Framework for Synchronous Gathering of Interaction- and Eyetracking-Data

Nico Lässig, Marius Maaß, Vithunan Maheswaran
Universität Stuttgart

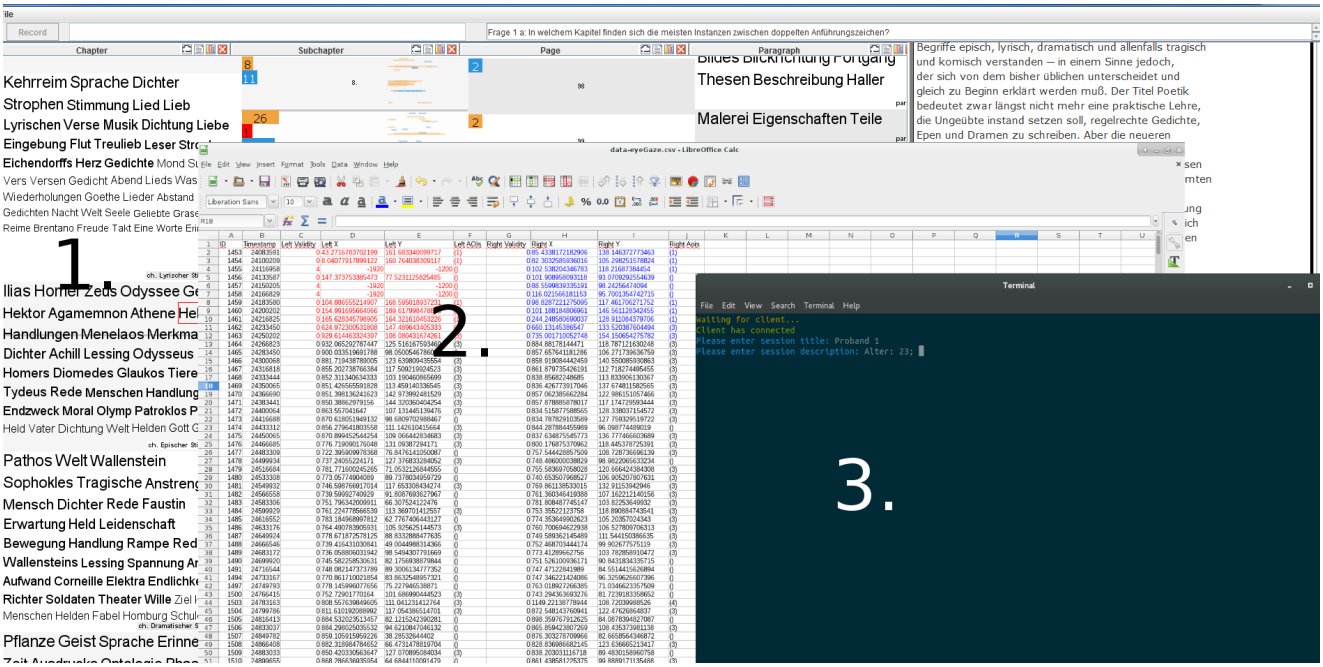

Figure 1: *Representation of the developed framework. 1) The application (VarifocalReader) where interaction and eye tracking data is recorded. 2) The resulting data being displayed in a spreadsheet application (LibreOffice Calc) 3) The commandline interface of the server application managing the recording sessions*

Abstract— The evaluation of user behavior has become a leading aspect when developing user-friendly software or analyzing the influence software modifications have on the user experience. These days, many methods to evaluate this data are available. This paper will focus on eye tracking, which is a method to evaluate user behavior by measuring the eye movement of a participant during the interaction with a graphical user interface of an application. Tracking the eyes is an important step for gathering eye movement data. However without more information about other actions of a user the value of the recorded data is limited. To supplement the eye tracking data other data sources like mouse movements or keystrokes can also be recorded. In this paper we present a framework that automatically records this data while keeping the different data sources synchronous in order to make analysis as accurate as possible.

Index Terms—Eyetracking, Synchronous Interaction, Data Collection, Thinking Aloud, Fixation Filtering

1 Introduction

Recording interaction data is a key aspect when analyzing the usability of a user interface of an application. Additionally, tracking the eye of a user yields important data about the user experience [4]. Recording interaction data as well as eye tracking data is mostly accomplished by using an external software. However, the application recording the eye movement data is typically not able to record the interaction data. This data has to be recorded using another application. So, the storage of the measurements occurs separately and without a uniform timestamp. As a consequence, the measured data has to be merged and synchronized manually.

Nico Lässig
st103855@stud.uni-stuttgart.de

Marius Maaß
marius.maass@studi.informatik.uni-stuttgart.de

Vithunan Maheswaran
vithunan.maheswaran@studi.informatik.uni-stuttgart.de

Consequently, by having to manually synchronize the eye tracking and interaction data the accuracy for the analysis gets lost. There is no unified architecture capable of handling both in a synchronous manner which leads to complications when analyzing the recorded data [4]. For this reason we have designed, implemented, and tested a framework which is capable of providing synchronous measurement and storage of interaction and eye tracking data. This makes the analysis much more accurate. The challenges and goals of our approach are to create a data model which is necessary to put the interaction data into a cohesive relationship and to ensure synchronous storage and recording when using the framework.

There are some important concepts that need to be explained before going into detail about the framework.

Interaction events: There are multiple forms how the user can interact with a given interface. This includes mouse movements,

mouse clicks, or keystrokes.

Thinking Aloud: To understand the user behavior, data is not always sufficient. For this reason the thoughts of the user must be considered. This is could be done by additionally asking the user to say their thoughts out loud. [6]

Area of Interest: In many user interfaces there are areas that need special attention. These are called "areas of interest" which means that events occurring within this area should be handled specially when analyzing the interaction data. There are static and dynamic AOIs. Static AOIs do not change position during the entire runtime of an application. Consequently, dynamic AOIs can change form and position at runtime which is especially useful when analyzing user interfaces with moving interface elements.

2 Related Work

The history of eye tracking dates back to over a century ago where initial observations about the eye movements were made [9]. The techniques improved over time and today eye tracking is used for a diverse set of problems e.g. marketing, psychology or usability research in Human-Computer Interaction (HCI).

These days and in previous work eye tracking and interaction data are measured separately using different external applications. For this reason eye tracking and interaction data have to be synchronized manually afterwards [4]. A consequence of merging the tracked data is that this could lead to an inaccurate analysis.

A modern approach to reduce the workload is called "eTaddy"[5], an integrative framework that should guide the supervisor in tracking and evaluating the eye tracking data. However this framework does not ensure synchronous tracking of eye activity and interaction data. Another modern approach of automatic processing is called "AdELE", a framework for **A**daptive **E**-**L**earning through **E**ye Tracking [3]. This framework tracks the interaction and eye tracking data synchronous as a requirement to enable an effective adaptive e-learning.

3 Model

Before we proceed to the documentation of our framework we will have a closer look at the model which we have created to describe how the measured data is represented in the framework and how each entry is structured.

The UML diagram in Figure 2 shows how the data of one session is organized. How each individual entry is structured up is described in the following:

- **Session:** A session contains all data items from one user interface recording session. In the implementation the data items are not kept in the memory but are instead streamed to multiple data files to reduce memory usage as the memory required to store all data items would put unnecessary strain on the system running the framework.

- **Metadata:** Each recording session has a small data structure that contains additional data related to one session. This contains a title and a description of the test that is currently being conducted. Both strings are entered by the test supervisor before the actual recording session begins.

- **DataItem:** The framework is designed around the idea that each interaction is saved in the form of a single specialized data item that contains all relevant information. Every data item has a globally unique numeric identifier and a timestamp that indicates when this event has occurred. This timestamp is the zero-based number of μs that have passed since the beginning of the recording session. Although the time is saved in μs the actual resolution of the timestamp depends on the operating system.

- **AOIChangeItem:** This item is generated when an AOI has changed. It contains the id, name, and the new screen bounds of the AOI.

- **EyeGazeDataItem:** The used eye tracker captures the location the user looks at every 16 ms. This data consists of four IEEE 754 double-precision binary floating-point numbers that specify the X and Y coordinates of each eye. The data is then converted into this data item and the framework also determines which AOI the user looked at.

- **KeyboardDataItem:** Whenever the user types on the keyboard this data item is generated. It contains the pressed keys and modifiers both as a textual description (e.g. `Ctrl+Shift` for modifiers and `F` for the pressed key) and as a numerical identifier which depends on the used UI framework. It also specifies which kind of event has happened by indicating if the button was pressed, released or typed. Normally one keyboard interaction generates multiple events because a single interaction consists of pressing and releasing the button. A *typed* event is generated if the button was quickly pressed and then released which happens when the user is typing text.

- **MouseClickDataItem:** Similarly to the keyboard interactions the mouse clicks are also captured. The contents of the data item are similar to the keyboard data but the application being tracked can also associate additional data with each event in order to provide additional data for later analysis.

- **MouseMotionDataItem:** In addition to individual mouse clicks the location of the mouse pointer is recorded. This data item only contains the current location of the pointer and the AOIs in which the pointer is currently located.

- **MouseScrollDataItem:** Typically a mouse also contains a scroll wheel. Interaction events from this device are also captured and are represented by a subclass of `MouseMotionDataItem`. In addition to the properties of the superclass it also contains the amount of how much the user has scrolled with this event.

- **ScreenshotDataItem:** When a screenshot is captured then a data item which contains the image data is generated. For each screen connected to the system one screenshot is captured. The data is a fully encoded image file which can be saved to disk without needing any further encoding.

4 Implementation

Based on the work shown in previous sections we created a list of design goals that the implementation of this framework would have to fulfill:

- Collection of the interaction data should require as little integration effort as possible

- Independence from UI-framework or programming language so the framework can be easily extended in the future

- Tracking of multiple AOIs that may be added or change position at runtime

The first design goal was solved by keeping the programming interface as minimal as possible. Most of the recorded data is captured automatically without further intervention of the application that is being tracked.

We solved the second design goal by turning the framework into a client-server application. A small language and UI-framework specific library would handle collecting the interaction data and then send it to the server application via a standard TCP-socket. This also opens up the possibility to run the client application on a different

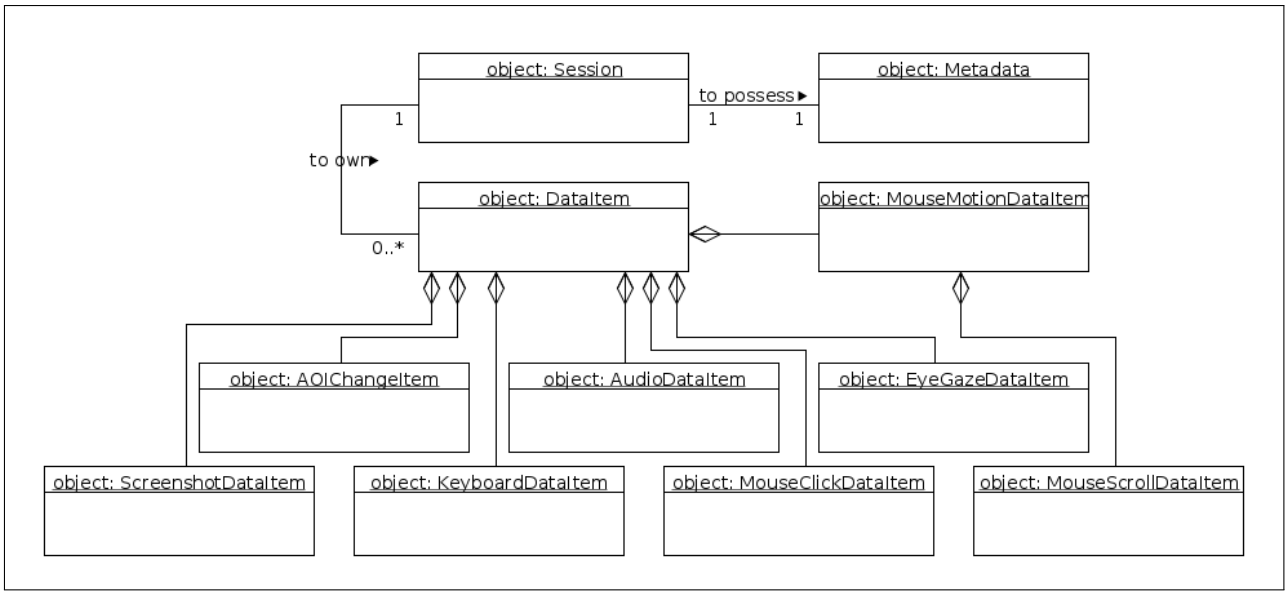

Figure 2: *Framework Model in UML notation*

device than the server which could be useful if the framework is extended to other types of devices (e.g. smarthphones) where saving and retrieving the data could be a problem.

4.1 The Client Library

As a part of this project the target language was Java and the targeted UI-framework was AWT/Swing which meant that we had to use these technologies when implementing the client library.

To start recording interaction data the client application only has to initialize the library and optionally add the AOIs by calling a method on the tracking state object. Mouse clicks are handled separately because they are frequently associated with some metadata. The client application must handle these events in some way (e.g. by registering a mouse click listener on the right component) and associate them with the appropriate metadata. The mouse event and the corresponding metadata is then passed to the client library which will handle the event similarly to the other events as described below.

4.1.1 Internal Structure of the Client Library

The client library is based on the concept of having multiple data sources (`IDataSource`) which generate a stream of data items (`IDataItem`) that have to be recorded. By keeping the data sources behind a generic interface, adding new data sources at a later time is easy to do without major changes of the client library. When a data source generates a new data item the client library uses the `IDataItem` interface to serialize it into the `OutputStream` of the TCP socket so the server can then process the data.

For tracking the eye movements of the user the Tobii Analytics SDK [1] is used. However it's relatively easy to add support for a different kind of eye tracker by implementing a new data source that uses the appropriate SDK. The Tobii SDK does not expose a Java API so we created a small JNI library that uses the C++ API to expose the API functions needed for our framework.

Recording of user interface events (e.g. mouse movements or keyboard events) is done by registering multiple global `AWTEventlisteners` which get called by AWT whenever a specific event has occurred. These AWT events are converted into appropriate implementations of `IDataItem` and then sent to the server like all other data items.

To support any possible shape of an AOI we decided to expose this functionality by using an interface. The interface has two functions:

- Expose a function that checks if a screen coordinate is within the AOI which has the signature `boolean hitTest(int x, int y)`
- Handling a set of listeners that get notified if the bounds of an AOI have changed

When using AWT, every interface element is a subclass of the `Component` class which has all the necessary information to perform the mentioned hit test and and notify the listeners should the bounds of the AOI component change. We used this to implement `IAreaOfInterest` for a `Component` instance. This makes the AOI usage easier to realize for client applications and can also be used as an example for how an implementation should be written.

The framework also handles recording an audio stream for when a user is thinking aloud. The Java Runtime offers an API for capturing audio from various sources in the system. However choosing which source to capture has to be done by test supervisor. To minimize the required level of interaction while performing the recording our framework simply captures all audio streams and saves them to multiple files. After the user studies are finished the right files can be selected by checking which file contains the right audio.

Initially, we also wanted to capture periodic screenshots or even a full video of the screen the user is looking at. However this caused performance issues because the system locked up for a few milliseconds each time the framework took a screenshot. This would distort the interaction data captured of a user.

4.2 The Server Application

The server application is responsible for deserializing the data received from the client and saving it into multiple **C**haracter **S**eparated **V**alues (CSV) formatted files. After deserializing the data items, they are handed to a separate thread which demultiplexes them into multiple streams of the same type (e.g., all eye gaze data items will be in one stream, mouse move items in another). The data items are then formatted into a line where each value is separated by a `;` character. This format is supported by common spreadsheet applications.

After a client has connected to the server, there is the possibility of specifying some metadata for this session that gets saved alongside the recorded data. This can be used to give the session a meaningful title and description. After the supervisor of the test has entered this data it is saved to a JSON (**J**ava**S**cript **O**bject **N**otation) formatted file which can be read at a later time to reconstruct the data.

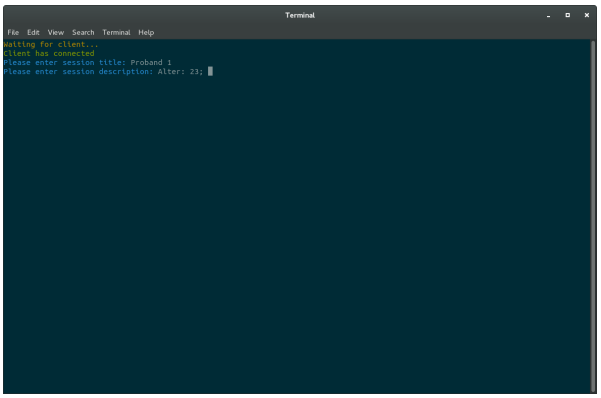

Figure 3: *Server commandline interface running in a terminal emulator on Ubuntu GNOME*

Currently, the server application only uses a basic commandline interface (see Figure 3) which allows to enter the required meta data. The internal structure of the server was designed around the idea that the UI could be easily replaced. The server uses an interface to communicate with the user interface. Creating a graphical user interface for the server could be done without major changes to the server code.

The server is written in C# and runs on both the Microsoft .NET and the open-source Mono platforms. This it can run on all major desktop platforms without modifications.

4.3 Communication Protocol

The communication protocol between the client and the server is a central part of our framework. It is the basis for future extensions to other programming languages or UI frameworks. After the TCP-connection is established the client first sends the ASCII string `"EyeTrackingEvaluation"` followed by a carriage return (CR) and a line feed (LF) character. The next line contains a version number (currently that is 1) again ended by the `CR LF` sequence. The client library can send some additional options to the server. The only option currently supported is `Application-Name` which should identify the application that is used. The format of these options is the same as in the HTTP-Header [7]. After sending all options, the client sends an empty line (just containing `CR` and `LF`) to signal the server that all options have been sent. The client now waits for the server to send the string `"Ready\r\n"` until proceeding with initializing the interaction tracking. This mechanism allows the server to interactively ask the supervisor to enter the metadata for this session. After all this data has been gathered, the server sends the ready string (`"Ready\r\n"`) and the client begins transmitting the serialized interaction data in binary form. After one session has ended, the server begins to listen on the same port (by default 43248) as before so the next tracking session can begin without interruption.

4.4 Data Import

Once the data has been recorded it needs to be analyzed somehow. The amount of recorded data is commonly too much to be analyzed manually. So, it needs to be converted into a data structure suitable for automatic processing. A common data structure for this kind of task is a database that contains all the data and allows a more efficient retrieval of individual items or a group of related records. For this project the database is a Microsoft SQL Server which can be accessed with a .NET based language. For this reason a generic parser for the generated interaction data was written in C#. The language features of C# make writing a CSV parser easy while keeping it fairly robust.

At the beginning of the parsing process, the saved metadata is converted into the appropriate data structure using the JSON deserializer available in the standard .NET and Mono framework.

To ensure a proper initialization order, the parser first reads the file containing the individual AOI events. Most other data items depend on this data so it is important that the AOI data is available when the rest of the data is parsed. The AOI events are converted into so called "`AreaOfInterestPoint`" data structures which contain the timing and location information. If an event contains informations about a new AOI, then the importer will also generate an "`AreaOfInterest`" object that contains the ID and the name of the AOI. Additionally, this object holds all points related to that AOI. The final data structure contains a list of AOIs with each individual object again containing a list consisting of the individual AOI points.

After creating the AOI data structure, parsing the actual interaction data is a simple process. Most importers parse the text data into the appropriate number formats (e.g. a 32 bit integer or a IEEE 754 double-precision binary floating-point) and create an object for the event. If an event contains information about which AOI was hit, then the parser has to do some additional work to cross-reference this event with the right AOI object. This is done by parsing the list of hit IDs (in the format `(1|2|...)`) into a sequence of integers. These represent the IDs of the hit AOIs. Then the parser searches in the AOI list for the first AOI object that has the same ID. This generates a list of references to the right AOIs.

After the importer is finished the `DataBase` object contains all data recorded in one session at which point it can be converted into the right data structure suitable for analysis.

5 Post-processing

During the recording, several CSV files are generated, which can be used for the analysis. The recorded data of the eye movement looks like the following (Fig. 4):

1. Each recorded data has its own ID.
2. The timestamp of the recorded data. It starts with 0, when the first data is recorded.
3. The range of the validity is between 0-4, it depicts how good the eye was tracked (0 = eye coordinates tracked without any problems, 4 = the eye movement could not be tracked)
4. These are the tracked eye coordinates. If the validity equals 4, then the coordinates in the table are default, -1920 and -1200.
5. The AOI where the coordinates are located. If they are located in a none predefined area, the default is "()", which means that no AOI is behold. This is also the case, when the eye gaze could not be tracked.

As one can see, the raw data consists of data from both eyes, the data is recorded for each eye independently. In this figure the data of the left eye is depicted as red, the other one as blue.

However, for our evaluation of the user study not all eye gazes are important to consider. There are two general types of eye movements, fixations and saccades. Fixations are eye movements where participants focuses on a particular region whilst saccades are rapid eye movements between such fixations. For most analyses the fixations are needed, that is why we decided to implement another external program. Thus, the users can decide if they need every eye movement or just the fixations. This program filters out the saccades because they make the dataparsing more complex and the analysis gets more elaborate because more points are considered. The algorithm which determines eye gazes fixations or saccades has an impact on the analysis, because bad algorithms rather filter out too many points or too few.

ID	Timestamp	Left Validity	Left X	Left Y	Left AOIs	Right Validity	Right X	Right Y	Right Aois
1453	24083591	0	43.2716783702199	161.683340099717	(1)	0	85.4338172182906	138.146372773463	(1)
1454	24100209	0	8.04077917899122	160.764038309117	(1)	0	82.3032585936016	105.298251578824	(1)
1455	24116958	4	-1920	-1200	()	0	102.538204346783	118.21687384454	(1)
1456	24133587	0	147.373753385473	77.5231125825485	()	0	101.908958093118	93.0709292554639	()
1457	24150205	4	-1920	-1200	()	0	88.5599839335191	98.24256474094	()
1458	24166829	4	-1920	-1200	()	0	116.021566181153	95.7001354742715	()
1459	24183580	0	104.886555214907	168.595018937231	(1)	0	98.8287221275095	117.461706271752	(1)
1460	24200202	0	154.991695664066	189.617998478843	(1)	0	101.188184806961	146.561128342455	(1)
1461	24216825	0	165.628345798905	164.321610453226	(1)	0	244.248580690037	128.911084379706	(1)
1462	24233450	0	624.972300531808	147.489643405333	(2)	0	660.13145386547	133.520387604494	(3)
1463	24250202	0	929.614463324397	108.080431674261	(3)	0	735.001710052748	154.150654275782	(3)

1 2 3 4 5

Figure 4: *Eye Gaze Data file generated by our framework. Described in chapter 5.*

The raw data we get from our implementation returns the coordinates and the AOIs seen by the right and the left eye separately. Therefore, we first have to average the value of both eye gazes. The exact process is explained in the next subchapter.

Additionally, we define two types of saccades [11]. When both eye gazes are not tracked, it is a type-1 saccade. The entries of type-1 saccades are removed immediately. So they are not part of the computation of our algorithm. Type-2 saccades are removed at the end of the process and are detected by implemented algorithm. Thus, the characteristic which determines if an eye gaze is a fixation or saccade depends on the implemented algorithm. The detection of the type-2 saccades in our algorithm is described in the subsection "Detection and filtering of Fixations and Saccades".

5.1 Eye Gaze Computation

As already mentioned above, the raw eye gaze data consists of data from the left and right eye separately. For the evaluation, and therefore for our algorithm, however, we need one eye gaze in each entry. We first have to calculate the eye gaze via the coordinates of the left and right eye. If both eyes are tracked, the arithmetic mean of the coordinate is determined. If both eyes could not be tracked, the entry is deleted immediately, as already noted. If just one eye is tracked, we take the coordinates of the tracked eye. Furthermore, we have another idea which can be realised as well. The x-coordinates of the tracked left eye is mostly left to the x-coordinates of the tracked right eye. Therefore, we could find an average distance and then adapt the eye gaze. This slight change would make the coordinates of the eye gaze more realistic and therefore it would enhance the accuracy of the algorithm.

Finally, we have to determine and adopt the focused AOI. For that, we have to import the data where the bounds of the AOIs are predefined.

5.2 Detection of Fixation- and Saccade-Points

We chose the "Velocity-Threshold Identification" algorithm as the basis for the implementation, which we call "Fixation Filter". We chose it for several reasons: it is a fast algorithm, which is easy to implement compared to other algorithms and the accuracy of the algorithm is also good [12].

The idea of the "Velocity-Threshold Identification" algorithm is, that the points get a type (fixation or saccade) assigned based on their point-to-point velocities. The formula for the calculation is:

$$\text{velocity} = \frac{\text{distance}}{\text{timespan}}$$

$$\text{distance} = \sqrt{(\text{x-coord.} - \text{prev. x-coord.})^2} + \sqrt{(\text{y-coord.} - \text{prev. y-coord.})^2}$$

$$\text{timespan} = \text{timestamp} - \text{prev. timestamp}$$

We decided to assign eye movements with a point-to-point velocity ≤ 100 deg/sec as a fixation, otherwise as a saccade.

Furthermore we categorize all tracked eye movements as a saccade, when the coordinates are not located in a defined AOI. This idea is used in the I-AOI algorithm [12].

After we categorized the fixations and saccades, we have to determine the duration of the fixations. The duration defines how long a participant looks at a particular region.

5.2.1 Duration

There are two steps to identify the duration of a fixation. First, fixation points are formed into fixation groups, if they fulfill two conditions. Condition one is that these points are between two (type-2) saccades. The other condition is, that the recognized AOI is the same in successive fixation points.

To get the duration of a fixation, we take the first and last timestamp of the fixations in the fixation group and compute the difference.

At the end, we remove all saccade points as well as fixation groups with a duration < 200 *ms*.

5.3 GUI and Visualization

We implemented a GUI that provides a front end for the previously described algorithm. The GUI (shown in Figure 5) allows to select an eye gaze data file and an AOI data file which were generated by our framework. The user can then select an algorithm for computing the fixations, execute it and examine the results which are shown in a table. At this time, the program provides the described algorithm as well as the I-AOI and I-VT algorithm itself which are used for it.

The resulting fixation data is written to a new CSV file which can be used as input for a visualization of the data. Figure 6 shows a possible visualization which was generated using the software presented in [4].

6 Use Case

In the following, we present a user study based on our implementation. For the user study we used the program "VarifocalReader" [10].

6.1 VarifocalReader

The VarifocalReader offers a multi-layer visualization approach and supports analysts in exploring and understanding documents based on the inherent structure of a document (e.g. chapters, pages and paragraphs) [4]. The VarifocalReader provides several views to show different abstractions of text, annotations appropriate to users needs, as well as results of search requests.

There are different visualizations that can be attached to each layer, except the last one which displays the text. These are either bar charts, pictograms, or word clouds. Clicking on a word in the word cloud marks all such words.

The hierarchical perspective on text documents and the navigation concept are based on the SmoothScroll approach [13]. It enables an analyst to navigate through the visualization and to keep track of the current position across all layers.

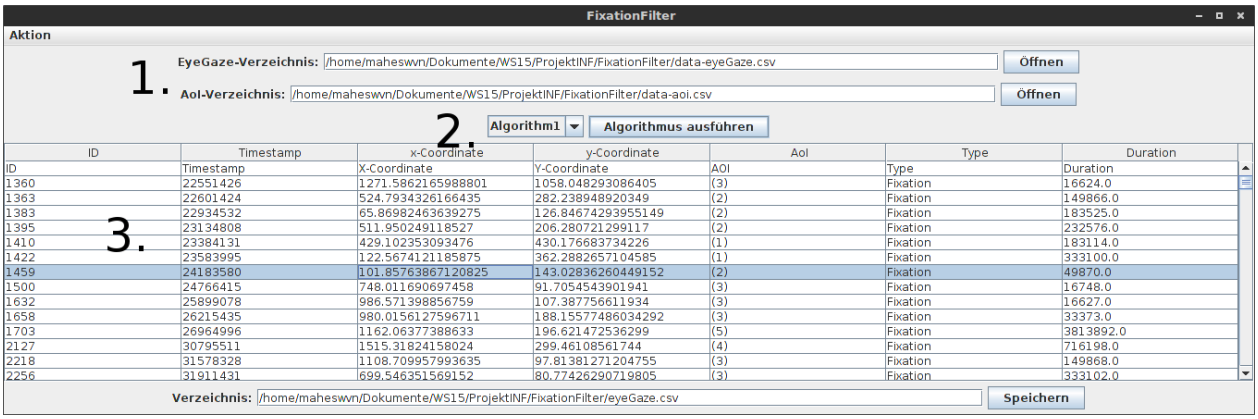

Figure 5: *The GUI of the Fixation Filter. 1) Selection of eye gaze and AOI data files 2) Selection of which algorithm should be used to compute the fixations 3) Table showing the results of the algorithm*

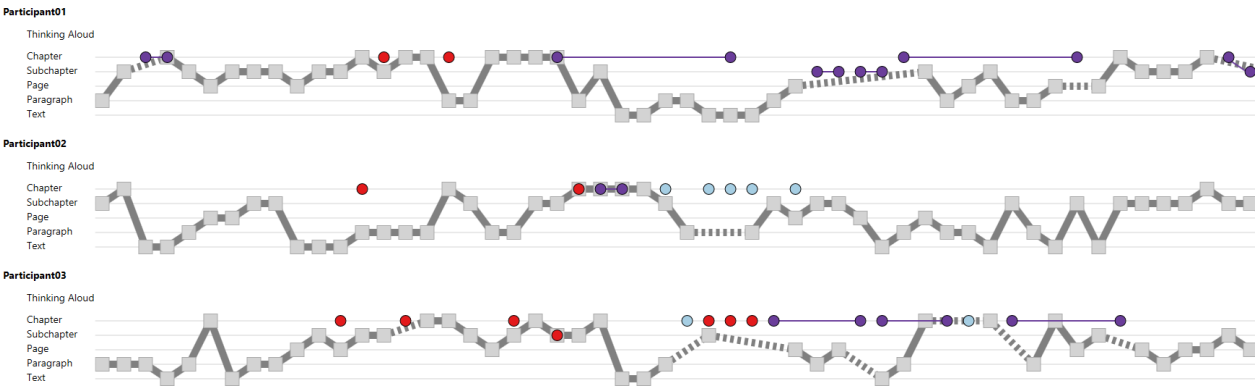

Figure 6: *Visualization of the recorded data using an AOI Sequence Chart [8]. This figure shows AOI sequence charts from three different participants. Interaction data are classified and therefore each interaction has a specific color based on the classification. All interaction categories and their visual representation are precisely described in [4].*

Each layer was defined as an AOI, independently of the used visualization. Hence, it made no difference for the AOI if the participants used a bar chart or pictogram. These layers are: chapters (1), subchapters (2), pages (3), paragraphs (4), and the text itself (5) as depicted in Figure 7.

6.2 Integration of the Framework

Integrating the framework into "VarifocalReader" was a fast process. In the source code of "VarifocalReader" we called `InteractionTracker.initialize` in the `main` function and stored the returned `TrackingState` object in a global variable.

As soon as the main window was created, multiple AOIs for the individual user interface areas (see Figure 7) were added. From previous studies there was already code present which saved the mouse click interactions into an Excel file. We used this code and redirected the `MouseEvents` to the `TrackingState.onMouseClicked` function. The additional data for this function call was also already recorded. This data was converted into a `InteractionInfo` instance which was also passed to `TrackingState.onMouseClicked`.

This shows that our design goal of creating an easy to integrate framework has been fulfilled. The recorded data also fully meets our expectations. It contains all data that was captured by previous tools and also has additional data streams like keyboard events or mouse scroll events. Using the data importer presented in section 4.4 we also successfully imported the data into an existing eye tracking data visualization application which was presented in [4]. The resulting visualization is shown in Figure 6.

6.3 Tobii Studio

For our approach we work with the tracking system developed by Tobii [2]. It is a complete eye tracking studio which provides many features to record analyze and visualize eye tracking data of a participant.
Before the use case starts, Tobii studio is used to calibrate the eye tracker for each user. The calibration is necessary to provide accurate test results during a study. In the first testing sequence with Tobii Studio we noticed that eye tracking requires some demands on the quality. An example would be a user wearing glasses. Glasses lead to difficulties in tracking the eye coordinates. Using an external dialog offered by Tobii Studio, we could examine the tracking status. When we noticed that the track status is under 80% of successful eye tracking, it was recommended to cancel the task of a user.

6.4 Example User Study

For the user study, we prepared several questions for the literary texts "Grundbegriffe der Poetik" (english: Basic Terms of Poetic) by Emil Staiger and "Iliad" by Homer.

6.4.1 Environmental Conditions and Technical Setup

The study was performed in a laboratory isolated from external distractions and with the rollers shuttered down. The participants had to turn off their mobile phones and other electronic devices which could disturb the user study.

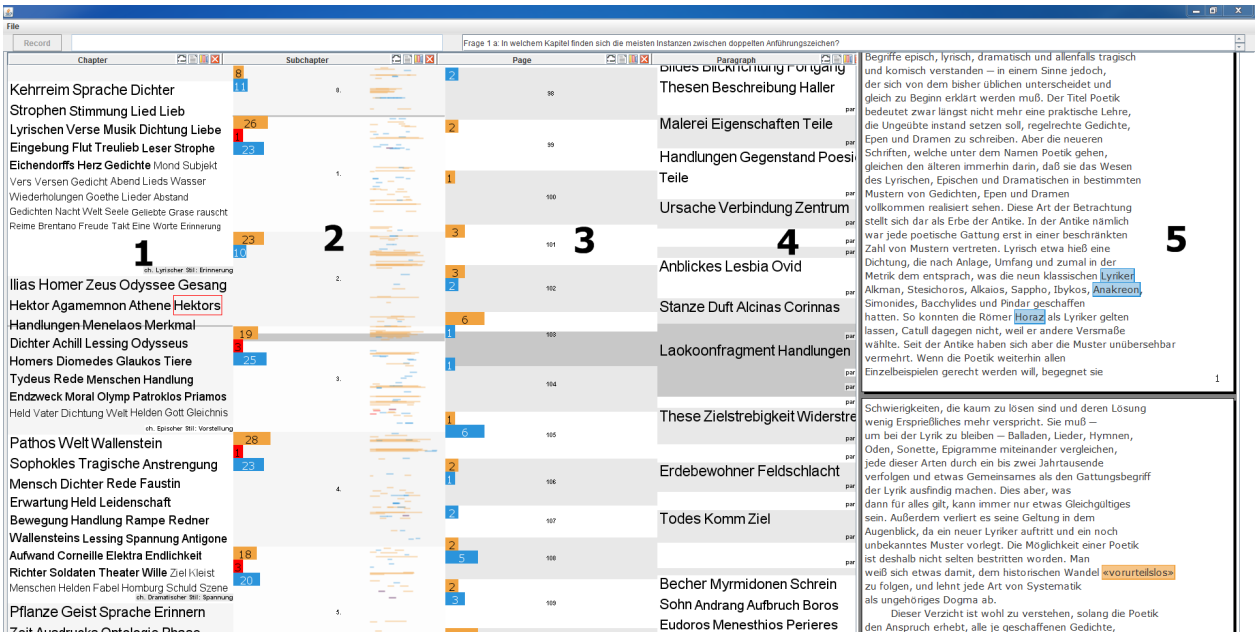

Figure 7: *VarifocalReader. The literary texts are divided into layers showing chapters (1) (with word clouds), subchapters (2) (with bar charts and pictograms), pages (3) (with bar charts), paragraphs (4) (with word clouds), and the text itself (5). As can be seen in this figure, the word Hektors is selected. The red marked bar charts in the second layer show in which subchapters the word is used and how often it is used.*

We tracked all eye and mouse movements as well as the interactions made by the participant. Additionally, we recorded an audio file, where the user described what their intentions are for each action as well as what their answers are for the tasks.

To start the user study, first "Tobii Studio" and the server had to be started. After calibrating the participant we selected the program for our user study. Before the program started we had to select an ID for the user and give additional information about him/her which we use for the analysis.

Eye movement data was recorded with a Tobii T60 XL eye tracker (sampling rate 60 Hz) with a 24-inch screen and a resolution of 1920 $\times$ 1200 pixels. To calibrate the eye tracker we conducted a five point calibration. For the thinking aloud part we used the Rode NT-USB studio mic.

The raw data was exported as .CSV-files. Each category (Eye Gaze, Mouse Movement, Mouse Click, Mouse Scroll, Keystroke and AOI) had its own file. The keystroke file is irrelevant in this user study, as the keyboard is not used in this program. In the AOI-file, the predefined AOI bounds are saved. Which AOI is viewed each time can then be looked up in the eye gaze file. Later described in Figure 4.
The recorded audio files had to be moved manually by us, because these files were saved in a subfolder of the client.

6.4.2 Participants

We conducted the study with six participants (one female, five male). The average age of the participants was 22.8 years (min = 21, max = 25). four of the participants had a slight experience with the program whilst two participants have never seen or used it before. Each participant successfully performed an Ishihara test and a Snellen chart to confirm that participants were physically able to accomplish the given task. The study took about 25 min, depending on the speed and experience of participants.

6.4.3 Procedure

First, the participants were asked to fill out some personal information (e.g. about their age and if they already have used the VarifocalReader). Then, we gave the participants a brief introduction into Thinking Aloud. They were admonished to always tell, what they are doing, what their intention is, as well as to answer the prepared questions. We also informed the participants on how to scroll in the program. Nevertheless, they were not informed on how the program exactly works.

Afterwards, we tested their visual ability with an eyesight test as well as a color test. Then, we calibrated the participant with a five point calibration as mentioned before.

Each participant had to solve several tasks where each new task was slightly harder than the previous one. All questions were based on a single text and with no pause between them for a continuous flow.

For each task, participants had to find out, for example, where specific words or topics are or where specific words are used most.

Afterwards, participants answered several questions about the VarifocalReader. How they found the tasks and the program itself, if it is easy to use, and how useful they find this program. The participants also gave some advice about how to improve the program and had the chance to give final remarks.

7 Analysis

The main reason for the use case is that we want to test our framework and evaluate the results after we filter out the saccades with help of our fixation filter to see if the evaluation is facile and how big the effort is for the analysis.

We analyze the results from the participants using three main categories: speed, correctness and eye gazes. Additionally, we distinguish other characteristics as well. Furthermore, we grouped the probands for the analysis into two different classes based on their experience with the VarifocalReader: inexperienced and experienced users.

At the end, we want to evaluate the evaluation, because the reason for use case was not to test the VarifocalReader, instead to test the workload with aid of our implementations.

7.1 Hypothesis

Our hypothesis for the user study is that inexperienced users need way more time than the experienced ones, but that it will become less time difference between the experienced and inexperienced probands for each new task, because the inexperienced probands learn more and more about the program in the progress and therefore become slightly experienced probands themselves. However, we believe that there is not any difference about the correctness of the answers between the probands.

7.2 Visualization of Recorded Data

Fig. 6 shows a sector of the visualization from the eye gaze of the tracked data from our use case, after saccades were filtered with the "Fixationizer", using an "AOI Sequence Chart" [8]. This representation of the data is used for the analysis of the use case. Every participant has its own AOI Sequence chart with each AOI having its own timeline. This allows an easy comparison of individual participants. Eye movements are represented as gray rectangles connected by lines whilst interaction data are depicted by colored circles. The meaning of each color is described in [4].

In this part of the visual representation it can be seen that participant 3 has the focus on the paragraph layer at first, then looks at the text and then back at the paragraph again. It also can be seen e.g. that participant 3 reads the text often for a short time whilst the other two participants focus on the text for a longer timespan, but not as often as participant 3.

The visualization of the complete user study can be analyzed simply, because of the visual representation with the same timeline used for each participant and because the interactions are categorized and each category has its own color.

7.3 Task Solving

- **Speed:** The slightly more experienced users needed $\sim 4\frac{1}{2}$min less to solve the tasks in comparison to the inexperienced users. This is kind of unsurprisingly, however, the inexperienced users needed about the same time for each task. The experienced users, though, needed more time for each new task, because the tasks got harder and more elaborate each.

- **Correctness:** There was no major difference between the experienced and inexperienced users. Most tasks were solved correctly, but there were few tasks which several probands struggled to solve.

- **Eye Gazes:** Another significant difference between experienced and inexperienced users is that the experienced ones nearly did not look on the icons, only for short time to select them whereas the inexperienced ones watched them more closely, to understand what they could mean, before selecting them. The biggest difference however was that the experienced participants did not watch the text often, because you did not really needed to read or watch it in order to answer the questions. They just looked on the text to verify if they selected the right word etc.. The inexperienced users have read pretty much at the beginning of the tasks and also tried to find some solutions in the text.

- **Other Characteristics:** The inexperienced participants played a little bit around with the program before trying to solve the first question. The reason behind this is that they wanted to get familiar with the program first in order to solve the rest of the tasks more quickly.

  One common problem which five out of six probands faced was that they struggled a few times with scrolling, because it is an extraordinary technique, but it got better the more time they were into the tasks. For two participants a mouse scroll .csv-file was created, which means that they still tried to use to scroll with the mouse.

7.4 Usability of our Implementations

Evaluating the user study is more efficient and less complex, because when using our framework, the results of the use case are more precise because all data is synchronized automatically. Tracking of data with our framework is working without any issues.

We represent the results with the software from [4] as depicted in Fig. 6. With aid of our fixation filter we reduce the exported data to the important informations we need for the evaluation, that is why the analysis of the user study is neither hard nor time-expensive.

As already mentioned in section "6.2 Integration of the Framework" the integration of our framework into the VarifocalReader was a fast process as well. The visualization of our data with the visualization software also proceeded without any complications.

In conclusion, with aid of our framework the workload of analysis of user studies is reduced.

8 Future Work

The focus for future work should be put on the visualization of the captured data and expansion to other UI-frameworks and devices.

The server application could be extended to provide a more user friendly interface as opposed to the current commandline based interface. Furthermore the server could be optimized to consume fewer system resources in order to reduce the impact of the recording on the use case.

The client library could be changed to conduct the eye tracker calibration without needing to use the Tobii Studio. As mentioned above the methods provided by the Java Runtime to record screenshots causes unacceptable performance problems. This could be solved or at least reduced by using a dedicated screen recording software that is specifically designed to perform more efficiently than the Java Runtime. The current method of recording what the proband says captures all available audio sources on the system. The correct audio stream must be selected later by the person doing the analysis. Providing a way to choose which audio source to record would streamline the process of recording interaction data. Currently the audio capture system is also limited to saving the recorded audio on the system the client runs on. By sending the audio data to the server via the TCP-socket and saving it with the other recorded data the files would not need to be copied by the person conducting the test which would remove another step needed for successfully executing a use case.

The preparation of the tested application needed before being able to conduct a use case is already pretty minimal but it could be further reduced by creating an external application that could be used to specify the AOIs that should be tracked. This could be done by selecting user interface elements and saving this information for later use by the client library.

The "Fixationizer" is yet an extern application for detecting and removing saccades and computing the duration of fixations. Future work could integrate the filter algorithm in our framework, so that saccades will be filtered out internal and automatically. The user should decide if saccades are needed or are not needed, therefore the filtering should be optional. Furthermore, there are many approaches for filtering fixations and saccades, that is why the algorithm can be optimized and/or new algorithms could be added to our existing implementation, that the user can choose the preferred filter algorithm.

9 Conclusion

The goal of this project was to create a framework for synchronously recording various interaction data from an application. This was done creating a model containing the various types of interaction data which share a common time base. The implementation uses this model to capture the data and save it in multiple data files but

now all these individual data files are sharing a synchronous time base and can be easily analyzed. The conducted use case has shown that the framework fulfills this goal adequately.

REFERENCES

[1] Tobii analysis sdk. http://www.tobiipro.com/product-listing/tobii-pro-analytics-sdk/. Accessed: 2015-06-10.

[2] Tobii studio manual. http://www.tobiipro.com/siteassets/tobii-pro/user-manuals/user-manual—tobii-studio.pdf. Accessed: 2015-10-04.

[3] V. M. G. Barrios, C. Gütl, A. M. Preis, K. Andrews, M. Pivec, F. Mödritscher, and C. Trummer. Adele: A framework for adaptive e-learning through eye tracking. *Proceedings of IKNOW*, pages 609–616, 2004.

[4] T. Blascheck, M. John, S. Koch, K. Kurzhals, and T. Ertl. Va2: A visual analytics approach for evaluating visual analytics applications. *IEEE Transactions on Visualization and Computer Graphics*, 22(1), 2015.

[5] T. Blascheck, M. Raschke, and T. Ertl. etaddy-ein integratives framework für die erstellung, durchführung und analyse von eyetracking-daten. *Gesellschaft für Informatik, editor, GI-Edition Lecture Notes in Informatics Informatiktage*, 12:111–114, 2013.

[6] K. A. Ericsson and H. A. Simon. Verbal reports as data. *Psychological review*, 87(3):215, 1980.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

[8] K. Holmqvist, M. Nyström, R. Andersson, R. Dewhurst, H. Jarodzka, and J. Van de Weijer. *Eye tracking: A comprehensive guide to methods and measures*. Oxford University Press, 2011.

[9] R. Jacob and K. S. Karn. Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. *Mind*, 2(3):4, 2003.

[10] S. Koch, M. John, M. Wörner, and T. Ertl. Varifocalreader in-depth visual analysis of large text documents. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2014.

[11] N. Lässig. Detection of fixations and saccades. *Eye-Tracking in der Visualisierung*, 2015.

[12] D. D. Salvucci and J. H. Goldberg. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, ETRA '00, pages 71–78, New York, NY, USA, 2000. ACM.

[13] M. Wörner and T. Ertl. Smoothscroll: A multi-scale, multi-layer slider. In G. Csurka, M. Kraus, L. Mestetskiy, P. Richard, and J. Braz, editors, *Computer Vision, Imaging and Computer Graphics. Theory and Applications*, volume 274 of *Communications in Computer and Information Science*, pages 142–154. Springer Berlin Heidelberg, 2013.