Leipzig University
Faculty of Computer Science
Degree Programme Informatik

Simulation of Web-Users for Online Discourse Analysis

Bachelor's Thesis

Kai Knappik
Born Jul 08, 1998 in Leipzig

Matriculation Number 3725204

1. Referee: Juniorprof. Dr. Martin Potthast

Submission date: June 13, 2022

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Leipzig, June 13, 2022

.....
Kai Knappik

Abstract

In this paper, we present a modular system that can be used for simulating and archiving user behavior on the internet. Users are described with a specific set of interests, influences, and a list of websites they frequently visit in a user-model. The usermodels can be created, easily understood, changed and then run in the simulation. To evaluate the gathered information, the traced users are made searchable with OCR. In an assessment, a few questions about the users' participation in online discourses were answered to test the simulation. In the current state, the user simulation collects static web content like URLs, video uploads from specific channels or blog posts, but lags the capability of reacting to it dynamically. It can be further improved to various extents. By generating the user's routine based on his interests and influences or expanding and deepening the behavior of the users. Furthermore, we see improvements in processing the evaluation of the trace data the users produce. This bachelor thesis provides reasons for the need of simulating users on the web and also tries to simplify the retrieval of an online discourse for interdisciplinary use.

Contents

Chapter 1

Introduction

The Internet is undisputed one of the most influential media at the time, if not even of all times. Like the archiving of print media is mandatory for knowledge retention and handing it over to upcoming generations, also archiving the internet-accessible content is obligatory for the understanding and obtaining our society [Zelkowitz, 2003, p. 150].
Modern discourse analysis highly depends on digital resources. In social research, a common approach to obtain insights into society is a discourse analysis referring to the work of Michel Foucault. "Discourses determine how something is talked about and how something is not talked about or may/cannot be talked about." (translated) [Pühretmayer and Puller, 2011]. Collecting text, statements, or concepts about those discourses is an essential part of the analysis since all those are expressions of prevailing power relations (conscious or unconscious) [Pühretmayer and Puller, 2011]. Nowadays, we find most of the produced or communicated content online [Hilbert and López, 2011] and it has to be considered in discourse analysis.
For the intention of archiving, the world wide web opens up many problems. An apt description is found in "Reproducible Web Corpora: Interactive Archiving with Automatic Quality Assessment" [Kiesel et al., 2018]. It states: "Meanwhile the web has evolved into a maelstrom of information, constantly published, shared, revised, and eventually lost. Despite the ease of copying and preserving digital information without loss of quality, digital media have proven the most volatile of all.[...], the web must be considered ephemeral". Not just the structure and accessibility of the web are highly ephemeral and complicated.
Collecting web content for discourse analysis has to be centered on the user. Since websites come at a cost for the host (energy, human resources), there are economic barriers set up to keep the pages free. Therefore, in most cases, the users need to pay with their data and time to access the content [Brynjolf-

sson and Oh, 2012]. This builds a remarkable relationship between the user and content provider on the internet. To access information for further online discourse analysis we found it mandatory to simulate the user (section 2.3). A user simulation tries to imitate the behavior of real humans to gather information, without spying on real people's personal data. We also want to have a look at what people actually see and reach on the internet and not just what is generally available to them.
The collective knowledge on the internet also needs to be an open source for a variety of scientists to be analyzed. It is needed that such a tool can be easily understood, adjusted, and executed. The user simulation that is used in this paper is designed minimalistic, to be easily understandable by the controller[1] of the software. Furthermore, since this can serve as a starting point for higher-level user simulations, the software needs to be modular and expandable. If the user simulation is intended to be flexible, easy-to-control, and -understand, the underlying archiver has to fit these needs as well.

[1]Controller in this paper means the person who manually creates a user model and executes it. So there is a distinct difference between the internet user (named just user) and the user of the software (named controller).

Chapter 2

Related Work

2.1 Online Discourses

First of the term "online discourse" needs to be defined. Since social discourse gets increasingly mediatized and interconnected through the ease of the World Wide Web more participants than ever can influence and take part in social discourse. Therefore, there is a need to view "online discourse analysis" not as a reduction of discourses analysis, but more like a shifted focus towards analysing how the connected media influences our perception of the discourse [Pentzold, 2014, p. 8]. So this work centers around capturing web content for online discourse analysis. With that in mind, the question about the collected content should not only refer to what the content is but also to how is the content is represented or therefore communicated (its mediation) and moreover where the content leads to (its interconnection).
In the book "Online Discourse. Theories and Methods of Transmedia Discourse Analysis" [Pentzold, 2014] we can also find a description of use-cases and platforms that appear in the context of online discourses. Those are especially helpful when classifying services and web applications we use.

2.1.1 Use-cases

Schmidt J. states three kinds of use cases [Pentzold, 2014, pp. 38-40]:

1. *Identity management* focuses on sharing information about yourself in any way or form. Not only your real personal data (e.g. location, date of birth, relationship), ideology (e.g. opinions, political ideas), or religion are shared, but also information about how people want to be viewed is provided.

2. *Relationship management* focus on connecting to new people with similar

ideas, making- or keeping up with friends, and interacting with other people in chats, forums, or other. Note that this doesn't just relate to personal use but also in the field of businesses (e.g. job applications, customer accumulation).

3. *Information management* is a broad use case that includes all actions on information. From creating (e.g. blogger), sharing (e.g. retweeting), filtering (e.g. search engines), selecting, collecting (e.g. RSS Feeds, journals) to working on information alone or with friends.

On different websites, we find the fulfillment of one or usually more of those use cases. But critical thoughts on those websites also need to include the intended use of those and more important the typical use of those websites. An example of this could be Wikipedia. Even though Wikipedia got a forum where contributors can connect and meet each other and fulfill relationship management. The intended-use and the typical-use is the encyclopedia and therefore the websites focus to fulfill the information management. For most websites, we can't evaluate the use cases in a simple way like this. The behavior varies from person to person and also the typical use can be really vague and develop over time including socialization aspects [Pentzold, 2014, p. 39]. So at the start, a person can visit YouTube to collect information and educate, but later after the website offers content to participate in, the user may take part in conversations in the comments. For this work, a user simulation should be able to include different ways to represent the same website or platform since different users might act on it differently. Also, a user simulation has to be adjustable since the platforms and the participating user-groups change over time and the software should suit that need.

2.1.2 Arenas of publicity

Meanwhile, for this work, I consider it a necessity to view our public appearance as plural publicities rather than one normative publicity. In reference to März A. [März, 2008] and Habermas J. [Habermas, 2018] a normative publicity is not sufficient to analyze our social, political, and personal thinking. It is better to differentiate between different public spheres, which can form, change and overlap even across different media. Good examples are hashtags that are not bound to specific websites or platforms and can be used on every common social media and then form a public sphere around topics (e.g. *"#blacklivesmatter"* or *"#MeToo"*). Also, a website, thread (e.g. reddit), or even single users (e.g. influencer communities) can form public spheres if their uses are very specific. To evaluate an online discourse with the help of a user simulation, the public

sphere the user participates in needs to be considered. Schmidt J categorizes four different arenas of publicity [Pentzold, 2014, pp. 40-43]:

1. *Arena of mass media publicity* expresses in a wide variety of journalistic work. Characteristic properties of this kind of publicity are higher barriers for publication, high frequency of publication and there is a distinct difference between publisher and receiver.

2. *Arena of expert publicity* is the digital continuation of scientific or professional journals. The barrier of entry is even higher than in mass media. The published content is subject-specific and is focused on creating or validating knowledge.

3. *Arena of collaborative publicity* is only possible through digital achievements within the internet. The prominent example is Wikipedia where people collaborate to collect knowledge and make it accessible to as many users as possible.

4. *Arena of personal publicity* is, like the above, only possible through the lowered entry barrier for participation on the internet. This publicity is quite complex in a way that every person can create communities as long as they find enough followers that they appear authentic to. They can publish on common social media like Facebook, Twitter, or personal publishing in blogs and many more. Even though the communication looks one-sided, there is a remarkable connection between the audience and the publisher. First, they share similar interests or existing relationships and secondly the publisher builds identity through the response he gets from the audience, which makes them interconnected.

As stated, users of the internet can take part in different types of publicity which change how much influence on the online discourse they have and how the online discourses are accessible or presented. The public spheres also can be interconnected and the online discourses can change at a different pace. In mass media, it can be a sufficient view of the online discourse to look up recent articles once every morning since they get published frequently and do not change significantly later. On the other hand, a polarizing video in an arena of personal publicity can change the perception of the online discourse as the conversation in the comments develops over time.

2.2 Web-Archiver

Most prominently and famous is the "Internet Archive" project created 1996. It started with the Wayback Machine [way, 10.06.2022] to collect website data

and to go back to older versions of websites. Over time the archive was expanded with digital books, text, and various kinds of media and got one of the biggest digital libraries in the world.
This way of capturing websites can cover a lot of content on the web and is especially helpful for preserving the look and feel of websites over time. This method of archiving is suitable for static websites but has problems with dynamic generated content. For archiving webpages with the online discourse in mind, dynamic or user-generated content is especially important to be collected.

2.3 User simulation in the web

Many modern web applications are highly complex and the screenshots or HTML documents representing them are just a fraction of the actual available content. The information that is presented to us is highly dependent on who we are as a user. When collecting data from the web for discourse analysis, the point of view needs to be individually shifted towards the view of the actual user. A similar approach was chosen when trying to archive virtual worlds in "Documenting a Virtual World - A Case Study in Preserving Scenes from Second Life" [Antonescu et al., 2009]. "We propose a solution which, in the particular case of Second Life, allows us to gather information the same way regular users perceive the world around them. We create and manipulate an object [...] to have it move through the virtual world and detect points of interest based on user activity and we perform a visual recording of such points of interest." Even though we are not in a virtual world the comparison is appropriate since the virtual world is the web application and the moving object is our automated browser. Analogical to the tracing used in the Webis-Scriptor, the paper records the content in a visual way following the user through his / her / its perception of the world. In the section User Simulation (section 3.2) there is more information on how the collected data changes when viewing it from the perspective of a web user.
Outside of the scientific environment web user simulation is mainly used for testing web pages and web apps. Therefore, browser automation frameworks are available for a wide variety of programming languages but differ in features and complexity. For our research, I found no examples of automated users for any of them. Typically, the tests are run by the developers of the website, so they focus on creating specific tests for the features of the web apps. In our research, the approach needed to be more general.
Our simulation was built upon the work of the paper "Reproducible Web Corpora: Interactive Archiving with Automatic Quality Assessment" [Kiesel

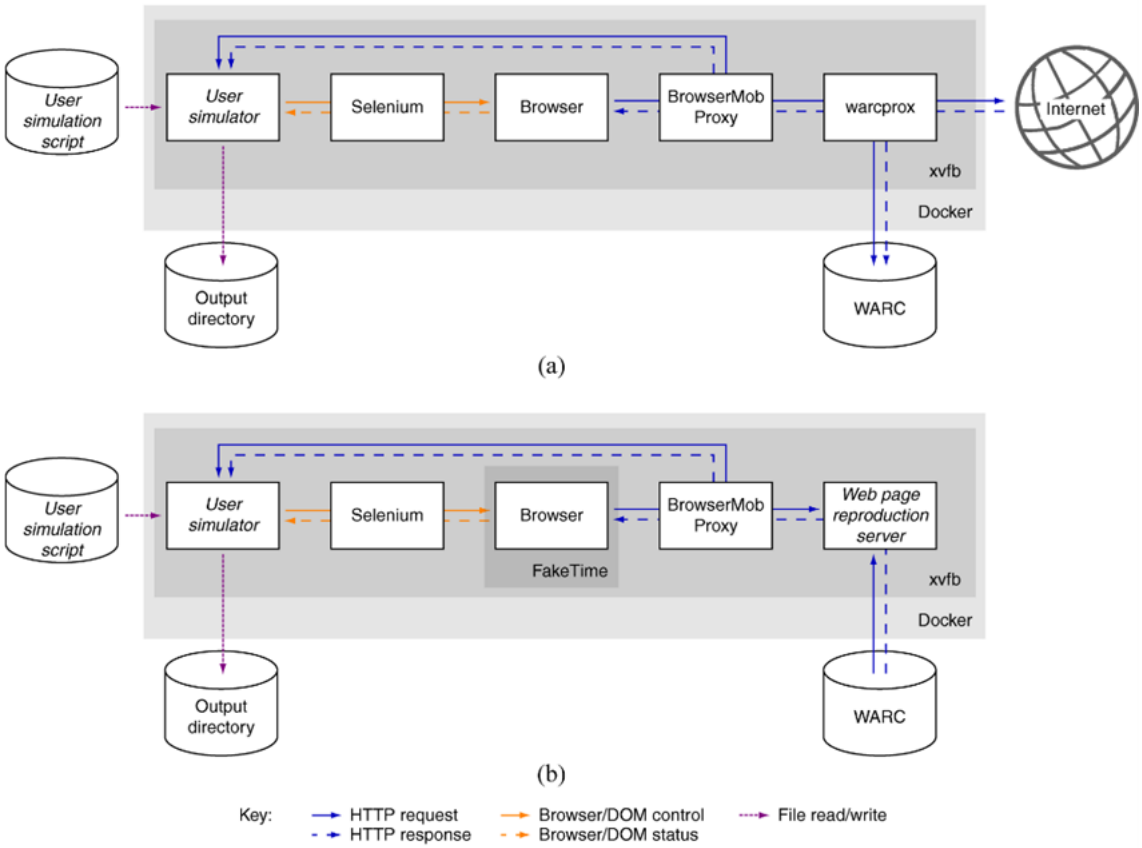


Figure 2.1: *"Software architecture of the Webis Web Archiver. Depicted are the building blocks, dependent libraries, and communications between them (a) for archiving and (b) for reproduction. Libraries are shown as rectangles, files and directories as cylinders. The components originally created for the Webis Web Archiver are highlighted in italics."* [Kiesel et al., 2018, p. 6]

et al., 2018, p. 8]. The paper uses an archiver that will be updated to fit the needs of the simulation. As described in the paper, the user simulation shipped with a "scroll down script" that opens a side and scrolls down the page. For more interactions, the controller needs to write additional scripts to operate the test framework Selenium in JAVA. In the following sections, the changes made to the archiver are described.

2.4 Webis Web Archiver

In the initial state of the Webis-Web Archiver shown in Figure 2.1 the controller writes the user simulation script where the actions the user has to take

on the internet are described, e.g. opening webpages, scrolling, clicking and other browser interactions. The user simulator manages the start of the script and operates the selenium framework in the desired way. With the help of the browser mob proxy, the user simulator collects the required data, e.g. screenshots, and HTML source code. The browser is the browser driver instance from selenium. The support covers common browser engines like Safari (by Apple), Edge (by Microsoft), Firefox (by Mozilla), and Chrome (by Google). The browser mob proxy collects content, performance, and protocol data (HTTP) to extend the functionality of selenium. The Web Archive Proxy (Warcprox) connects to the internet and collects all the archive data to reproduce the connection later.

Chapter 3

Approach

3.1 Webis Scriptor

To enhance the functionality for user simulation, significant changes were made to almost all parts of the Webis Web Archiver by the developer of the Web-Archiver Johannes Kiesel. The changes were decided in collaboration with me to ensure their compatibility with the goals of this thesis. To separate development, the name was changed to Webis Scriptor and moved to a new GitHub-Repository[1].
In Figure 3.1 the structure of the Webis Scriptor is shown. The archiving part of the Web Archiver didn't change significantly in the new Scriptor, since it worked in the desired way in the old Archiver already. The python wayback toolkit still records the browser interactions and creates web archives in WARC format. The first and most important change was done to the browser automation framework (3.1.1) which is the foundation of the software.
The Webis Scriptor changed the interaction with the user simulation script a few times during the development. The implementation of the script is described in the section user simulation script (3.1.2). In the final iteration, the Scriptor, settled on the idea of a rather complex output structure that is described in subsection 3.1.3. Furthermore, the tracing of the simulated user is another topic that is of high importance for the evaluation in the end of the paper. The collection of this data is addressed in subsection 3.1.4.
To start the Webis Scriptor there are different parameters that can be specified. For the user simulation to work we need the following parameters:

[1] https://github.com/webis-de/scriptor

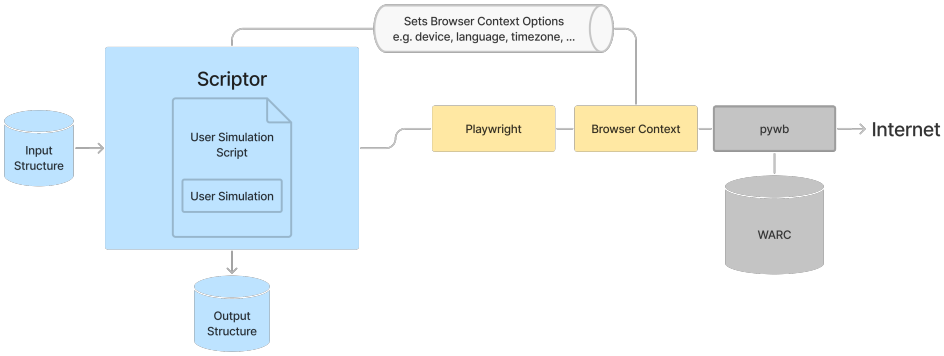


Figure 3.1: *Software Architecture of the Webis-Scriptor including the user simulation. Depicted are the building blocks, dependent libraries, and connections between them. Libraries are shown as rectangles, files as files and directories as cylinders. The components that were changed from Webis Web Archiver are highlighted in yellow. The rewritten and the newly created components are highlighted in blue.*

- *-inputDirectory*, *-outputDirectory*, *-scriptDirectory* sets the path to the respective directory.
- *-chain* will loop the execution forever. Also, the old output gets the new input.
- *-showBrowser* sets the playwright option *headless* = *false*. The browser gets a graphical interface that is shown to the controller. It's a needed option if a manual interaction with the browser is desired.

There are more parameters, which are not used for simulating users and therefore not explained here, but can be found on the GitHub Page. Also, all the parameters got default values that are stated in the Scriptor documentation. In the end, everything was packed into a docker image for easy execution, accessibility, and monitoring for the Kubernetes engine it will later run in.

3.1.1 Browser automation framework

The automation frameworks that are most widely spread are Selenium (by ThoughtWorks), Puppeteer (by Google), and Playwright (by Microsoft). Selenium was released in 2004 and is the oldest of those and most complex. Puppeteer is a lightweight alternative for Selenium with crucial feature improvements. It is written in JavaScript in NodeJS and can easily be parallelized. The big improvement for this project is the tracking of the network

state. The browser can wait for the network-load to be in "idle" and act after the page and all its dynamic content is loaded completely. For Selenium, the browser mob proxy was needed to add this functionality. Also, there is an easy way to automatically capture browser flows for users. The sessions can be captured through a Google Chrome extension and the created script can be re-run later and exported as a script. The biggest downside of puppeteer, at the time of creation of the Webis Scriptor, is the limitation to only automate the Google Chrome browser.
In the end, the browser automation framework chosen for the project is playwright (by Microsoft). Both frameworks share the same concepts. Since the main developers of the project are the same for first puppeteer and later playwright, it supports all the above features in a similar way. In playwright, the emulation of different browsers is possible and the developers enhance the framework with a useful concept that is helpful for the project. Playwright is capable of running multiple instances of browser contexts with different configurations (browser context options) in one browser instance. That enhances performance, especially when thinking about running the simulation for a longer period and simulating multiple users.

3.1.2 User simulation script

Running a script in the Webis Scriptor is as straightforward as it gets. The Scriptor reads a *Script.js* file that is located in the script directory specified at the start of the Scriptor. Since JavaScript doesn't natively supports types, it was helpful to use TypeScript as a superset of JavaScript to provide context for the classes and methods we are dealing with.
The Scriptor provides an abstract class called *AbstractScriptorScript* that can be inherited from to extend the *Script* class containing the instructions for the browser context. The script instantiate with the *constructor()* method by setting a name and a version number for the script. After this step the *run()* method is called to execute the user simulation (Figure 3.2).
In the old framework, the user simulation script got an open browser window to work with, so different properties (e.g. cookies, view-port, or language) of the browser were set by default. To open a browser context in playwright the framework needs "browser context options". In the browser context all those properties are set. Unlike with the Webis Web Archiver, in the Webis Scriptor, the properties should be adjusted by the user simulation script. So the script fits the identity of the user by setting language or saving login data (via cookies), which are kept over different executions.
As described – In order to get the browser context instance, the Webis Scriptor needs the browser context options from the script and in order to create the

```
module.exports = class Script extends
    AbstractScriptorScript {

    constructor() {
        super("UserSimulation", "1.0.0");
    }

    async run(
        browserContexts: any,
        scriptDirectory: string,
        inputDirectory: string,
        outputDirectory: string) {
        // instructions for the browser context
    }

}
```

Figure 3.2: *The script class requested by the Webis Scriptor.*

browser context options, the script needs a browser instance to collect the cookies from the websites.
To solve this dilemma a few iterations of development were needed to finally settle on a suitable solution. In a previous version of the Webis Scriptor, the script also had a *getBrowserContextOptions()* method that was called by the Scriptor before the *run()* method is called, but there was no way of changing it according to the users' needs during the execution of the script. In the beginning of development, the goal was to run multiple users in parallel in one execution of the Scriptor. Therefore setting up the browser context options

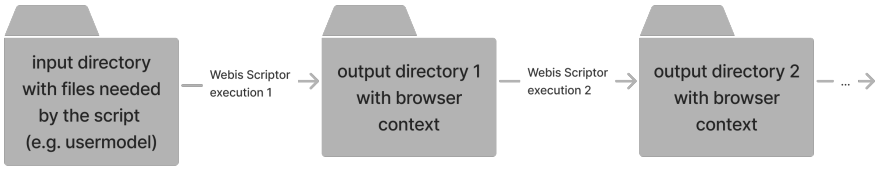


Figure 3.3: *Chaining the execution of the Webis Scriptor.*

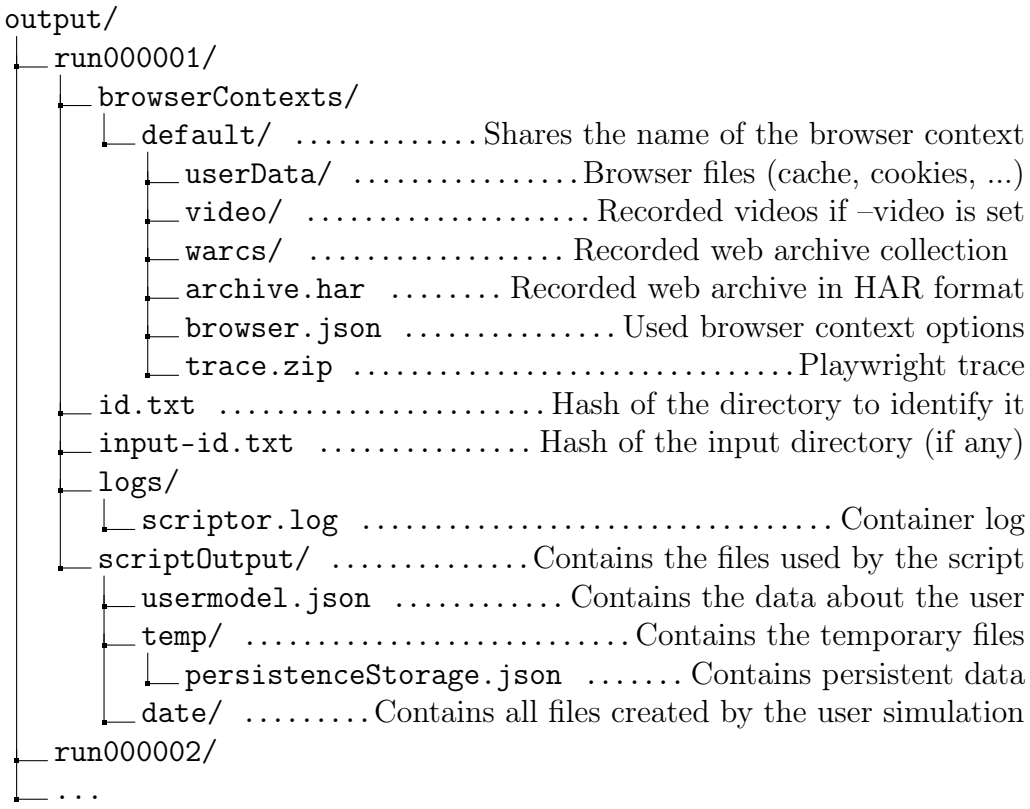


Figure 3.4: *Output Structure of the Webis Scriptor when running it with the –chain argument. Adapted from Webis-Scriptor Repository.* [GitHub - Webis Scriptor, 17.05.2022]

once was not sufficient. To solve the problem, after a few iterations the Scriptor settled for a chain execution design shown in Figure 3.3. By passing the *--chain* parameter to the execution of the Scriptor the old output directory gets the input directory of the next execution. If the script wants to change the browser context options, it saves the new options to the output directory and finishes the execution. After the restart, the Scriptor reads the new options accordingly.

3.1.3 Output-Structure

As shown in Figure 3.3 the output directory has to have the same structure as the input directory for the next execution. Therefore, a detailed listing of the content is shown in Figure 3.4. The root level of the structure is just named "output" but can be any folder as long as the *-outputDirectory* parameter is adjusted accordingly at the start of the Scriptor. In the next level, the different

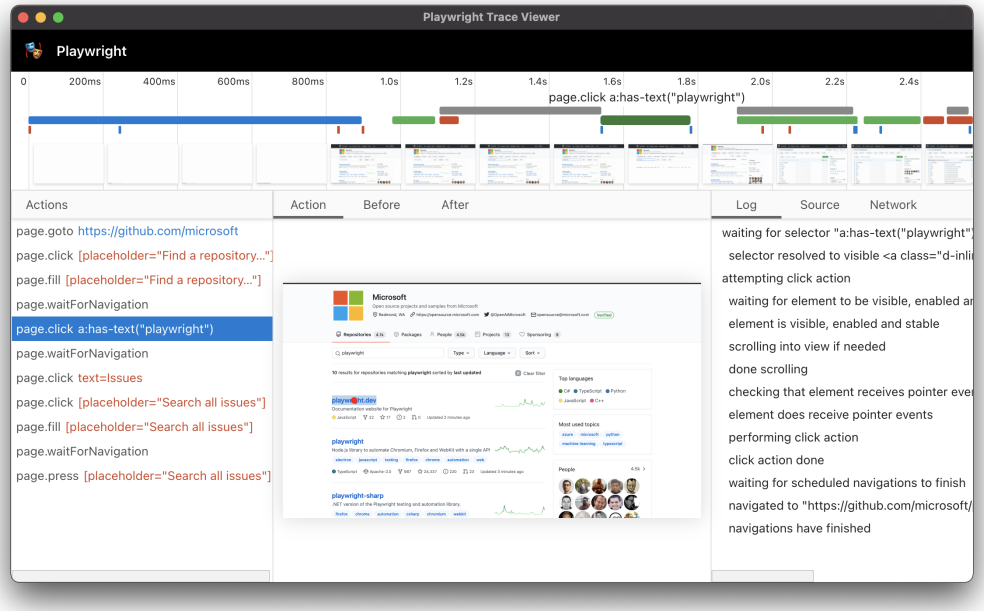


Figure 3.5: *"Playwright Trace Viewer is a GUI tool that helps to explore recorded Playwright traces after the script ran. Open traces locally or in your browser on trace.playwright.dev." [tra, 30.05.2022]*

runs of the chain are located when the *-chain* option was set at the beginning. So the *run000001* directory contains the output of the first execution and will be the input for the next execution, which then writes the *run000002* directory and so on. In the next level there are *logs* and files (*id.txt*, *input-id.txt*) related to the container to manage execution.
The *scriptOutput* directory is created by the script and will therefore be different depending on the given script. In this paper, the user simulation needs the *usermodel.json* to know what the controller wants to be executed, so it is read from the input directory and transferred to the output directory for the next execution. In the *temp* directory the script can save temporary data if needed, but this directory was not used for the final implementation of the user simulation. The only file that is created is a *persistenceStorage.json* that serves as a data store for simple key-value pairs over different executions. In the user simulation, it tracks the name of the executed user and the count of executions already done with the script.
Before chaining the execution of the Scriptor, the execution was done once for a user and ran for multiple days on its own, so the output was written under different directories named after the date they were created at. Even though

that's not needed anymore the directory still is named in that manner (e.g. *2022-02-22*). The content used to include the tracing data, and additional data generated while simulating the user (e.g. downloads, screenshots) that was created by the script. In the final implementation, the tracing is done by the Scriptor as described in the next section (3.1.4) but the additional data is still saved here.
The *browserContexts* directory is fully managed by the Webis Scriptor and contains mostly self-explanatory files. A web archive collection (*warcs*) and a HTTP-Archive (*archive.har*) [W3C, 10.05.2022] are created by the pywb toolkit[2]. Those were not used in the evaluation of this paper, so are not further discussed. For this user simulation the *-video* option was never used since it is highly memory demanding and the trace option already is sufficient, but when present, the generated videos will be saved in the *video* directory.
The *userData* in conjunction with the *browser.json* can fully reproduce the browser context from the last execution. In the next section, a detailed explanation is provided of how the *trace.zip* comes together.

3.1.4 Tracing

When monitoring an automated browser the output needs to be comprehensive to understand the processes that went on in the background. In the specific case of user simulation, it is desired to have a good overview of what the user did in the controller's absence. Therefore, a deciding factor for the framework was the capability of tracing down the browser easily. When the browser context is established, the tracing process can be started by calling ...

```
await context.tracing.start(
    { screenshots: true, snapshots: true }
);
```

... , is ended with ...

```
await context.tracing.stop({ path: 'trace.zip' });
```

and captures everything in between. The resulting archive (*trace.zip*) is found in the output.
Since there are too many data features that are captured in the metadata that are no concern for this project, the focus is on the features that are most important and used in the user simulation. An overview is shown in Figure 3.5. The top structure is a timeline where every captured event is laid out. Events can happen within playwright for different reasons.

[2]https://github.com/webrecorder/pywb (31.05.2022)

The network connection can broadcast events if the browser starts navigating, finishes navigating, or is going into another network state (e.g. "load" or "idle"). In this paper, this feature is used to track when the user actually sees content, all dynamic content is placed and can further interact with it.
Furthermore the trace receives events on every action the script commands, like navigation, clicking, or scrolling. Those actions are documented, so the controller does not need to have any understanding of the user simulation script to understand what is going on, since all taken steps are shown separately on the left side of the trace viewer. Combining these features the *trace.zip* also gets very memory sparing, since it just captures data like frames or HTML documents when the website changes. Also, every data entry is ordered accordingly and opens up easy evaluation possibilities later in the project.

3.2 User-Simulation

At first, I want to show how the perception of the content in the internet changes with our usage.
The Internet can be viewed as an ever-growing system. That counts complexity-wise and for content size. Not only dedicated content providers like Netflix or News Websites keep expanding the available content but also the amount of user-generated content has "exploded" over the last years and will not get less in the near future:

> *"90% of the data on the internet has been created since 2016, according to an IBM Marketing Cloud study. People, businesses, and devices have all become data factories that are pumping out incredible amounts of information to the web each day." [Schultz, 06.08.2019]*

Even if that's an overestimation since raw memory demand of the internet is no suitable metric to evaluate the available amount of content[3] the argument holds.
Also, the complexity of the internet raises in different ways. Stated here are just a few of them.

Unstructured data

> *"Research from IDC shows that 80% of worldwide data produced will be unstructured" [Schultz, 06.08.2019]*

[3] e.g. A higher resolution image doesn't add more content to the internet but still raises the memory demand.

Dealing with unstructured data makes it naturally harder to gather impotent information from the data.

Access limitations

To access user generated content in social media an own account could be needed like on Facebook or Instagram. Websites can block content in case you don't accept the tracking with cookies (e.g. news website "bild.de"), if an ad-blocker is installed and activated (e.g. also "bild.de") or you didn't paid for the content[4]. The reasons being economical, "privacy" concerns, or else is not relevant for this paper.

Dynamic content

Modern websites are built on various web-, database- or recommendation systems, to provide the user with suitable information, advertisements, or else. These can factor in the user's browser history, his recent clicks, items in his shopping cart, and every imaginable more.

Short lived content

Even though there is a saying "What happens on the internet, stays on the internet" and that's true to most extent. There is no denial about some content is better visible to users than others. Search engines are designed to highlight just specific parts of the web that are considered impotent for the audience. The same goes for news articles that can be outdated fast and are not shown in the news feed of the users anymore.

Visibility

The last very impotent factor when collecting data for evaluation is the visibility of content. On modern websites often content just gets visible after interacting with the website (e.g. scrolling). New content then gets added in the background without the user noticing. But a more important aspect of visibility in the context of online discourses is the layout of content. A few questions that come to mind are: Has the content enough weight to be recognized by a user (e.g. text size, text weight)? Is it in the center of the website or out of his field of view? Is it on the main page or does the user reach it with multiple interactions? The answers change the actual information the user receives.

[4]Almost every news website got separated premium content hidden behind a "pay-wall" (including bild.de, sueddeutsche.de or welt.de)

In section 2.3, in reference to a virtual world [Antonescu et al., 2009], is already stated that the perspective of an actual user can be useful when immersing in a complex world like the internet to gather data.

In summary, it has shown, that the perception the user has of the internet, changes regarding his browser-history, having or not having an account, an ad-blocker, a paid/unpaid subscription, his field of view, and much more. And the subjective view of different online discourses changes with his/her/its perception.
In the first subsection 3.2.1 the approach of abstracting the user is described.

3.2.1 User Model

The topic of user simulation turns up multiple questions that need to be answered before approaching an actual implementation. By definition, simulations are trying to recreate something from reality by approaching it as far as possible. In this paper, "something" means a human that is browsing the internet day in and day out. To approach this user a few abstractions were made. The representation of that user needed to fit in a simple file, so creating, understanding, or editing those users require no additional programming skills. To simplify the user simulation to even greater extent, this usermodel marks the only entry point for the software. The obvious choice for the format was JavaScript Object Notation (json). It consists of easy to understand key-value-pairs and lists and can be easily funneled in JavaScript.

Name

At first, a fundament for the user was needed to describe some characteristics and identify the user even when different controllers are involved. In Figure 3.6 there is an excerpt from the usermodel for "Arif Sykes"[5], a fictional user that was manually created in form of a *usermodel.json*. The first key is the name key. Its main purpose is for identification. The controller can choose any name or ID number and the user simulation script also identifies the user with this name across the execution. In this paper, all the names were generated on "name-generator.org" for a better recognizability and are kept consistent for their accounts' username, out- and input filenames, or generated directory-names.

[5]All five usermodels are available in the git repository: https://git.webis.de/code-teaching/theses/thesis-knappik/-/tree/master/inputUsermodels

```
{
 "name": "Arif Sykes",
 "interests": [
  "theatre", "culture", "society"
 ],
 "influencedBy": [
  "YouTube", "Instagram", "Der Spiegel"
 ]
}
```

Figure 3.6: *Usermodel-entries regarding the user's character or identification (Excerpt)*

Interests

The usermodel was created with the concept of a high- and low-level abstraction in mind. To give the controller easy-to-understand characteristics of the user an *interests* key was added. In an unstructured list of strings, any number of interests can be added to describe the user, just the first value gets highlighted as the most impotent interest in the *toString()* method. These values are not directly used in the simulation, but their main purpose is to preserve the character of the user over time. It is the highest abstraction layer of the user in the model. Even if the media we consume or our schedule of internet-use changes, the users' interests likely won't. This property will come into play in the last paragraph of this section. It is also worth mentioning that this property helps when multiple controllers are collaborating. From this layer, every controller can start to understand the following lower abstraction levels of the user better.

Influenced by

As described in subsection 2.1.2 there are different arenas of publicity that we can take part in. Under the *influencedBy* key in the usermodel, entries can be listed which describe those different arenas we participate in. Just like the *interests* those values don't directly affect the simulation but rather add another layer of abstraction. The name was changed since "arenas of publicity" is not self-explanatory and comprehensibility is still a goal.

The next excerpt in Figure 3.7 shows the options the controller can set to describe the user to the browser. In real sessions, those are obvious through settings or tracking. Those also include some of the browser context options

```
{
 "device": "iPhone 11",
 "locale": "de-DE",
 "timezoneId": "Europe/Berlin",
 "geolocation": { longitude: 48.858455, latitude:
    2.294474 }
 //geolocation grants permission to access it to the
    browser
}
```

Figure 3.7: *Usermodel-entries regarding the user data visible to the browser(Excerpt)*

of playwright. As described in subsection 3.1.2 early stages of development referred to default values for the device or the language, but even if some take no part in the evaluation of this paper, it is interesting to watch how the perception of the online discourse changes in different regions, languages or devices. So those options were added and change the output of the simulation drastically.

Device

The used device is structured data read from a *deviceDescriptor.json* file by playwright[6]. This list is unique to playwright and was, at the time of creation, not provided in the other frameworks. It sets a combination of values including:

- *userAgent* provides information about the client
- *viewport* sets the height and width of the device
- *deviceScaleFactor* can improve visibility
- *isMobile* requests the mobile version of websites
- *hasTouch* enables touch gestures for the page
- *defaultBrowserType* sets the browser engine, at the time of creation supports webkit, chromium, firefox

Having all these values at hand by only setting *"device":"iPhone 11"* greatly increases simplicity and readability for the controller. Also, we get a clear

[6]https://github.com/microsoft/playwright/blob/main/packages/playwright-core/src/server/deviceDescriptorsSource.json (10.06.2022)

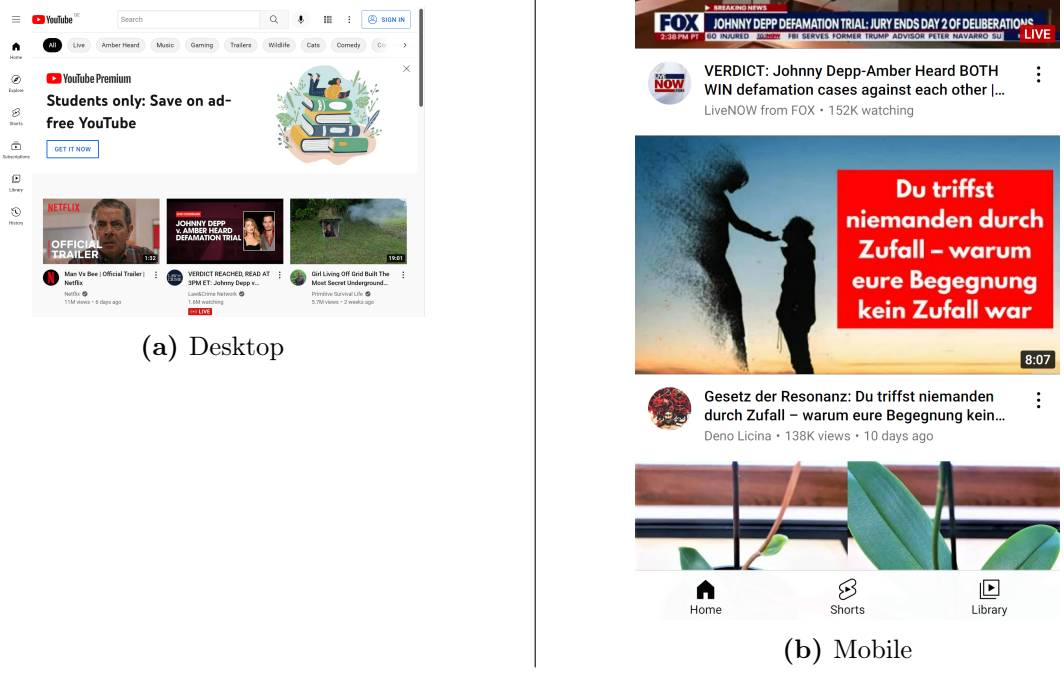
vision of what to expect in the output. There are images provided in Figure 3.8 to confirm our expectations.

Local, Timezone and Geolocation

The next settings are mostly self-explanatory. The important one being the *local*. It not only decides about the language of the content but changes with it. On modern news websites, the created content can exceed 100 articles per day [Meyer, 26.05.2016]. Just a fraction of them are translated to the desired language and if they are not related to your region or even *geolocation* they probably won't appear in the news feed. That can lead to big regional differences in online discourses. Again in the Figure 3.9 there is a comparison attached.
The *timezone* was added since it was little to no effort after implementing the other settings. In contrast, I found no evidence that the *timezone* field is used to change the displayed content. Not even the time from the requested timezone gets adapted. It seems like the field that is later passed to the header by the browser has no impact on the resulting websites, since they seem to calculate time from the *local* and *geolocation* property only.

Frequently visits

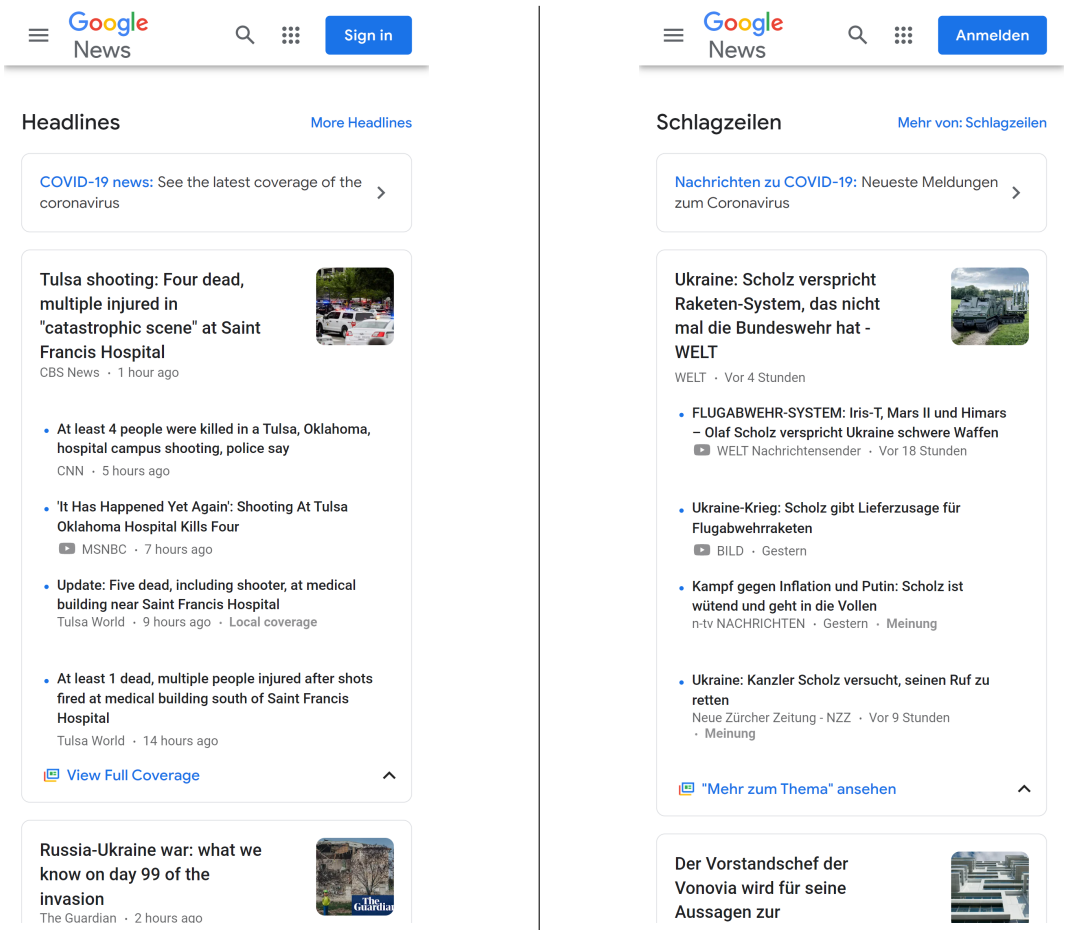
As the main source of interactions, a schedule for the user was created (Figure 3.10). The name changed a few times during development in favor of readability. The idea of interaction modules that can be added, edited, and expanded to the controller's needs was the main focus from the beginning. The name for this section was "interaction modules" accordingly, changed to just "modules" later, but still wasn't easy to understand either, so the name ended on *"frequentlyVisits"* in the end.
In the section 3.2.2 there is a detailed description to all of these interaction modules since they can act very different within the simulation, but in the model, they are kept similar. The controller adds a *type* and *executionTime* and in the user simulation they are then repeated every day. A weekly schedule would have been possible, but for a few reasons decided against. At first a weekly schedule of interaction modules would have been full of copy and paste modules when creating different users manually and therefore were even less readable. Second, the size of the schedule would have grown in size significantly and is not small to begin with. In appendix (A) there is a full usermodel already fitting 2 pages. At last, the complexity also grows with the schedule size, but in the end, an expansion to a full week, or monthly schedule could be big expansions to the usermodel. Our content consummation changes over



(a) Desktop

(b) Mobile

Figure 3.8: *youtube.com layout comparison between device setting* ***"Desktop Chrome HiDPI"*** *and* ***"Pixel 5"*** *(mobile)*



(a) Local: *"en-US"* **(b)** Local: *"de-DE"*

Figure 3.9: *news.google.com content comparison between local setting* ***"en-US"*** *and* ***"de-DE"****.*

```
{ "frequentlyVisits": [
  {
   "type": "Instagram",
   "executionTime": "20:30" },
  {
   "url": "https://www.stern.de/",
   "type": "OpenUrl",
   "executionTime": "21:00" },
  {
   "type": "YouTubeAbo",
   "executionTime": "21:15",
   "subscriptions": [
    {
     "name" : "KOOKIELIT",
     "representation" : "UCd7yIRGoYvi1DUKIGTYvwFg" },
    {
     "name" : "kpopfication",
     "representation" : "UC2507Z5YOsfud33imrtj8ug" }]
  }
 ]
}
```

Figure 3.10: *Usermodel schedule, listing the websites a user frequently visits (Excerpt)*

the week or month[7] so the time of surfing the internet can definitely make a difference in the content we perceive.
It is worth mentioning that these usermodels are created manually and didn't change during the execution in this paper, but given their simple *json* they could be generated or changed (even during execution) by scripts in the preferred programming language[8].
To have a quick introduction to the user and to condense it in human readable sentences the *usermodel* class can write a short text about the user in the *toString()* method (appendix B).

[7]For example on weekends users consume more content, due to having more free time, and media publishes more content accordingly

[8]Every modern programming language can process json files (e.g. python dict) or can access library's that add the functionality (e.g. Java Jackson library).

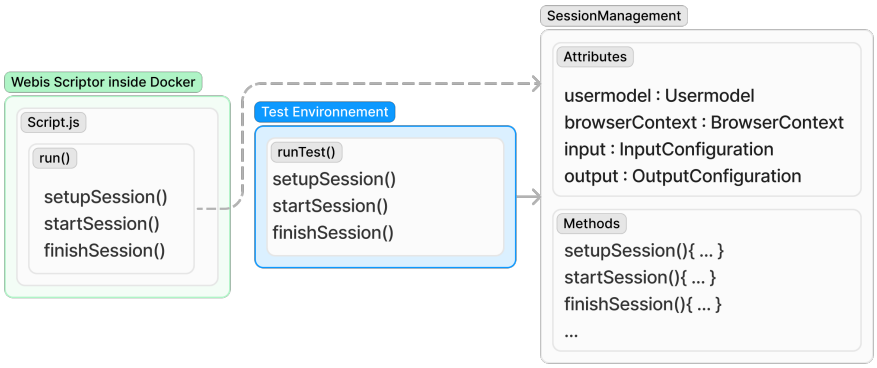


Figure 3.11: *The development environment for the user simulation script with the Session-Management separated. (sketch)*

3.2.2 Script

The script file is structured as shown in Figure 3.2. Additions to the *script* class were just made by adding tests for some modules or functions. Its main purpose for the development of the user simulation script was to be the interface with the Webis Scriptor. The major business logic was relocated to a *SessionManagement* class. In this way there was no interference with the Scriptor development happening in parallel and a separate testing environment was established to only test the functions of the user simulation part, seen in Figure 3.11.

The procedure of reading the usermodel and executing the interaction modules changed multiple times throughout the development, just one is mentioned. In an early version of development all the modules were looped like here:

```
// Pseudo code
Script.run(browserContext){
 while(true){
  browserContext.startTracing()
  let modules = usermodel.getModulesScheduledNow()
  for(module in modules){ module.run(browserContext)
     }
  browserContext.stopTracing()
  sleep(5min)
 }
```

```
}
```

The script didn't need to finish since the output is written during the execution. In case of an error or by canceling the process no data is lost. But to implement chaining executions (Figure 3.3) and the tracing outside of the script, the *run()* method needed to come to an end after at least one module was executed. Therefore extra classes for the input, the output, and the execution times needed to be written or changed.

In the final implementation the *run()* method looks like this:

```
// Pseudo code
Script.run( browserContext : BrowserContext,
       inputDirectory : string,
       outputDirectory : string){
  let sessionManagement = SessionManagement(
   browserContext,
   inputDirectory,
   outputDirectory
  )

  let setupIsDone : boolean = sessionManagement.
     setupSession()

  if(setupIsDone){
   sessionManagement.startSession()
  }

  sessionManagement.finishSession()
}
```

In the next section the most important attributes and methods from the *SessionManagement* are described.

Session Management

The term *"session"* in this paper refers to the execution of one or more interaction modules from the user. Analog it can be seen as "a user surfing the web at 9 o'clock on his phone". Another session starts when he/she/it reads news in the evening.
To schedule those sessions and track them across executions the *SessionManagement* was created. As shown in Figure 3.11 in the gray container, it relies mainly on 3 objects: An *inputConfiguration*/*outputConfiguration*, the *browser-*

Context and a *usermodel*.
The *inputConfiguration* is created with the path to the *inputDirectory*. It is read-only and checks if the browser context options are already set up and if user data is needed. The first 3 runs usually look like this:

```
1. Execution -> write initial browser context options
    (e.g. device, locale); restart;
2. Execution -> open the browserContext and collect
   the needed user data for given interaction modules
   (e.g. login, accept cookies); restart;
3. Execution -> start first session;
```

To write the needed data an *outputConfiguration* is needed. It extends the *InputConfiguration* class and therefore can be parsed from the *inputConfiguration*. Beside changing some paths and files, and being able to write, it serves the same functions to the *SessionManagement*.
The *browserContext* from playwright is needed to perform all the interactions that need to be traced (e.g. open pages, navigation, clicking on elements). It is passed as a parameter in the constructor of the session.
The last requirement is the usermodel. It is initialized with the *usermodel.json* but extends it with multiple functions for the session to work with.
With all these parts explained the methods of the Session-Management should be understandable. In the last Pseudo code example, the Session-Management is instantiated with given parameters. The *"setupIsDone"* variable turns **false** if the Scriptor is in first or second execution and with it saves browser context option (1. execution) or executes every interaction module that needs a setup (2. execution). If the setup for the modules is done the script starts the session by requiring the next interaction modules and the duration until they execute from the usermodel. To end the session it saves the usermodel for the next execution.

Interaction Modules

The smallest units of the user simulation are the interaction modules. The main idea of those is to interact with multiple websites or web apps according to the users' behavior. So for example on Instagram, the typical way to consume content is to scroll down the main page, on the other hand on YouTube the user may search for some channels and looks for recent videos. To separate these cases every interaction module can build its own behavior for desired websites. All interaction modules can be used multiple times in the schedule. An abstract description is shown in Figure 3.12. At the time of writing the user simulation provides 6 different types of interaction modules:

```
export abstract class InteractionModule {
  url: string;
  id: number;
  type : InteractionModuleType;
  executionTime: Time;
  subscriptions: Subscription[];
  needsSetup: boolean;

  abstract runModule( sessionManagement:
    SessionManagement): Promise<void>

  abstract runModuleSetup( sessionManagement:
    SessionManagement): Promise<void>

  toString() { ... }

  timeToExecution() : number { ... }
}
```

Figure 3.12: Excerpt of the interaction module class.

```
export enum InteractionModuleType{
  OpenUrl= "OpenUrl",
  YouTubeAbo = "YouTubeAbo",
  ManualUrl = "ManualUrl",
  InstagramModule = "Instagram",
  FacebookModule = "Facebook",
  GoogleNewsModule = "GoogleNewsQueries"
}
```

The first being the *"OpenUrl"* type, made to open any website, by adding it to the usermodel. After loading the page the script will also scroll down a few times. In the simulation of this paper this is used to open blogs, twitter hashtags or news websites as shown in Figure 3.10. The *"Instagram"* and *"Facebook"* modules work similar. In those modules, the account setup is needed and will run before the first session. In those instances the *runModulSetup()* method is called and requires you to fill in login data to the page and save the login in the browser. Also, the usage of cookies for the website needs to be accepted or declined. In the Instagram module, the page scrolls down the feed for a few seconds. In Facebook, the simulation is visiting a view destinations in the web app (e.g. home, news). The accounts need to have preparation

(e.g. following users, accepting friends requests).
For creating multiple users with various interests and influences a lot of accounts need to be created and filled with content. That refuses the possibility of generating these models automatically. A solution for collecting content without having to rely on accounts was the implementation of the *"YouTubeAbo"* and the *"GoogleNewsQueries"*. With a separate *subscriptions* key, channels or news queries can be added to those modules that are then iterated through in the simulation (Figure 3.10).
The last interaction module type is called *"ManualUrl"*. Its idea was to manually navigate the browser (with a given start URL) and capture real user interactions in the browser, which are saved as *trace.zip* and can serve as a comparison to the simulation or add interactions of the user that are out of the ordinary schedule. To not run it every day the execution time value is set to *"atStart"* resulting it to start only once, with the setup of all the other interaction modules.
The selection of modules was made with the coverage of the uses-cases of the internet in mind (subsection 2.1.1). Facebook covers identity management for a lot of users since it is the largest social media platform. Instagram tries to cover a lot of use-cases for Relationship management. Also a Telegram[9] module was created to simulate relationship management, but the setup needed multiple phone numbers and was unpractical to use. Also spying on real users' chats was not the intention of this paper, so public groups were left as the only purpose and the interaction module was deferred.
With RSS Feeds, articles from different news publishers can be distributed to various platforms (including GoogleNews). All though GoogleNews doesn't use RSS Feeds anymore but rather uses their own service, a huge amount of articles are collected there, providing a simple implementation to cover information management over mass media as well.
With only a limited amount of interaction modules, users can be built and simulated in docker or Kubernetes. To evaluate the process of running a simulation, in the next chapter users were created and simulated.

[9]Telegram offered the most stable web app at the time of writing. WhatsApp was just working with a Smartphone connected to the internet at all times and Signal is not as widely distributed.

Chapter 4

Evaluation

4.1 Created Users

In appendix B the 5 manually created usermodels are described. The interests were collected with no particular intentions, but distinct from another. The influences were assigned at random, but spread around the users, there are two features that make them valuable to serve as a test group for an assessment (Table 4.1). Those features should have an impact when visually evaluating the presence of online discourses in their simulation trace. Depending on the usermodels we can assume a few hypotheses :

1. The mobile website usage should limit the content that is visible to the user (simply because it is smaller) and thus should find less text in total.
2. German-speaking users should find more information about discourses limited to the german speaking regions.
3. Most of the users' interests don't overlap and therefore should participate in some distinct online discourses.

Feature	**Applies to**	**Doesn't apply to**
Mobile	Arif, Mandy, Suzannah	Kia, Menna
Language german	Arif, Kia, Suzannah	Mandy, Menna

Table 4.1: *Features shuffled into the usermodels. The corresponding values if the features doesn't apply are: desktop instead of mobile and language english instead of german.*

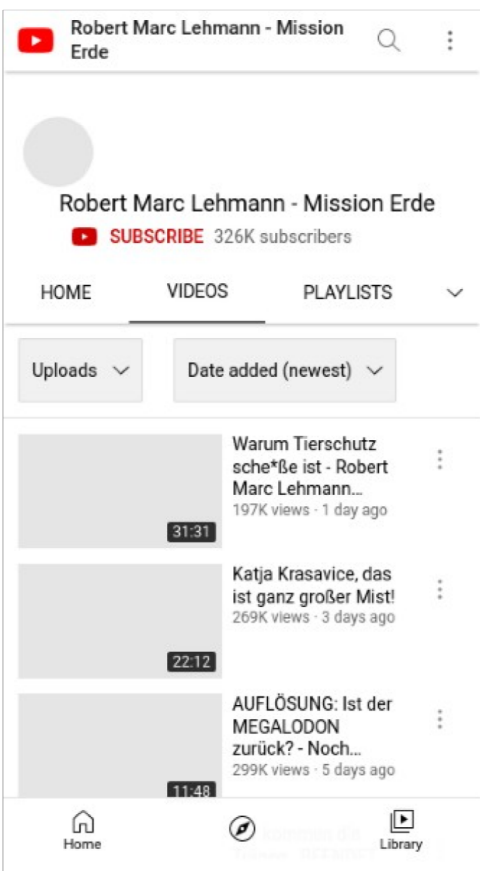
To evaluate these hypotheses, the collected traces need to be searchable. At first, a brief introduction of the contents of the *trace.zip* directory is provided, to understand the collected available data. It contains all the events in a text file (*trace.trace*) and all the network calls are saved in a separate *trace.network* file. All files are structured in json format. The searchable content is then located in a *resource* folder, containing the HTML files and all captured frames during execution.
As described in section 3.2 the goal is to search just through the data that is actually visible to the user, so the HTML-DOM that may contain metadata the user isn't aware of or parts of the webside that the user didn't reach when browsing, was no option. A better approach to search through the output of the simulation is to use optical character recognition (OCR), in form of the open source tesseract software.

4.2 Text Corpus

All the screenshots of the pages found in the trace archive are fed into tesseract with a python script. In Figure 4.1 an example for the OCR is shown. In the example shown, besides some spelling mistakes with german letters ("ö; ü; ß" get "o; ii; B"), the words mostly match in this case. Some devices produced worse OCR results, with more spelling errors, maybe due to smaller letters. Prepossessing the images was tested but since all the frames had different conditions, usually some images got recognized better and others worse[1]. In the end the default images where used instead.
To generate a corpus, the text from all frames was collected. Even if the trace function of playwright tries to guess when the content changed it still produced a lot of duplicate frames (e.g. they just differ in showing a progress bar). But after the OCR the frames that produced the same strings as the previous got removed and only one copy was kept, solving that problem.
Especially while scrolling, the frames still capture the same text multiple times, as seen in Figure 4.2. That behavior might look unpractical for evaluation but can be helpful in some cases[2] and therefore was left in there. It just needs to be kept in mind for advanced processing. Of course, every user gets his / her / its own text corpus with every session saved under the respective timestamp.

[1]For example binarization, sharpening or raising contrast improved the readability of black on white text but worsened or made impossible to read text on thumbnails or images, which was expected but tested anyway.

[2]If a text is present over multiple frames it is likely for it to be an important element of the website, like navigation or a sticky header.



(a) original frame

Robert Marc Lehmann - Mission Erde
(@ SUBSCRIBE 326K subscribers

HOME VIDEOS PLAYLISTS

Uploads ~ Date added (newest) v

Warum Tierschutz
sche*Be ist - Robert
Marc Lehmann,

197K views 1 day ago

Katja Krasavice, das
ist ganz groBer Mist!
269K views -3 days ago

AUFLOSUNG: Ist der
MEGALODON
zuriick? - Noch,

299K views °5 days ago

a @® a

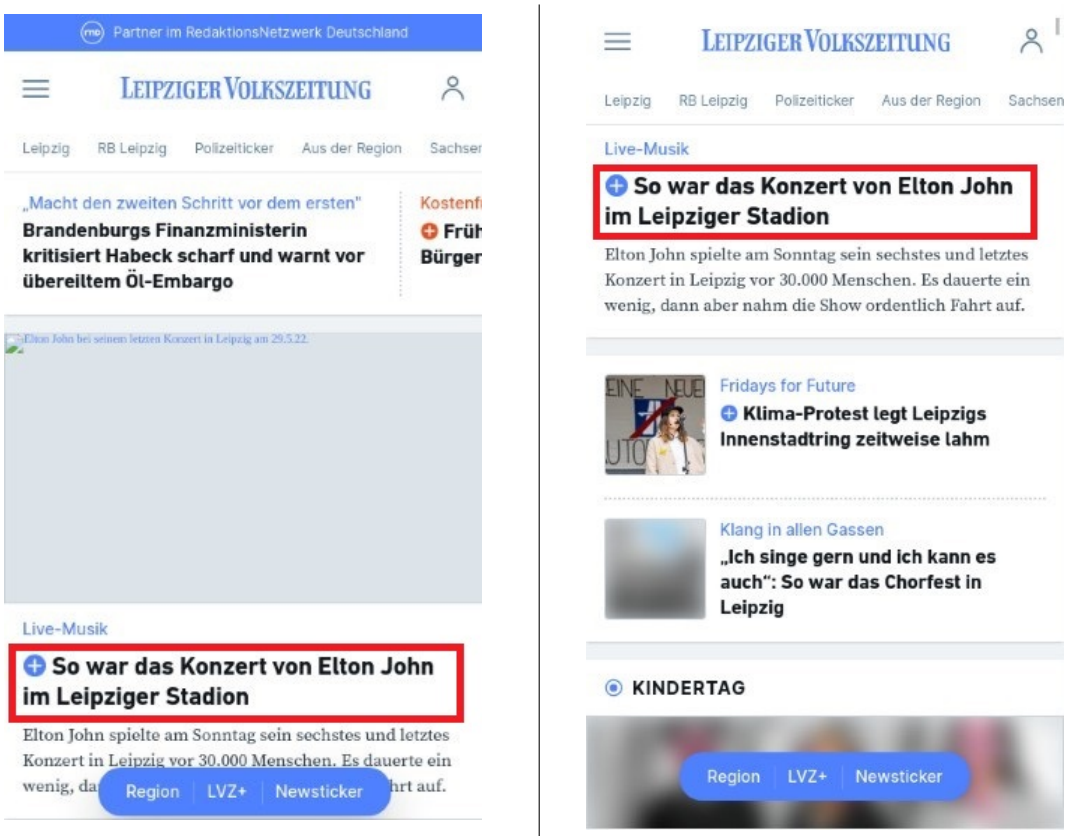
Home Library

(b) tesseract-ocr

Figure 4.1: *The comparison of the original frame captured during the user simulation (a) and the characters recognized by the tesseract-ocr (b).*

4.3 Searching for online discourses

To search the corpora for specific online discourses we assume that the online discourses have a collection of keywords that are bound to them. So for example in the discourse about the Ukraine conflict one of the keywords (*"ukraine"*, *"selenski"*, *"melnyk"*, ...) has to be present. So by counting every appearance of those keywords we get a glimpse of how much a user was aware of it.
The script for searching these keywords is using a sliding window through the character sequence. The window for searching is larger than the actual keyword (keyword-length + 8), so it can capture a prefix and a suffix of 4 characters. When sliding the window the middle part gets a string distance



(a) initial page

(b) page scrolled

Figure 4.2: *The comparison of the frame of the initial page (a) and after it was scrolled (b). Highlighted is a headline that appear on both frames and will have a dublicate in the corpus.*

ratio calculated with the *SequenceMatcher* of the *difflib* python library[3]. The whole window will be collected if it matches the keyword that is searched for. The ratio calculated by the *SequenceMatcher* needs to exceed the threshold of 0.7 to be counted. After all matching windows of the frames for the whole session are selected, the windows can be filtered for duplicates. If the corpus had 2 frames like in Figure 4.2 with a duplicate text. The search for keyword *"concert"* would find the windows:

*"das␣***Konzert***␣von"* - from screenshot (a)
*"das␣***Konzert***␣von"* - from screenshot (b)

[3]It is calculated with an extension of the "gestalt pattern matching" algorithm by Ratcliff and Obershelp [dif, 04.06.2022].

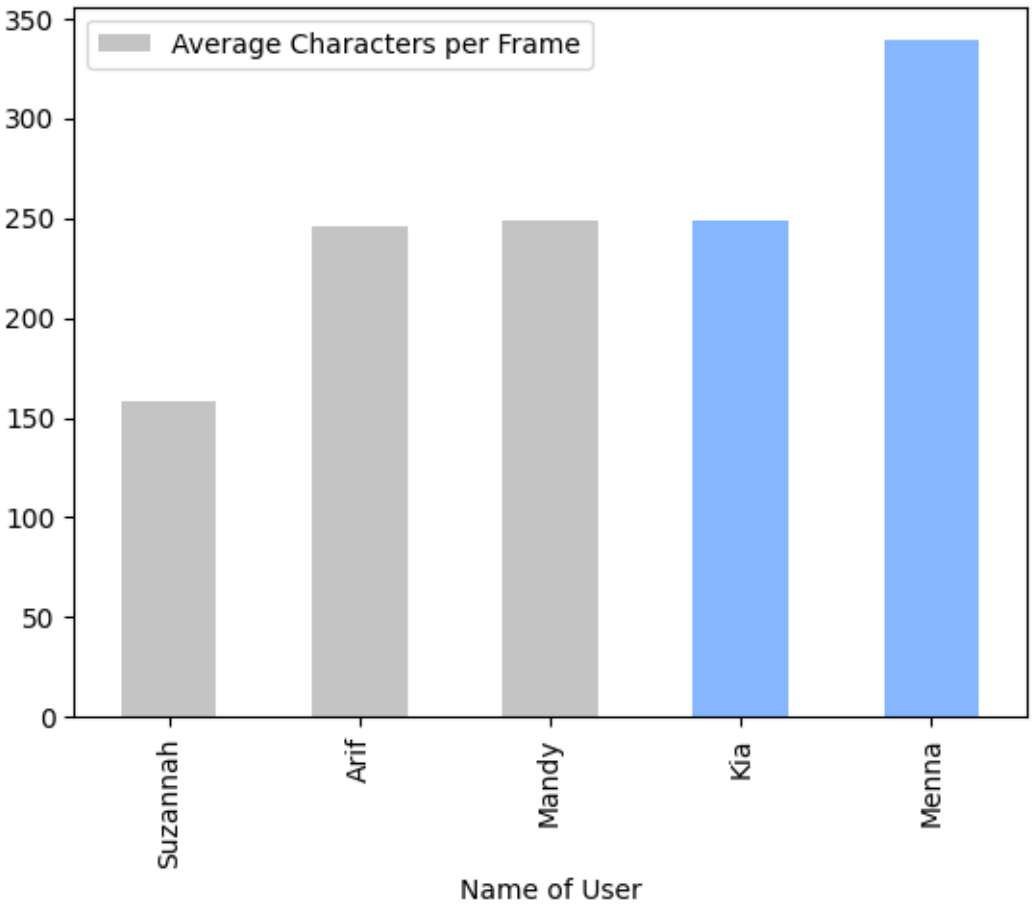


Figure 4.3: *Shown is the average character count per seen frame. Highlighted in blue are the desktop user.*

The pre- and suffix is written in italic. The matched string (ratio 0,714) is highlighted in bold.

Since the windows are the same we can assume they represent the same occurrence of the word and drop one of them. In the last section of this chapter, the hypotheses for online discourses in section 4.1 will be answered to evaluate the quality of the gathered information from the users.

4.4 Assessment

During the 11 days period (between 25.05.2022 and 05.06.2022) of running all 5 users in Kubernetes, the user simulation generated 162 traces. There was data lost for the users Kia and Mandy both users had interactions not finishing,

due to missing timeouts in the script and websites loading forever. The Webis Scriptor also had a problem reading the last output. Both problems were fixed but the data was lost for the time anyway. They finished only 13 and 6 sessions to evaluate. But since the evaluation of the user simulation will only be used for simple assumptions it won't matter as much.

4.4.1 Word Clouds

To get a quick understanding of what the corpus consists of, we created word clouds attached on the next pages (figure **??** - 4.4). The corpus needed to be preprocessed with the help of natural language processing (NLP). The open-source python library spaCy[4] was used to tokenize the character sequences in the corpus with german and english word lists. Per session, only one occurrence of a word is counted. The clouds of Arif and Suzannah seem to represent their influences and interests. *"Leipziger"* and *"spiegel"* clearly refer to the newspaper Arif frequently reads. In Suzannah's cloud *"championsleague"* refers to the champions league final that took place on 28.05.2022 and at the top right the formula 1 race in *"monaco"* is present. Right below *"monaco"* the word *"madrid"* (Real Madrid) can be found which was the winner of the champions league. For Kia very prominent examples are *"beauty"* (from the fashion magazines), "folge", and "newsletter" since she is asked to follow the newsletter often on the websites. Menna's and Mandy's cookie setting wasn't set up properly on Reddit and Twitter because when setting them up the banner didn't appear. But in the simulation words regarding the cookie banner (e.g. *"performance", "cookie", "purpose", "analytic", "develop", "privacy"*) clearly stand out.
We have seen word clouds as one method of approaching the corpus created by the user simulation. In the next section the script for finding keywords in the user corpus is applied.

4.4.2 Finding Discourse

This section will try to answer the hypotheses from section 4.1 starting with:

1. The mobile website usage should limit the content that is visible to the user (simply because it is smaller) and thus should find less text in total.

In Figure 4.3 the average character count the user see per frame is shown. As expected users that use mobile devices see less text. Even when taking out

[4]https://spacy.io/ (10.06.2022)

Figure 4.4: The wordcloud from of user Arif (top-left), Kia (top-right), Mandy (middle-left), Menna (middle-right), and Suzannah (bottom), generated with "www.wordclouds.com/"

Name	Arif	Kia	Mandy	Menna	Suzannah
$\varnothing words$	0.333	0.18182	0	0	0.05556

Table 4.2: *The average count of words found for the users over all sessions. The script searched for "Zensus", "Affenpocken", "Tornado", "Kartenzahlung", and "Gewitter"*

Mandy and Kia out of the picture the results don't change. The first hypothesis can be answered with true considering the small sample size.

2. German-speaking users should find more information about discourses limited to the german speaking regions.

For the next assumption, online discourses related to the german region needed to be collected. I considered GoogleTrends as a possible source of those and found the topics people were interested in for the last two weeks. Search queries suggested the topics *"Zensus 2022; Affenpocken; ESC; Tornado; Kartenzahlung; Gewitter"*. Since *"ESC"* (representing the euro-vision song contest) is not only related to germany it gets dropped. Also non of these are particularly interest related making them valuable topics to search for. The results show in Table 4.2 how often per session one of the words appear[5]. For this test, the string distance ratio needed to be adjusted to 0.76, since searching for *"Gewitter"* also found *"Twitter"* consistently raising the count for the user Mandy to 8. Other than that, the only false occurrence happened with user Suzannah finding *"zensus"* in *"\nDatenscusnina"* (referring to german word *"Datenschutz"*) with 0.833 accuracy. As seen Mandy and Menna doesn't collect occurrences of those words since they mainly receive English content. That confirms the second hypothesis made above.

3. Most of the users interests don't overlap and therefore should participate in some distinct online discourses.

As a disclaimer for the last hypothesis, it needs to be stated that those analytics just serve as simple examples of how a user simulation can be used to analyze online discourses. Their goal is not to skip essential steps of full discourse analysis. That would reach beyond the means of this paper. Having the vague validity in mind the last hypothesis tries to answer if the content really changes with our interests. For that matter, a topic was chosen in the field of economics.

[5] 1 meaning a word is occurring once per browser session.

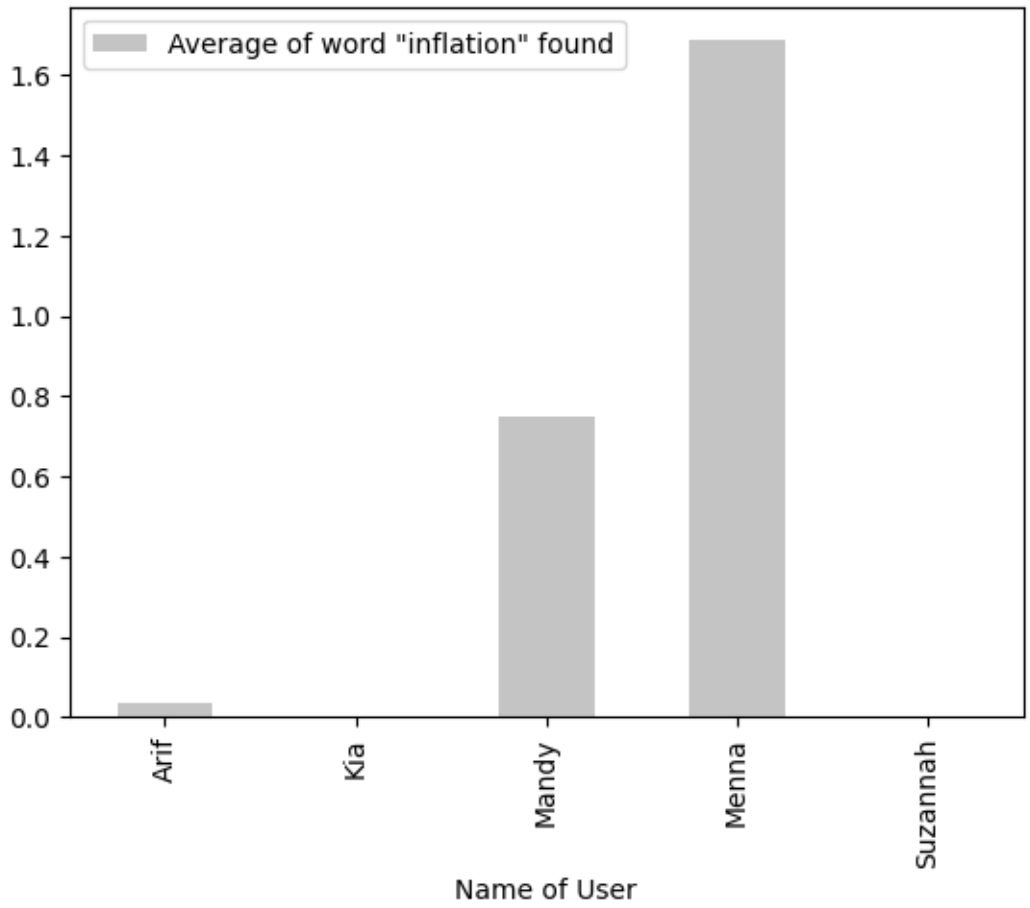


Figure 4.5: *Shown is the average occurrences of the word "inflation" per session.*

Figure 4.5 was calculated on the same basis as Table 4.2, but searched only for the online discourse regarding *"inflation"*. The word is an internationalism[6] and can be collected for all users. As expected the simulation of Menna and Mandy have the highest count of occurrences since some of their interests (e.g. economy or fiance) are directly related to *"inflation"*.
All hypotheses resolve in the expected way. The short assessment of the user simulation already shows that the user data created is highly dependent on who the user is, what interests he has or which device he uses. It also shows that the created data can be used to various extents to approach an online discourse analysis.

[6]It has the same meaning in multiple languages. For german and english "inflation" is even written the same way.

Chapter 5

Limitation, Future Work & Conclusion

5.1 Limitations

On many websites, when requiring the mobile variant, the HTML-page changes significantly and elements like thumbnails, and layouts have different names or don't appear at all. The playwright locator[1] can't click any elements on the side if they are not visible to the user. A better solution to click and find impotent content on the website is needed, it should be layout-, HTML, and language-independent. Otherwise, the interaction modules need to navigate over the URL and just use general interactions like scrolling.
Another limitation is that not a lot of public research was available on the general behaviors of users on the internet. There is a lot of research on what queries users type or which websites they visit, but analyzing how users actually use different accounts (e.g. facebook, instagram) and adapt the interaction modules accordingly could help to improve the user simulation.

5.2 Future Work

When working out this paper I didn't expect a lack of web user simulations for social research since the topic sounds so broad, but most implementations focus on specific platforms (e.g. social media bots) or are made for testing web apps. A lot of the software couldn't relate to other implementations or expand on previous user simulations. It can be stated that the user simulation was built from the ground up and therefore has a lot of room for improvement.

[1] Is used to locate different elements on the page. Those elements later can be used to interact with the page.

The first improvement can be achieved by expanding the scale of the user simulation. So more users can run for a longer time with a larger schedule to collect more information to build upon. When adding monitoring to the Kubernetes the quality of the data can also improve by minimizing holes in the data.
Another way to improve the collected information could be, to deepen the user interactions to make every user more unique and impactful. That can mean adding more interaction modules to the users, expanding the ones used at the time, or filling the accounts of the user with more applicable content (e.g. subscriptions, friends, groups). The schedule of the user can also see some refinement. When describing the schedule in section 3.2.1 it is already mentioned that the schedule could be expanded to cover a week or a month, maybe exceptions for the interaction modules can be added to hinder them from running sometimes. Creating new schedules could be simplified by automatically generating some modules depending on the interests[2].
In the evaluation section of the paper, the algorithms all rely on the raw data created in the *trace.zip* files and write them to spreadsheets for evaluation. A database could be created to store all these features in different tables and enable further data analysis and data queries.
Finally, a large idea that came up during development was to make users more dynamic by rebuilding their influences or interests on behalf of outside sources. We think of users that already like sport and during the time of the soccer world cup, they add the tournament to their usermodel-interests.

5.3 Conclusion

In this paper, we presented a way of simulating simple users on the internet. The goal was to collect information about online discourses they participate in. The unified usermodel can be adapted to describe a wide variety of users in their participation on the internet. The simulation for a user can be easily set up, understood, and changed during execution. Gathered information can be viewed in a graphical user interface provided by playwright or be searched with the help of OCR by a script. In the future, the software can be expanded in multiple ways, like a database for easier data accessibility or more complex interaction modules that improve the behavior of the users.
The paper clearly showed that a user-centered approach is useful when analyzing online discourses digitally, but there is yet very limited research done.

[2]For example adding the Google-News-Module with all the interests as queries could be a start to the schedule.

Bibliography

difflib - helpers for computing deltas - python 3.10.4 documentation, 04.06.2022. URL https://docs.python.org/3/library/difflib.html. 3

Wayback machine, 10.06.2022. URL https://web.archive.org/. 2.2

Trace viewer | playwright, 30.05.2022. URL https://playwright.dev/docs/1.20/trace-viewer. 3.5

Mircea-Dan Antonescu, Mark Guttenbrunner, and Andreas Rauber. *Documenting a Virtual World - A Case Study in Preserving Scenes from Second Life*. 2009. 2.3, 3.2

Erik Brynjolfsson and Joo Hee Oh. The attention economy: Measuring the value of free digital services on the internet. 2012. URL http://pinguet.free.fr/brynjoo.pdf. 1

GitHub - Webis Scriptor. Github - webis-de/scriptor: Plug-and-play reproducible web analysis, 17.05.2022. URL https://github.com/webis-de/scriptor. 3.4

Jürgen Habermas. *Strukturwandel der Öffentlichkeit: Untersuchungen zu einer Kategorie der bürgerlichen Gesellschaft : mit einem Vorwort zur Neuauflage 1990: Zugl.: Marburg, Univ., Habil.-Schr., 1961*, volume 891 of *Suhrkamp-Taschenbuch Wissenschaft*. Suhrkamp, Frankfurt am Main, 15. auflage edition, 2018. ISBN 9783518284919. 2.1.2

Martin Hilbert and Priscila López. The world’s technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011. doi: 10.1126/science.1200970. URL https://www.science.org/doi/abs/10.1126/science.1200970. 1

Johannes Kiesel, Florian Kneist, Milad Alshomary, Benno Stein, Matthias Hagen, and Martin Potthast. Reproducible web corpora. *Journal of Data and Information Quality*, 10(4):1–25, 2018. ISSN 1936-1955. doi: 10.1145/3239574. 1, 2.3, 2.1

Annegret März, editor. *Internet: Öffentlichkeit(en) im Umbruch*, volume 8.2008,2 of *Navigationen*. Schüren, Marburg, 2008. ISBN 9783894725501. 2.1.2

Robinson Meyer. How many stories do newspapers publish per day? *The Atlantic*, 26.05.2016. URL `https://www.theatlantic.com/technology/archive/2016/05/how-many-stories-do-newspapers-publish-per-day/483845/`. 3.2.1

Christian Pentzold, editor. *Online-Diskurse: Theorien und Methoden transmedialer Online-Diskursforschung*, volume 10 of *EBL-Schweitzer*. Halem, Köln, online-ausg edition, 2014. ISBN 9783869621234. URL `http://swb.eblib.com/patron/FullRecord.aspx?p=1909894`. 2.1, 2.1.1, 2.1.1, 2.1.2

Hans Pühretmayer and Armin Puller. Michel foucaults diskurstheorie und diskursanalyse, 2011. URL `https://www.univie.ac.at/sowi-online/esowi/cp/denkenpowi/denkenpowi-33.html`. 1

Jeff Schultz. How much data is created on the internet each day? 06.08.2019. URL `https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/`. 3.2, 3.2

World Wide Web Consortium W3C. Http archive (har) format, 10.05.2022. URL `https://w3c.github.io/web-performance/specs/HAR/Overview.html`. 3.1.3

Marvin Zelkowitz. *Advances in Computers: Information Repositories*. Elsevier, 2003. ISBN 9780080493510. 1

Appendix A

Usermodel

A complete usermodel in json format


```
{
  "name": "Arif Sykes",
  "device": "iPhone 11",
  "locale": "de-DE",
  "interests": [
    "dancing", "theatre", "culture", "philosophy",
    "governance", "web-culture", "society"
  ],
  "influencedBy": [
    "YouTube", "Instagram", "Der Spiegel",
    "LVZ", "Welt", "books", "Facebook"
  ],
  "frequentlyVisits": [
    {
      "type": "GoogleNewsQueries",
      "executionTime": "09:00",
      "subscriptions": [
        { "name": "Spiegel articles",
          "representation": "DER SPIEGEL" },
        { "name": "Jena", "representation": "Jena" }
      ]
    },
    {
      "url": "https://www.lvz.de/",
      "type": "OpenUrl",
      "executionTime": "9:10"
```

```
},
{
  "url": "https://www.artanalog.de/mein-blog",
  "type": "OpenUrl",
  "executionTime": "9:20"
},
{
  "type": "Instagram",
  "executionTime": "13:30"
},
{
  "type": "GoogleNewsQueries",
  "executionTime": "17:10",
  "subscriptions": [
    { "name": "locale opera", "representation": "
       oper leipzig"},
    { "name": "theater", "representation": "
       theater jena" },
    { "name": "philosophy", "representation": "
       philosophy" },
    { "name": "culture", "representation": "
       culture" },
    { "name": "Die Welt", "representation": "Die
       Welt" },
    { "name": "literature", "representation": "
       literature" },
    { "name": "LVZ", "representation": "LVZ" }
  ]
},
{
  "type": "YouTubeAbo",
  "executionTime": "18:00",
  "subscriptions": [
    {
      "name" : "WALULIS STORY - SWR3",
      "representation" : "walulissiehtfern"
    },{
      "name" : "WALULISDAILY",
      "representation" : "WALULISDAILY"
    },{
      "name" : "whatthehazel",
```

```
            "representation" : "whatthehazel"
          }
        ]
      }
    ]
}
```

Appendix B

Usermodel Strings

The corresponding usermodel-string from Arif:

"My name is Arif Sykes and my interests are theatre, culture, philosophy, governance, web-culture, society and especially dancing. In the recent time YouTube is quite an influence for me. I also consume Instagram, Der Spiegel, LVZ, Welt, books or Facebook. Luckily I made notes to my daily routine and can show them to you:

- At 09:00 o'clock I read some news articles about the topics Spiegel articles and Jena.
- At 9:10 o'clock I visit https://www.lvz.de/.
- At 9:20 o'clock I visit https://www.artanalog.de/mein-blog.
- At 13:30 o'clock I visit instagram and scroll through some new posts.
- At 17:10 o'clock I read some news articles about the topics locale opera, theater, philosophy, culture, Die Welt, literature and LVZ.
- At 18:00 o'clock I watch some youtube videos from the channels WALULIS STORY - SWR3, WALULISDAILY and whatthehazel.

Those were all sites I visit each day. I hope the insights in my daily web routine were helpful." - Arif Sykes, a fictional internet user

The other usermodels strings:

"My name is Kia Lucero and my interests are demography, fashion, gntm, confectionery, health, k-pop and especially social sciences. In the recent time Instagram is quite an influence for me. I also consume Twitter, Pinterest, YouTube, TikTok, Vogue, Gala or Der Stern. Luckily I made notes to my daily routine and can show them to you:

- At 22:30 o'clock I visit instagram and scroll through some new posts.
- At 20:15 o'clock I visit https://www.pinterest.de/search/pins/?q=fashion.
- At 20:15 o'clock I visit https://www.pinterest.de/search/pins/?q=dinner.
- At 20:20 o'clock I visit https://www.tiktok.com/foryou.
- At 20:25 o'clock I visit https://www.vogue.de/.
- At 20:30 o'clock I visit https://www.gala.de/.
- At 20:35 o'clock I visit https://www.stern.de/.
- At 20:00 o'clock I watch some youtube videos from the channels BUNTEtv, Josie, Karina Gomez, Delaney Childs, KOOKIELIT and kpopfication.

Those were all sites I visit each day. I hope the insights in my daily web routine were helpful." - Kia Lucero, a fictional internet user

---

"My name is Menna Rivas and my interests are finance, indoor planting, stock market, computer-hardware, japan, publishing and especially frogs. In the recent time Faz is quite an influence for me. I also consume Instagram, doppelgaenger, finanzen.net or reddit. Luckily I made notes to my daily routine and can show them to you:

- At 10:30 o'clock I visit instagram and scroll through some new posts.
- At 11:10 o'clock I read some news articles about the topics finance, crypto, stock market, computer-hardware, finanzen.net, Faz, japan and reddit.
- At 12:00 o'clock I visit https://www.reddit.com/r/germany/.
- At 12:10 o'clock I visit https://www.reddit.com/r/gaming/.
- At 12:20 o'clock I visit https://www.reddit.com/r/hardware/.

- At 12:30 o'clock I visit https://www.reddit.com/r/frog/.
- At 12:40 o'clock I visit https://www.reddit.com/r/finance/.

Those were all sites I visit each day. I hope the insights in my daily web routine were helpful." - Menna Rivas, a fictional internet user

---

"My name is Suzannah Wheatley and my interests are sports, football, gaming, fische, cannabis and especially formula1. In the recent time Instagram is quite an influence for me. I also consume Facebook, Youtube, Twitch, MeinMMO, SportBild or Auto Motor und Sport. Luckily I made notes to my daily routine and can show them to you:

- At 13:30 o'clock I visit instagram and scroll through some new posts.
- At 17:30 o'clock I visit instagram and scroll through some new posts.
- At 17:45 o'clock I spend some time on facebook.
- At 19:00 o'clock I watch some youtube videos from the channels Sky Sport HD, Formel1.de, Football Daily, SportsHD and MrBeast Gaming.

Those were all sites I visit each day. I hope the insights in my daily web routine were helpful." - Suzannah Wheatley, a fictional internet user

---

"My name is Mandy Pitt and my interests are tech, economy, tennis, wealth, nature, climate activism and especially politics. In the recent time YouTube is quite an influence for me. I also consume Twitter, Fefes Blog, sea shepard, ZDF, Tagesschau or Die Linke. Luckily I made notes to my daily routine and can show them to you:

- At 08:10 o'clock I read some news articles about the topics tennis, parliament, politic, economy, tech and sea shepard.
- At 09:00 o'clock I watch some youtube videos from the channels WATOP, Alicia Joe, Robert Marc Lehmann - Mission Erde and CultTennis.
- At 20:15 o'clock I visit https://www.zdf.de/.
- At 20:20 o'clock I visit https://www.tagesschau.de/.
- At 20:25 o'clock I visit https://twitter.com/explore.
- At 20:30 o'clock I visit https://blog.fefe.de/.
- At 20:35 o'clock I visit https://twitter.com/dieLinke.
Those were all sites I visit each day. I hope the insights in my daily web routine were helpful." - Mandy Pitt, a fictional internet user