Incorporating the User Attention in User Interface Logs

A. Martínez-Rojas[1][a], A. Jiménez-Ramírez[1][b], J. G. Enríquez[1][c] and D. Lizcano-Casas[2][d]
[1]*Departamento de Lenguajes y Sistemas Informáticos, Escuela Técnica Superior de Ingeniería Informática, Avenida Reina Mercedes, s/n. 41012, Sevilla, Spain*
[2]*Universidad a Distancia de Madrid, UDIMA, Vía de Servicio A-6, 15, 28400 Collado Villalba, Madrid, Spain*

Keywords: Robotic Process Automation, Noise Filtering, Eye Tracker, Gaze Analysis, Human-Computer Interaction, Feature Extraction

Abstract: Business process analysis is a key factor in the lifecycle of Robotic Process Automation. Currently, task mining techniques provide mechanisms to analyze information about the process tasks to be automated, e.g., identify repetitive tasks or process variations. Existing proposals mainly rely on the user interactions with the UIs of the system (i.e., keyboard and mouse level) and information that can be gathered from them (e.g., the window name) which is stored in a UI event log. In some contexts, the latter information is limited because the system is accessed through virtualized environments (e.g., Citrix or Teamviewer). Other approaches extend the UI Log, including screenshots to address this issue. Regardless of the context, the aim is to store as much information as possible in the UI Log so that is can be analyzed later on, e.g., by extracting features from the screenshots. This amount of information can introduce much noise in the log that messes up what is relevant to the process. To amend this, the current approach proposes a method to include a gaze analyzer, which helps to identify which is process-relevant information between all the information. More precisely, the proposal extends the UI Log definition with the *attention change* level, which records when the user's attention changes from one element on the screen to another. This paper sets the research settings for the approach and enumerates the future steps to conduct it.

1 INTRODUCTION

Over the last decade, the industry has embraced Robotic Process Automation (RPA) as a new level of process automation aimed at mimicking the behavior of humans interacting with user interfaces (UI) rather than orchestrating sequences of API calls (van der Aalst, 2016). It is recognized that a successful RPA adoption goes beyond simple cost savings but also contributes to improved agility and quality (Asatiani and Penttinen, 2016; Capgemini, 2017; Lacity and Willcocks, 2015). Most RPA projects start by observing human workers performing the later automated process. Recently, terms such as Robotic Process Mining (Leno et al., 2020; Agostinelli et al., 2020) and Task Mining (Reinkemeyer., 2020; Mayr et al., 2022) have been coined to refer to the application of process mining techniques (van der Aalst, 2016) for the discovery of the processes behind human behavior. To this end, interactions with the user interface are recorded in what is known as a UI Log (i.e., a series of mouse and keyboard events). These methods have become very convenient in helping analysts to identify candidate processes to be robotized, their different variants and their decision points efficiently (Jimenez-Ramirez et al., 2019).

However, human work is sometimes based on information that is displayed on the screen (cf. Fig. 1 *a*, where the same activity is shown with different properties, i.e., the first includes a checkbox selected while the checkbox is not selected in the second one) but not captured in the UI Log since there is not a mouse or keyboard event over it. In order to understand this human behavior, screen captures can be obtained along with each event in the UI Log (Jimenez-Ramirez et al., 2019). Then, features can be extracted to enrich the UI Log information (Martínez-Rojas et al., 2022) as additional columns for each event (cf. Fig. 1 b). These feature extraction methods extract information regarding *all* the UI elements in the screen (e.g., position, type, or text), thus po-

[a] https://orcid.org/0000-0002-2782-9893
[b] https://orcid.org/0000-0001-8657-992X
[c] https://orcid.org/0000-0002-2631-5890
[d] https://orcid.org/0000-0001-7928-5237

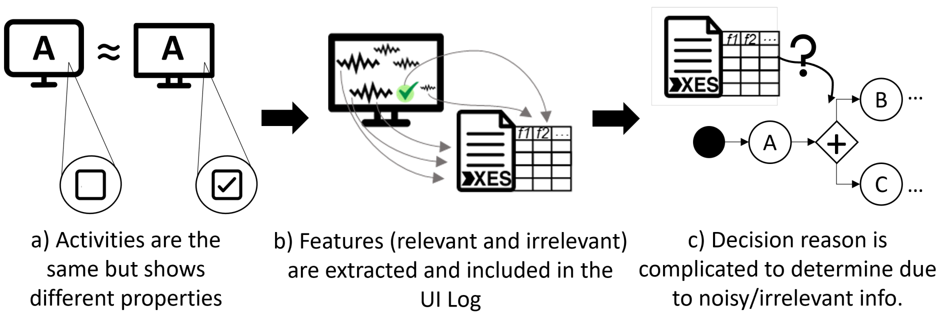

A
≈
A
f1
f2
XES
B
A
C
a) Activities are the same but shows different properties
b) Features (relevant and irrelevant) are extracted and included in the UI Log
c) Decision reason is complicated to determine due to noisy/irrelevant info.

Figure 1: Motivation problem.

tentially including the immense number of columns in the log. Moreover, only a tiny fraction of these features can explain the worker's behavior. The rest is irrelevant information. Therefore, when trying to explain this behavior, all this irrelevant information introduces noise, which negatively impacts the accuracy of the discovered models.

To address this issue, this work proposes a filtering mechanism that wipes out the process-irrelevant information from the UI Log. For this, the worker's attention is obtained using a gaze analysis tool, including this information as new events in the log. Then, only the features related to the elements that the user has looked at are those that will be considered relevant, from those present in the log.

The current paper sets the research settings and evaluates the approach over a running example. The initial results show a considerable noise reduction in such an example and point out the future research line.

The rest of the paper is organized as follows. Section 2 briefly summarizes related work. Section 3 describes the proposed approach. Section 4 presents an application example of the proposal Section 5 presents a critical discussion. Finally, Section 6 concludes the paper and describes future work.

2 RELATED WORK

To capture the attention of users through gaze analysis, there are several proposals, all of them related to eye-tracking. Most of the eye-tracking studies on business processes focus on understanding the process models (Petrusel and Mendling, 2013; Tallon et al., 2019; Zimoch et al., 2017; Duarte et al., 2020).

Other proposals focus on combining user interactions with gaze data from eye trackers. They aim to associate this information with key objects in the interface and be able to extract features to predict the intentionality, for example, to use the user's attention for the UI adaptation (Feit et al., 2020), or to study the user's behavior in a more accurate way (Loyola et al., 2015; Slanzi et al., 2017). However, these proposals are focused on the web domain, which entails that most of the tools used are limited to the browser.

On the other hand, if we focus on the context of RPA, there are proposals to identify and filter events that do not belong to any action, i.e. noise filtering (Agostinelli et al., 2020; Leno et al., 2020). But they do not propose methods to filter out non-relevant features associated with the same event.

Conversely, our proposal focuses on the use of eye tracking as a new source of information to store the worker's attention as events. These are used to filter out non-relevant features related to each event, independently of any system and application.

3 APPROACH

This proposal focuses on finding the process-relevant information based on the user's attention. For this purpose, it is assumed that relevant information will be the one the user interacts with through her gaze. In this way, to recede the noise introduced by irrelevant information, we will focus on storing only the information related to the elements observed by the user. To do so, the following steps are necessary.

First, the **user behavior is monitored** (cf. Fig. 2.a). The mouse and keyboard actions are recorded in a UI Log associated with the screenshots taken during process monitoring. This can be done using tools such as RPA-logger (López-Carnicer et al., 2020). In addition, during the recording session, the activity at the gaze level is recorded through tools for monitoring the user's gaze (i.e., eye-trackers), such as Tobii Pro Lab[1]. They provide a gaze log with the X and Y coordinates where the user looks at every process moment.

[1]Source: https://www.tobiipro.com/product-listing/tobii-pro-lab/

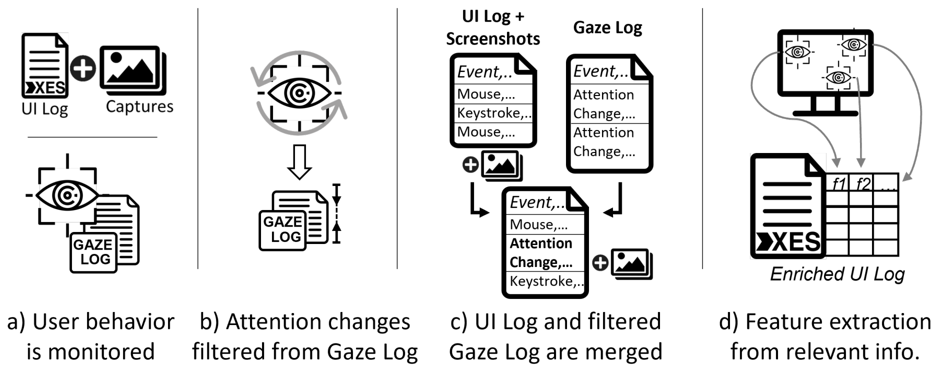


Figure 2: Approach.

Algorithm 1: Filtering Gaze Log.

```
1 fontsize= **Input** : GazeLog $gazeLog$, int $thresholdDistance$
  **Output:** GazeLog $filteredGazeLog$
2 $GazeLog\ filteredGazeLog \leftarrow copyGazeLog(gazeLog)$ $double\ lastGazeX = 0$ $double\ lastGazeY = 0$ **for**
   $event\ in\ filteredGL$ **do**
3     $double\ distanceToTheLast \leftarrow distanceBetween([event.gazeX, event.gazeY], [lastGazeX, lastGazeY])$ **if**
       $thresholdDistance \geq distanceToTheLast$ **then**
4         $filteredGazeLog.remove(event)$
5     **end**
6     $lastGazeX = event.gazeX$ $lastGazeY = event.gazeY$
7 **end**
8 **return** $filteredGazeLog$
```

In second place, **attention changes are filtered from gaze log** (cf. Fig. 2.b). A threshold is set to determine the percentage of screen resolution that must be present between one coordinate and the next for considered an attention change. This means that, given two coordinates that exceed this distance, it will be considered that the gaze has changed its focus and, therefore, exists an attention change. This process is detailed in Algorithm 1. As a result, the gaze log is converted into a filtered gaze log with only the coordinates of the attention changes.

Continuing in third place, the **UI Log and filtered gaze log are merged** (cf. Fig. 2.c) to obtain a User Behaviour Log (cf. Definition 3.1) as a result. This means that it extends the information stored in the UI Log, adding a new event type, the *attention change* event. To obtain this *UB Log*, the procedure detailed in Alg. 2 is followed. Here, attention change events are combined with the UI Log ones, sorted by their timestamp. These events will have the same general properties (i.e. screenshot, nameapp, etc.) as the immediately following UI Log event in terms of timestamp. [2] Thus, a log is obtained with information of the keyboard and mouse events, as well as screenshots and the user's attention change events, sorted by timestamp. This log will be called hereafter User Behavior Log, i.e. *UB Log*.

Definition 3.1. An **User Behaviour Log** (UB Log) is a log that stores *the aggregate of the responses, reactions, or movements made by an user interacting with an information system.* [3].

Finally, the information stored in the *UB Log* is used to perform a **feature extraction of the relevant information** (cf. Fig. 2.d). That is, from all the possible features that can be extracted from the screenshots, only those to which the user has paid attention are considered. Obtaining an a result an enriched log with the process-relevant information.

How the features are extracted depends on the objective of the research. For instance, these logs can be used to apply task mining techniques or decision model discovery (Martínez-Rojas et al., 2022). In this case the features extracted consist of the number of UI elements of each type, present in the screenshot. So those whose position does not receive attention will not be counted.

[2]This is due to the way screenshots are taken. They are taken just before the event is processed. So, the event occurs on the screenshot itself. Attention change events, do not generate screenshots. Therefore, its associated capture will be that of the subsequent event, that is the next one to be captured.

[3]Based on *behaviour* definition: https://www.wordnik.com/words/behaviour

Algorithm 2: Building User Behaviour Log from Filtered Gaze Log and UI Log.

```
   Input  : UILog uiLog, GazeLog $filteredGazeLog$
   Output: UserBehaviourLog result
1  UBLog ubLog ← copyAsUBlog(uiLog)
2  for gEvent in filteredGL do
3      $UILogEvent\ uiLogEvent,\ int\ position \leftarrow eventTimestampGreaterThan(uiLog, gEvent.timestamp)$
4      $uiLogEvent.removeProperties([keystroke, mouseX, mouseY, eventType])$
5      $uiLog.addProperties(\{'EventType' : 'AttentionalChange',$
         $'GazeX' : gEvent.gazeX,\ 'GazeY' : gEvent.gazeY\})$
6      $ubLog.setEventPreceding(position, uiLogEvent)$
7  end
8  return ubLog
```

4 APPLICATION EXAMPLE

This section details the application of the proposal to a use case based on the mortgage approval process in the context of a banking company. This proof of concept consists of an artificial business case selected simply to illustrate the gaze analysis approach.

Fig. 3 shows the main screenshots of the process. In this process, the user checks the list of mortgage applications (Activity A, not present in Fig. 3) and navigates to one of the pending applications. Here, the user must validate whether the customer's financial information is consistent with the amount requested (cf. Act.B, Fig. 3). Specifically, the percentage of income relative to the customer's debt (DTI) must be below a given threshold. If it is below, the mortgage application is pre-approved (cf. Act.C, Fig. 3). If it exceeds the threshold, the mortgage application is rejected, and an email is sent to the client indicating the reason (cf. Act.D, Fig. 3). The user repeats this process as many times a day as the mortgage applications she has in the queue.

The UI and the Gaze Logs are obtained after the execution of the first part of this proposal (cf. Fig.2.a). These logs consist of the user interactions at mouse and keyboard level on the one hand and at gaze level on the other hand.

In the Gaze Log, it can be observed that there are many events whose coordinates are located in the same area of the screen (cf. highlighted groups in Fig.5). In this example, a threshold of 5% of the resolution is set, so assuming a resolution of 1920x1080, we have a threshold distance of 96 pixels on the X-axis and 54 on the Y. Therefore, it can be determined that all distances greater than this are considered an attention change. As a result, the gaze log is filtered, remaining only attention changes events as shown in Fig. 5.

Then, following the procedure of Alg. 2, the UI and the Gaze Logs are merged to give rise to the UB Log. An extract of this UB Log is shown in Fig. 6. It can be observed that all the events of attention changes take the same *nameapp* and *screenshot* as the immediately following event since attention changes take place in this screenshot.

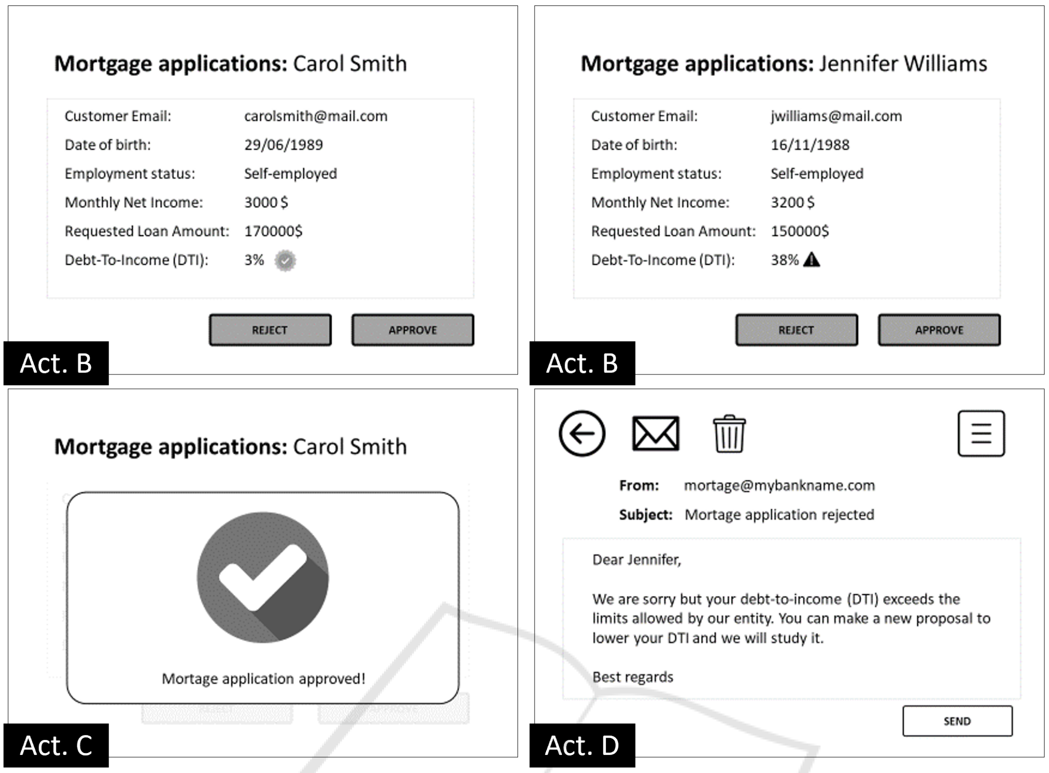
Using this UB Log, filtering can be performed. Some of the existing techniques to crop and detect components, such as (Xu et al., 2017; Moran et al., 2018) can be used. So, we can extract the information of the bounding box associated with each component in the screenshot and the type of UI element (e.g., checkbox or button) to which it corresponds. These are helpful to cross the data with the gaze information of the UB Log. Only if there is a point (i.e., [*GazePointX, GazePointY*]) inside the bounding box associated to the UI element, features are extracted from it. In this case, if we focus on the screenshot *2_img.png*, it can be seen how crossing the data shows those UI elements that the user has checked (cf. Fig. 7).

Subsequently, a feature extraction technique has to be selected to be applied to extract information from the fields *"Customer Email"*, *"29/06/1989"*, *"\$170000"* and *"3%"* followed by a checked. Which are those that are considered relevant by this approach.

Finally, using all data present in the UB Log, it is possible to conclude that having a percentage higher than 35%, the application is rejected. Therefore, the reason for the decision taken is discovered, proving to be a promising approach.

5 DISCUSSION

This study establishes the basis for developing a method to improve the way of retrieving information from the user human-computer interaction, including



Mortgage applications: Carol Smith

Customer Email:	carolsmith@mail.com
Date of birth:	29/06/1989
Employment status:	Self-employed
Monthly Net Income:	3000 $
Requested Loan Amount:	170000$
Debt-To-Income (DTI):	3%

Mortgage applications: Jennifer Williams

Customer Email:	jwilliams@mail.com
Date of birth:	16/11/1988
Employment status:	Self-employed
Monthly Net Income:	3200 $
Requested Loan Amount:	150000$
Debt-To-Income (DTI):	38%

Mortgage applications: Carol Smith

Mortage application approved!

From: mortage@mybankname.com

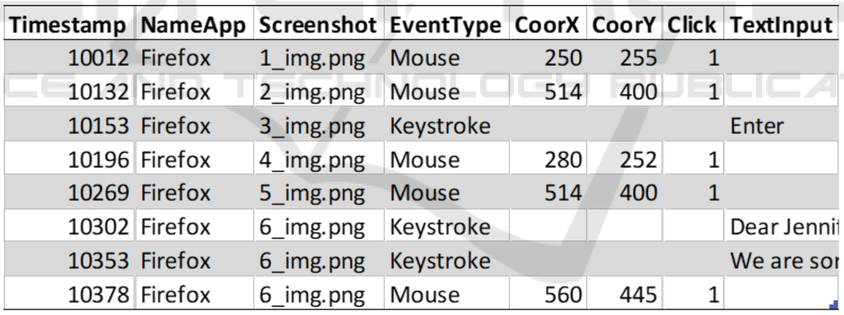
Subject: Mortage application rejected

Dear Jennifer,

We are sorry but your debt-to-income (DTI) exceeds the limits allowed by our entity. You can make a new proposal to lower your DTI and we will study it.

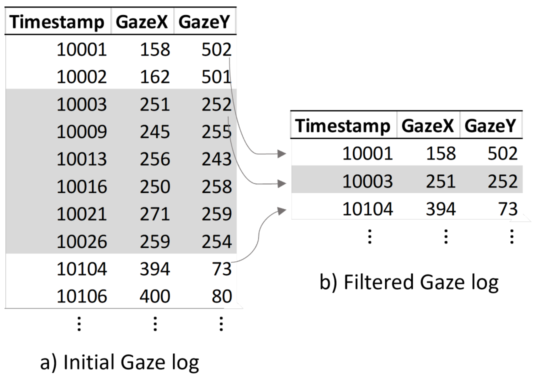
Best regards

Figure 3: Screenshots associated with the example process.



Timestamp	NameApp	Screenshot	EventType	CoorX	CoorY	Click	TextInput
10012	Firefox	1_img.png	Mouse	250	255	1	
10132	Firefox	2_img.png	Mouse	514	400	1	
10153	Firefox	3_img.png	Keystroke				Enter
10196	Firefox	4_img.png	Mouse	280	252	1	
10269	Firefox	5_img.png	Mouse	514	400	1	
10302	Firefox	6_img.png	Keystroke				Dear Jenni
10353	Firefox	6_img.png	Keystroke				We are sor
10378	Firefox	6_img.png	Mouse	560	445	1	

Figure 4: Excerpt of a UI Log obtained form a keylogger.



Timestamp	GazeX	GazeY
10001	158	502
10002	162	501
10003	251	252
10009	245	255
10013	256	243
10016	250	258
10021	271	259
10026	259	254
10104	394	73
10106	400	80
⋮	⋮	⋮

a) Initial Gaze log

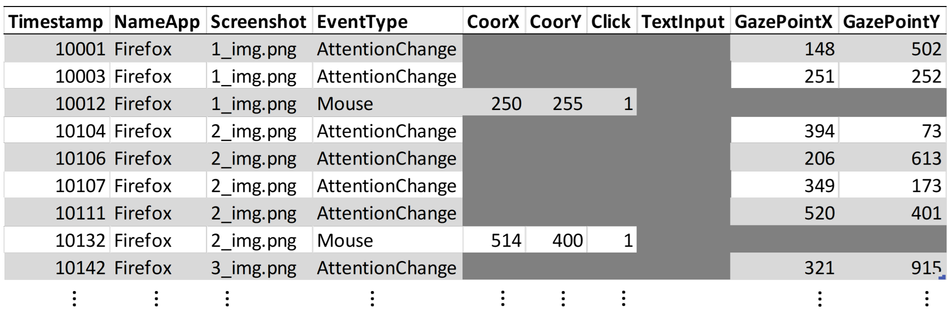
Timestamp	GazeX	GazeY
10001	158	502
10003	251	252
10104	394	73
⋮	⋮	⋮

b) Filtered Gaze log

Figure 5: Exceprt of Gaze Log obtained from an eyetracker.

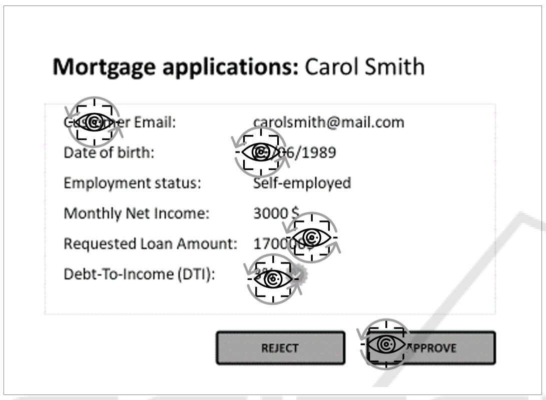
a gaze analyzer. The main goal of this approach is to discard irrelevant information to produce a higher-quality UB Log to be later processed.

There are several ways to store the information provided by the gaze analyzer. One of them could be to store this information as new columns of the UB Log. In this way, we could store each event change (i.e., gaze attention) in a new column, similar to a mouse click coordinates or text input in a keystroke event. Through this alternative, it is possible to obtain better performance indicators when executing the flattening of the UB Log. However, relevant information associated with the timestamp of each event would be lost (e.g., the order in which events occur). To mit-



Timestamp	NameApp	Screenshot	EventType	CoorX	CoorY	Click	TextInput	GazePointX	GazePointY
10001	Firefox	1_img.png	AttentionChange					148	502
10003	Firefox	1_img.png	AttentionChange					251	252
10012	Firefox	1_img.png	Mouse	250	255	1			
10104	Firefox	2_img.png	AttentionChange					394	73
10106	Firefox	2_img.png	AttentionChange					206	613
10107	Firefox	2_img.png	AttentionChange					349	173
10111	Firefox	2_img.png	AttentionChange					520	401
10132	Firefox	2_img.png	Mouse	514	400	1			
10142	Firefox	3_img.png	AttentionChange					321	915
⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮	⋮

Figure 6: User Behaviuor Log (UB Log) associated to example process.



Mortgage applications: Carol Smith

Customer Email: carolsmith@mail.com
Date of birth: 06/1989
Employment status: Self-employed
Monthly Net Income: 3000 $
Requested Loan Amount: 1700
Debt-To-Income (DTI):

Figure 7: Attention changes highlighted on screenshot.

igate this threat, this solution proposes to store the attention change events as a new event type (i.e., attention change), including as many rows as you need when generating the UI Log.

Due to the incipient nature of the proposal, this solution does not consider anomalous user behavior during the recording session that results in the gaze log. In this sense, this approach delegates gaze analysis tools to detect these most common behaviors (e.g., pupil dilation due to ambient light change or operator fatigue effect).

Moreover, the extraction of features from the UB Log can be done in several ways, not only as indicated in this proposal. For instance, it would be possible to extract plain text or features from the exact position of each type of UI element on the screen to store them as columns of the log. Our proposal is applicable to any of the approaches, since the features can be filtered as long as we can determine if the position of the UI element from which it comes coincides with an attention change event.

Finally, the most important threat we have detected in the development of our study is its validation. So far, we have laid the foundations for the proposal and instantiated it on a simple and synthetic example. To mitigate this threat, we will propose as future work the execution of more robust experimentation to check the robustness and effectiveness of the proposal.

6 CONCLUSIONS AND FUTURE WORK

This paper presents a novel approach for generating a higher quality UI Log (i.e., UB Log) using gaze tracking techniques to filter the most relevant user interaction features. Although in the literature, there are some proposals related to this topic (Loyola et al., 2015; Slanzi et al., 2017), the major drawback we found is that they are focused on web environments. Conversely, our approach contributes to the use of any application. The proposed method has been conceptually tested in a synthetic example, obtaining promising results. The present solution, in addition, could be used for work in progress research such as the one presented in (Martínez-Rojas et al., 2022).

As further future work, we plan: (1) as exposed in the discussion, to validate the proposal on a real industry-driven use case. (2) to analyze results obtained using different values of the threshold distance to identify attentional changes. (3) to consider cases, such as the use of all-in-one applications, where attention change events can lead to an activity change. (4) to evaluate the information provided by the different eyetrackers and based on this determine if the results are improved by including some additional column to the X and Y coordinates of the gaze.

ACKNOWLEDGEMENTS

This publication is part of the project PID2019-105455GB-C31, funded by MCIN/ AEI/10.13039/501100011033/ and by the “European Union” and the project P20_00644, funded by Junta de Andalucía; Moreover, the first author of this work is currently supported by the FPU scholarship program, granted by the Spanish Ministry of Education and Vocational Training (FPU20/05984).

REFERENCES

Agostinelli, S., Lupia, M., Marrella, A., and Mecella, M. (2020). Automated generation of executable rpa scripts from user interface logs. In *International Conference on Business Process Management*, pages 116–131. Springer.

Asatiani, A. and Penttinen, E. (2016). Turning robotic process automation into commercial success - Case OpusCapita. *Journal of Information Technology Teaching Cases*, 6(2):67–74.

Capgemini, C. (2017). Robotic Process Automation - Robots conquer business processes in back offices.

Duarte, R. B., da Silveira, D. S., de Albuquerque Brito, V., and Lopes, C. S. (2020). A systematic literature review on the usage of eye-tracking in understanding process models. *Business Process Management Journal*.

Feit, A. M., Vordemann, L., Park, S., Berube, C., and Hilliges, O. (2020). Detecting relevance during decision-making from eye movements for ui adaptation. In *ACM Symposium on Eye Tracking Research and Applications*, ETRA '20 Full Papers, New York, NY, USA. Association for Computing Machinery.

Jimenez-Ramirez, A., Reijers, H. A., Barba, I., and Del Valle, C. (2019). A method to improve the early stages of the robotic process automation lifecycle. In *CAiSE 2019*, pages 446–461. Springer.

Lacity, M. and Willcocks, L. (2015). What Knowledge Workers Stand to Gain from Automation. *Harvard Business Review*.

Leno, V., Polyvyanyy, A., Dumas, M., La Rosa, M., and Maggi, F. M. (2020). Robotic Process Mining Vision and Challenges. *Business & Information Systems Engineering*.

López-Carnicer, J. M., del Valle, C., and Enríquez, J. G. (2020). Towards an opensource logger for the analysis of rpa projects. In *International Conference on Business Process Management*, pages 176–184. Springer.

Loyola, P., Martinez, G., Muñoz, K., Velásquez, J. D., Maldonado, P., and Couve, A. (2015). Combining eye tracking and pupillary dilation analysis to identify website key objects. *Neurocomputing*, 168:179–189.

Martínez-Rojas, A., Jiménez-Ramírez, A., Enríquez, J., and Reijers, H. (2022). Analyzing variable human actions for robotic process automation. In *International Conference on Business Process Management*, pages 75–90. Springer.

Mayr, A., Herm, L.-V., Wanner, J., and Janiesch, C. (2022). Applications and challenges of task mining: A literature review.

Moran, K., Bernal-Cárdenas, C., Curcio, M., Bonett, R., and Poshyvanyk, D. (2018). Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering*, 46(2):196–221.

Petrusel, R. and Mendling, J. (2013). Eye-tracking the factors of process model comprehension tasks. In *International Conference on Advanced Information Systems Engineering*, pages 224–239. Springer.

Reinkemeyer., L. (2020). *Process Mining in Action. Principles, Use Cases and Outlook*. Springer.

Slanzi, G., Balazs, J. A., and Velásquez, J. D. (2017). Combining eye tracking, pupil dilation and eeg analysis for predicting web users click intention. *Information Fusion*, 35:51–57.

Tallon, M., Winter, M., Pryss, R., Rakoczy, K., Reichert, M., Greenlee, M. W., and Frick, U. (2019). Comprehension of business process models: Insight into cognitive strategies via eye tracking. *Expert Systems with Applications*, 136:145–158.

van der Aalst, W. M. P. (2016). *Process mining: data science in action*. Springer, Heidelberg.

Xu, Z., Baojie, X., and Guoxin, W. (2017). Canny edge detection based on open cv. In *2017 13th ICEMI*, pages 53–56.

Zimoch, M., Pryss, R., Schobel, J., and Reichert, M. (2017). Eye tracking experiments on process model comprehension: Lessons learned. In Reinhartz-Berger, I., Gulden, J., Nurcan, S., Guédria, W., and Bera, P., editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 153–168, Cham. Springer International Publishing.