captured events ❼ if they are to be stored; the final result is then sent to the LogUI server ❽.

---

[3]The Binder binds events to a specific function within LogUI, and does not interfere with EventListeners from elsewhere.

3.2.2. LogUI Server

The LogUI server's ⑧ primary purpose is to receive incoming captured events from the LogUI client—after a connection has been established (§3.2.3). LogUI server has been *fully containerised,* meaning that a new instance of the server can be easily started on any installation with the correct version of Docker installed. Full documentation is provided on how to do this.

The server *endpoint* ⑨ accepts connections from instances of the LogUI client, and stores incoming data in backend data stores ⑩. At present, we use an instance of *MongoDB* to store data, where each log entry is stored as a *JSON* object. We also include a *worker* process ⑪ within the containerised environment. This exposes a web server hosting the LogUI control application (§3.4). This web server is only intended to be accessed by researchers using the given instance of LogUI server. Both the endpoint and worker processes use the *Django* framework to provide the necessary infrastructure for fulfilling requests. Like the LogUI client, the design of the server is modular in nature—with specific responsibilities for each component.

3.2.3. Client-Server Protocol

Communications between the LogUI client and server are achieved via *WebSockets* [39] ⑫ to provide for a full-duplex channel between them. This contemporary solution is preferable to AJAX-based implementations, as WebSockets are not subject to the *Same Origin Policy,* meaning the LogUI server can reside elsewhere from the page's host. WebSockets also guarantee that data will arrive at the recipient in the order in which it was sent—and with only one connection needing to be established when the LogUI client starts, bandwidth and other resource requirements are reduced [56].

Sitting on top of the WebSocket connection is the LogUI protocol; an application-specific series of rules to enable the client and server to be able to understand one another. The protocol includes a *handshaking* routine, whereby the LogUI client sends an encrypted *authorisation token* which, when decrypted, provides information about the web application that the LogUI client is watching. This helps the LogUI server know who is connecting. Connections are terminated by the server if the web application is unrecognised.

The protocol also considers what happens when the connection is lost. If this happens, the client will continue to capture all events (and cache them). While doing so, it will also attempt to reconnect to the server. When reconnection is successful, the client will immediately send its cache to the server (after the initial handshake). This process ensures that no loss of captured interaction events result when temporary connectivity issues occur.

```json
{
  "eventType": "interactionEvent",
  "eventDetails": {
    "type": "submit",
    "name": "FORM_SUBMISSION"    A
  },
  "timestamps": {
    "eventTimestamp": "2021-04-24T20:40:27.052Z",
    "sinceSessionStartMillis": 416981,
    "sinceLogUILoadMillis": 416981
  },
  "applicationSpecificData": {    B
    "userId": "5847e60f73170700013697c6",
    "groupId": "60848097ae062cf467fdbc37",
    "variant": "mid"
  },
  "metadata": [{                  C
      "name": "QUERY_VALUE",
      "value": "wildlife extinction"
  }],
  "applicationID": "APPLICATIONID",
  "flightID": "FLIGHTID",
  "sessionID": "SESSIONID"        D
}
```

Figure 4: A recorded log entry, adapted from the user study reported by Roy et al. [52]. Here, a query has been submitted.

3.3. Recorded Interaction Logs

LogUI produces interaction log files in JSON format; logs can easily be parsed by any analysis library that supports JSON, such as *Pandas.* Each recorded log entry corresponds to a unique JSON object; details of this output are available as part of the LogUI documentation. Refer to Figure 4 for an example of a logged entry.

Of particular interest in the illustrated log entry example are the values for the `applicationID`, `flightID`, and `sessionID` keys D. These denote to the concepts of an *application, flight,* and *session,* respectively. In LogUI parlance, an application is the representation of a single web application (e.g. experimental apparatus). A flight is a variant of an application; think of a user study with three conditions. In this case, the given application would have three flights. Finally, a session is the grouping of a single user's interactions, over one or more webpages pages, examined in a single sitting. These key/value pairings are appended to *every* log entry, such that filtering by either of the three is trivial to achieve.

3.4. LogUI Control Application

LogUI also comes with an elementary *control application* ⑬. This React-based application allows researchers to login to an instance of LogUI server, and control various aspects of the server ⑭—such as creating applications and flights, or viewing session details and authorisation tokens. Further functionality includes the ability for researchers to download complete logs for a given flight, which can then be used in subsequent data analysis ⑮.

4. Experiences of Using LogUI

Since its initial release in March 2021, LogUI has been successfully used (to the best of our knowledge) in five unique research projects, one of which is now published [52]. Each of the researchers who used LogUI relayed feedback about their experiences in informal discussions. We now outline some of issues raised by the researchers, some of which are elaborated on in §5.

Fast Setup A total of three experimental systems have been adapted to work with LogUI so far. In addition, LogUI has also been integrated with *Jupyter Notebooks* to capture interactions within its interface (such as logging what *cells* are being interacted with). Researchers who have used LogUI have highlighted the ease of being able to get everything required up and running quickly with their experimental system. This success has been attributed to the the easy-to-follow documentation, which has a step-by-step guide for setting up LogUI.

Integrating Experimental Systems with LogUI One of the experimental systems integrated with LogUI is SearchX [46], a large-scale, modular, open-source search framework designed as the apparatus to be used in high-quality crowdsourced IIR-based user studies. Indeed, the study reported by Roy et al. [52] used SearchX to provide a straightforward SERP (consisting of *ten blue links* plus a small *interface widget*) to crowdsourced workers—with LogUI being used to record interactions by the workers within the said SERPs. Five distinct interfaces were trialled, with each interface logged as part of a unique LogUI flight. Additional application-specific data were supplied to LogUI, allowing for the recording of interactions alongside the unique `userId` **B** assigned by the crowdsourcing platform. A LogUI configuration object bespoke to the SERP provided by SearchX ensured that all required browser events were captured—with all necessary metadata **C** captured (such as the issued query, or the rank of a clicked document).

Success Stories Indeed, across all systems, researchers have commented on the simplicity of defining *what* interactions to capture through the configuration object (§3.1). Of course, being well-versed in web technologies, we would imagine that this may be more of a hindrance for those less experienced. One researcher has noted the ease with which their gathered logs could be parsed—the lack of *"fluff"* (erroneous, or superfluous events) made the *post-hoc* script writing process for data analysis much quicker, and less error-prone. This is a vindication of the *less is more* approach (§3.1), and also of our event groupings which resulted in more compact logs—especially with scrolling and mouse hover interactions, two of the major obstacles we have experienced when conducting user studies in the past [35].

Identified Difficulties Although researchers were ultimately successful with integrating LogUI into their apparatus, this was not however without its difficulties. Researchers at times struggled to figure out where to add the necessary API **3** calls into their codebase to start LogUI. With SearchX being built with React, for example, finding the correct points to start and stop LogUI was crucial, and did require some effort to successfully achieve. This is a crucial point: as we advertise LogUI to be decoupled, more can be done to enable it to work without this (potential) requirement, which we discuss more in §5. However, other approaches we took to mitigate the effects of decoupling logging infrastructure from the core experimental system—such as metadata—were reported to be straightforward to use.

5. Ideas/Remaining Challenges

Despite the achievements in developing LogUI, a number of non-trivial challenges remain that would add value to the project as a whole. From researcher feedback or discussions in the literature, we have identified a series of challenges and potential ideas that we outline below.

Testing Infrastructure While the basic interaction logging capabilities of LogUI are functional, and it has been demonstrated to work over different web browsers, *are all interactions being captured in the same way across each browser?* We know that browser vendors can interpret standards in different ways, leading to slight variations in how web applications look on different platforms [53]. To examine this issue in more detail, we plan to develop an extensive *compatibility testing* [40] framework that automatically reports any differences, allowing us to pinpoint our resources and work to ensure interaction logs are recorded consistently across browsers.

Backwards Compatibility As we outlined above, wide-ranging support a variety of web browsers and platforms is cruicial for success of LogUI. While the initial focus of our efforts has been to ensure LogUI works with all *major, contemporary web browsers,* an instance of LogUI attempting to run on an older browser (say, without support of the `MutationObserver` web API, §3.2.1) is inevitable as we gain traction.[4] To address this shortcoming, we should invest time in developing LogUI to be cater for older browsers. Thanks to readily-available *Polyfills* in the JavaScript ecosystem, patching holes in what a browser supports should be straightforward.

Improving Device Support Work also needs to be undertaken to ensure that support is sufficient for the

---

[4]According to *caniuse.com*, support for the `MutationObserver` is high, at ~97%—but not quite 100%! How would we cater for the 3% without support?

increasing number of events associated with touchscreen devices like smartphones (or other input devices). Examples include multi-touch *gestures* (like *'pinching'* on an image to shrink it or zoom out). Moving away from web-based applications, support for interaction event logging on *native* apps under *Android* or *iOS* would also be beneficial, and increase the potential number of use cases for LogUI even further. A recent patent by Google demonstrates a possible alternative view of interaction logging without necessarily using a web browser [41]. Work by Jeong et al. [27] discusses a framework for capturing interactions on native *Android* apps. (Conversational) assistants and other associated systems/devices—and how one could potentially log interactions with these—should also be considered as part of LogUI.

Analytics Dashboard   One of our major goals with LogUI is to develop a functional and intuitive *analytics dashboard* to allow researchers to examine, filter, and compare interaction logs from one flight against another, for example. Dashboards have been devised for similar projects in the past [21, 27]; our major challenge here is *how to design an analytics interface that caters for any kind of interface layout, or even supports interactions from different devices/systems.* LogUI should not be constrained purely to IIR research; flexibility should be available, but not at the expense of adding complexity. From this, we could also provide functionality to automatically compute *aggregated, segmented,* or *individual* measures [31]. A prototypical analytics interface has already been developed. The interface produces boxplots that visualise the average time taken to perform selected events (e.g. the time between the `focus` of a query box, and the `submit` event for a query to be issued). Feedback will be required from researchers to fully understand what can be manipulated and presented—at both an aggregated and individual (per user) level.

Improving the LogUI Stack   As part of developing LogUI to provide a solution for providing analytics and the visualisation of captured data, we will need to modify the LogUI server-side stack to promote such analyses—especially at scale. At present, the stack simply acquires and stores interaction data for post-hoc download by researchers. For future development, we may follow the path of the *ELK stack,* using *Kibana*, *Logstash*, and *Elasticsearch* to provide the necessary groundwork and functionality for data analysis. We must also consider the scalability of any developed stack, both in terms of the number of users, plus the volume of data that must be examined for any analysis to take place.

Recording Viewports   To complement the proposed changes above, recording the viewport of a user's browser would also be desirable. While there are undoubtedly challenges to capturing, storing and synchro-

nising video against captured events [1, 23], context provided by a video stream of a user's interactions will offer insights as to *why* events took place [1]. The *Screen Capture API* (alluded to earlier) provides the functionality to achieve this without additional software.

Reducing Integration Barriers   As discovered by Hoiem and Sullivan [23], integrating logging software requires considerable knowledge of the software your code integrates into. This is still an issue with LogUI, as highlighted in §4. Our objective is to reduce the need for individuals to write code to integrate LogUI. A major stumbling block at present is the need for the configuration object (§3.1), a necessary trade-off to uphold the *less is more* principle. Taking this further, GUI-based tools to generate this object may be useful (as demonstrated by *uBlock Origin* [37]); or even storing configurations on the server, and sending the correct configuration object for the page in question. An ideal solution would be to develop this line (if indeed possible) to the point where only a solitary `<script>` tag needs to be dropped into the application's template.

Community Engagement   With a number of open-source interaction logging solutions recently presented at conferences [6, 33, 36, 54], interaction logging is clearly an area of interest to other researchers. We have begun to work together to explore whether collaboration with each other is possible, starting with the consideration of a *common interaction log format*—a format that could in theory be used by any future pipeline for interaction log analysis, be it through a GUI or API.

Privacy Concerns   A further important point to raise is how we store data. By capturing user interactions, researchers would be subject to data privacy regulations in their jurisdiction, such as the EU's *GDPR* [64], or California's *CCPA* [51]. We could also explore the potential use of *data pods*, which would allow users to control use and access to stored data. The *Solid Project* is an example of how this could be realised in practice.

6. Conclusions

In this paper, we have presented discussed LogUI, our new web-based interaction logging infrastructure. Based on the recommendations and observations of researchers who have developed similar systems, our implementation has been well-received by several researchers who have successfully used LogUI for their experimentation since its initial release in March 2021. Despite the positive feedback we have received, several exciting ideas for future development have been discussed that will improve the offering that LogUI provides to potential end users. Our end goal for LogUI is to be a complete end-to-end solution for interaction logging, all without the need for researchers to write a line of (analysis) code.

Acknowledgments

References

[1] J. Alexander, A. Cockburn, R. Lobb, AppMonitor: A tool for recording user actions in unmodified Windows applications, Behavior Research Methods 40 (2) (2008) 413–421.

[2] A. Apaolaza, S. Harper, C. Jay, Longitudinal Analysis of Low-Level Web Interaction through Micro Behaviours, in: Proc. 26th ACM HT, 337–340, 2015.

[3] R. Atterer, Logging usage of AJAX applications with the "UsaProxy" HTTP proxy, in: Workshop on Logging Traces of Web Activity, Proc. 15th WWW, 2006.

[4] R. Atterer, M. Wnuk, A. Schmidt, Knowing the User's Every Move: User Activity Tracking for Website Usability Evaluation and Implicit Interaction, in: Proc. 15th WWW, 203–212, 2006.

[5] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, A. Secret, The world-wide web, Comm. ACM 37 (8) (1994) 76–82.

[6] N. Bhattacharya, J. Gwizdka, YASBIL: Yet Another Search Behaviour (and) Interaction Logger, in: Proc. 44th ACM SIGIR, 2585–2589, 2021.

[7] R. Bierig, J. Gwizdka, M. J. Cole, A user-centered experiment and logging framework for interactive information retrieval, in: Proc. 32nd ACM SIGIR, 8–11, 2009.

[8] J. Bigham, A. Cavender, Evaluating Existing Audio CAPTCHAs and an Interface Optimized for Non-Visual Use, in: Proc. 27th ACM CHI, 1829–1838, 2009.

[9] D. Bilal, J. Gwizdka, Children's eye-fixations on google search results, Proc. ASIS&T 53 (1) (2016) 1–6.

[10] M. D. Byrne, B. E. John, N. S. Wehrle, D. C. Crow, The tangled web we wove: A taskonomy of WWW use, in: Proc. 17th ACM CHI, 544–551, 1999.

[11] S. Denning, D. Hoiem, M. Simpson, K. Sullivan, The value of thinking-aloud protocols in industry: A case study at Microsoft Corporation, in: Proc. Human Factors Society Annual Meeting, vol. 34, 1285–1289, 1990.

[12] A. Dogac, I. Durusoy, S. Arpinar, N. Tatbul, P. Koksal, I. Cingil, N. Dimililer, A workflow-based electronic marketplace on the web, ACM SIGMOD Record 27 (4) (1998) 25–31.

[13] M. Doolan, L. Azzopardi, R. Glassey, ALF: A Client Side Logger and Server for Capturing User Interactions in Web Applications, in: Proc. 35th ACM SIGIR, 1003, 2012.

[14] D. A. Ducharme, I. Arcand, Using Noldus Observer XT for research on deaf signers learning to read: An innovative methodology, Behavior Research Methods 41 (3) (2009) 833–840.

[15] A. Edmonds, R. W. White, D. Morris, S. M. Drucker, Instrumenting the Dynamic Web, J. Web Eng. 6 (3) (2007) 244–260.

[16] R. D. Ellis, T. B. Jankowski, J. E. Jasper, B. S. Tharuvai, Listener: A tool for client-side investigation of hypermedia navigation behavior, Behavior Research Methods, Instruments, & Computers 30 (4) (1998) 573–582.

[17] P. Fraternali, Tools and approaches for developing data-intensive web applications: a survey, ACM Computing Surveys (CSUR) 31 (3) (1999) 227–263.

[18] M. Hall, E. Toms, Building a common framework for IIR evaluation, in: Proc. 4th CLEF, 17–28, 2013.

[19] M. L. Hammontree, J. J. Hendrickson, B. W. Hensley, Integrated data capture and analysis tools for research and testing on graphical user interfaces, in: Proc. 10th ACM CHI, 431–432, 1992.

[20] J. He, SurfLogger: A logging browser and data processing method in web-based studies, Proc. Soc. of Computer in Psychology .

[21] D. Hienert, W. van Hoek, A. Weber, D. Kern, WHOSE – A Tool for Whole-Session Analysis in IIR, in: Proc. 37th ECIR, 172–183, 2015.

[22] D. M. Hilbert, D. F. Redmiles, Agents for collecting application usage data over the Internet, in: Proc. 2nd AGENTS, 149–156, 1998.

[23] D. E. Hoiem, K. D. Sullivan, Designing and using integrated data collection and analysis tools: challenges and considerations, Behaviour & Information Technology 13 (1-2) (1994) 160–170.

[24] J. I. Hong, J. A. Landay, WebQuilt: a framework for capturing and visualizing the web experience, in: Proc. 10th WWW, 717–724, 2001.

[25] B. J. Jansen, R. Ramadoss, M. Zhang, N. Zang, Wrapper: An Application for Evaluating Exploratory Searching Outside of the Lab, in: Proc. 29th ACM SIGIR, 2006.

[26] M. Jazayeri, Some trends in web application development, in: Proc. Future of Software Engineering, IEEE, 199–213, 2007.

[27] J. W. Jeong, N. H. Kim, H. P. In, GUI information-based interaction logging and visualization for asynchronous usability testing, Expert Systems with Applications 151 (2020) 113289.

[28] S. R. G. Jones, Was there a Hawthorne effect?, Amer-

ican J. of Sociology 98 (3) (1992) 451–468.

[29] U. Kukreja, W. E. Stevenson, F. E. Ritter, RUI: Recording user input from interfaces under Windows and Mac OS X, Behavior Research Methods 38 (4) (2006) 656–659.

[30] M. Lassila, T. Pääkkönen, P. Arvola, J. Kekäläinen, M. Junkkari, Unobtrusive Mobile Browsing Behaviour Tracking Tool, in: Proc. 4th IIiX, 278–281, 2012.

[31] F. Lettner, C. Holzmann, Automated and unsupervised user interaction logging as basis for usability evaluation of mobile applications, in: Proc. 10th MoMM, 118–127, 2012.

[32] J. R. Lewis, Usability testing, Human factors & ergonomics .

[33] H. Li, H. Lu, S. Huang, W. Ma, M. Zhang, Y. Liu, S. Ma, Privacy-Aware Remote Information Retrieval User Experiments Logging Tool, in: Proc. 44th ACM SIGIR, 2615–2619, 2021.

[34] F. Manola, Towards a richer Web object model, ACM SIGMOD 27 (1) (1998) 76–80.

[35] D. Maxwell, L. Azzopardi, Y. Moshfeghi, A study of snippet length and informativeness: Behaviour, performance and user experience, in: Proc. 40th ACM SIGIR, 135–144, 2017.

[36] D. Maxwell, C. Hauff, LogUI: Contemporary Logging Infrastructure for Web-Based Experiments, in: Advances in IR (Proc. 43rd ECIR), 525–530, 2021.

[37] J. Mazel, R. Garnier, K. Fukuda, A comparison of web privacy protection techniques, Computer Communications 144 (2019) 162–174.

[38] J. McGrenere, The design and evaluation of multiple interfaces: A solution for complex software, University of Toronto, 2002.

[39] A. Melnikov, I. Fette, The WebSocket Protocol, RFC 6455, 2011.

[40] A. Mesbah, M. R. Prasad, Automated cross-browser compatibility testing, in: Proc. 33rd ICSE, 561–570, 2011.

[41] A. K. Moshe, E. Wu, X. Wu, Dynamically Configurable Client Application Activity, U.S. Patent 16/564298, September 2019.

[42] L. P. J. J. Noldus, The Observer: a software system for collection and analysis of observational data, Behavior Research Methods, Instruments, & Computers 23 (3) (1991) 415–429.

[43] T. O'Reilly, What is Web 2.0: Design patterns and business models for the next generation of software, Communications & strategies (1) (2007) 17.

[44] F. Paternò, G. Ballardin, RemUSINE: a bridge between empirical and model-based evaluation when evaluators and users are distant, Interacting with computers 13 (2) (2000) 229–251.

[45] B. H. Philips, J. S. Dumas, Usability Testing: Identifying Functional Requirements for Data Logging

Software, In Proc. Human Factors Society Annual Meeting 34 (4) (1990) 295–299.

[46] S. R. Putra, F. Moraes, C. Hauff, SearchX: Empowering collaborative search research, in: The 41st ACM SIGIR, 1265–1268, 2018.

[47] G. Renaud, L. Azzopardi, SCAMP: a tool for conducting interactive information retrieval experiments., in: Proc. 4th IIiX, 286–289, 2012.

[48] S. Ripp, S. Falke, Analyzing user behavior with Matomo in the online information system Grammis, in: Proc. 18th EURALEX, 87–1000, 2018.

[49] G. E. Rodríguez, J. G. Torres, P. Flores, D. E. Benavides, Cross-site scripting (XSS) attacks and mitigation: A survey, Computer Networks 166.

[50] J. Roschelle, S. Goldman, VideoNoter: A productivity tool for video data analysis, Behavior Research Methods, Instruments, & Computers 23 (2) (1991) 219–224.

[51] M. A. Rothstein, S. A. Tovino, California takes the lead on data privacy law, Hastings Center Report 49 (5) (2019) 4–5.

[52] N. Roy, A. Câmara, D. Maxwell, C. Hauff, Incorporating Widget Positioning in Interaction Models of Search Behaviour, in: Proc. 7th ACM ICTIR, 2021.

[53] T. Saar, M. Dumas, M. Kaljuve, N. Semenenko, Cross-browser testing in browserbite, in: Proc. 14th ICWE, 503–506, 2014.

[54] H. Scells, Jimmy, G. Zuccon, Big Brother: A Drop-In Website Interaction Logging Service, in: Proc. 44th ACM SIGIR, 2021.

[55] J. Scholtz, S. Laskowski, L. Downey, Developing usability tools and techniques for designing and testing web sites, in: Proc. 4th HFWeb, vol. 98, 1–10, 1998.

[56] I. Shah, L. Al Toaimy, M. Jawed, RWELS: A remote web event logging system, J. King Saud University-Computer & Information Sciences 20 (2008) 1–11.

[57] G. Singer, U. Norbisrath, E. Vainikko, H. Kikkas, D. Lewandowski, Search-Logger: Analyzing Exploratory Search Tasks, in: Proc. 26th ACM SAC, 751–756, 2011.

[58] J. Solís-Martínez, J. P. Espada, R. González Crespo, B. C. Pelayo G-Bustelo, J. M. Cueva Lovelle, UXJs: Tracking and Analyzing Web Usage Information With a Javascript Oriented Approach, IEEE Access 8 (2020) 43725–43735.

[59] R. Sun, G. Zhang, Z. Yuan, The Preliminary Application of Observer XT (12.0) in a Pilot-Behavior Study, in: Proc. 15th EPCE, 686–700, 2018.

[60] M. Ter Louw, J. S. Lim, V. N. Venkatakrishnan, Enhancing web browser security against malware extensions, J. Computer Virology 4 (3) (2008) 179–195.

[61] E. Toms, L. Freund, C. Li, WiIRE: the Web interactive information retrieval experimentation system prototype, Information Processing & Management

40 (4) (2004) 655–675.

[62] S. Trewin, InputLogger: General-purpose logging of keyboard and mouse events on an Apple Macintosh, Behavior Research Methods, Instruments, & Computers 30 (2) (1998) 327–331.

[63] A. van Drunen, E. L. van den Broek, A. J. Spink, T. Heffelaar, Exploring workload and attention measurements with uLog mouse data, Behavior research methods 41 (3) (2009) 868–875.

[64] P. Voigt, A. Von dem Bussche, The EU general data protection regulation (GDPR), A Practical Guide, 1st Ed. 10.

[65] X. Wei, Y. Zhang, J. Gwizdka, YASFIIRE: Yet Another System for IIR Evaluation, in: Proc. 5th IIiX, 316–319, 2014.

[66] S. J. Westerman, S. Hambly, C. Alder, C. W. Wyatt-Millington, N. M. Shryane, C. M. Crawshaw, G. R. J. Hockey, Investigating the human-computer interface using the Datalogger, Behavior Research Methods, Instruments, & Computers 28 (4) (1996) 603–606.

[67] G. Williams, Apple Macintosh computer., Byte 9 (2) (1984) 30–31.

[68] P. H. Zimmerman, J. E. Bolhuis, A. Willemsen, E. S. Meyer, L. P. J. J. Noldus, The Observer XT: A tool for the integration and synchronization of multimodal signals, Behavior research methods 41 (3) (2009) 731–735.