Analyzing Variable Human Actions for Robotic Process Automation

A. Martínez-Rojas[1](✉), A. Jiménez-Ramírez[1], J. G. Enríquez[1], and H. A. Reijers[2]

[1] Departamento de Lenguajes y Sistemas Informáticos, Escuela Técnica Superior de Ingeniería Informática, Avenida Reina Mercedes, s/n, 410121 Sevilla, Spain
{amrojas,ajramirez,jgenriquez}@us.es

[2] Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, Utrecht 3584 CC, The Netherlands
h.a.reijers@uu.nl

Abstract. Robotic Process Automation (RPA) provides a means to automate mundane and repetitive human tasks. Task Mining approaches can be used to discover the actions that humans take to carry out a particular task. A weakness of such approaches, however, is that they cannot deal well with humans who carry out the same task differently for different cases according to some hidden rule. The logs that are used for Task Mining generally do not contain sufficient data to distinguish the exact drivers behind this variability. In this paper, we propose a new Task Mining framework that has been designed to support engineers who wish to apply RPA to a task that is subject to variable human actions. This framework extracts features from User Interface (UI) Logs that are extended with a new source of data, namely screen captures. The framework invokes Supervised Machine Learning algorithms to generate decision models, which characterize the decisions behind variable human actions in a machine-and-human-readable form. We evaluated the proposed Task Mining framework with a set of synthetic UI Logs. Despite the use of only relatively small logs, our results demonstrate that a high accuracy is generally achieved.

Keywords: Robotic Process Automation · Process discovery · Task mining · Decision model discovery

1 Introduction

Robotic Process Automation (RPA) is a software technology that facilitates the automation of human tasks, especially when they are structured and repetitive.

This research has been supported by the Spanish Ministry of Science, Innovation and Universities under the NICO project (PID2019-105455GB-C31) and the Centro para el Desarrollo Tecnológico Industrial (CDTI) of Spain under the CODICE project (EXP 00130458/IDI-20210319 - P018-20/E09) and by the FPU scholarship program, granted by the Spanish Ministry of Education and Vocational Training (FPU20/05984).

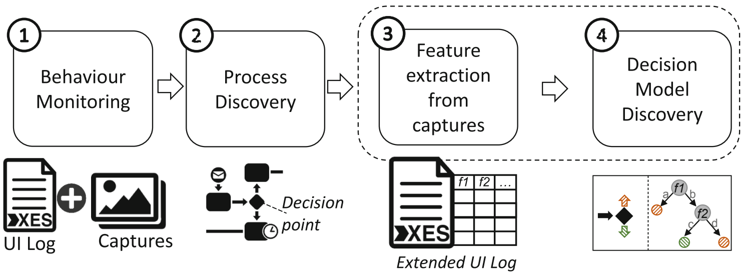


Fig. 1. Proposed framework for variability analysis through interpretable decisions from UI Logs. The current paper focuses on steps 3 and 4.

In contrast to other automation approaches (i.e., API-based), RPA works by closely mimicking the way humans interact with computer applications [2].

Some of the typical benefits are that the technology helps to save costs, increases agility, and improves quality [9,18] while its level of intrusiveness is low [33]. Due to these wide range of benefits, industry has adopted this technology on a wide scale in recent years [10].

Most RPA projects start out by observing how human workers perform work that is to be automated. To support this initial RPA analysis, approaches such as Task Mining [1,31] and Robotic Process Mining [22] are highly suitable. What all these techniques have in common is that they operate on UI Logs, i.e., a series of timestamped events (e.g., mouse clicks and keystrokes), obtained by monitoring and recording user interfaces.

An open issue concerning discovering the process model behind the UI Log is to disclose the drivers behind the variations that are shown in a process model. These variations indicate that human operators take different decisions for different cases, but it is generally not possible to find rules which explain how these decisions were made. That, however, is crucial knowledge for the engineer who aims to develop the RPA bot. A clear example occurs in the development of a business process outsourcing operation, where it is necessary to work with different systems that, in their turn, are virtualized. In this type of scenario, it is difficult to understand certain operator decisions that depend specifically on the context of the problem being addressed. Although efforts have been made to overcome this problem, they rely on what is observable in the log [3,12,19,32]. This approach has inherent limitations since human work is sometimes based on information that is simply not captured in the log, i.e., it only appears on the screen. Therefore, human decision-making remains *hidden* in this respect and, as a result, difficult to automate.

Against this backdrop, this paper proposes a framework to analyze the variability in human actions automatically. The framework leverages information on the screen to detect factors that influence human decisions – an angle that existing approaches have neglected so far.

In previous work, we proposed (1) a tool to monitor the user behavior which generates a UI Log, including one screen capture for each event (cf. step 1 in Fig. 1) [25], and (2) a method to analyze such logs to discover the underlying process model (cf. step 2 in Fig. 1) [15]. This paper significantly extends these contributions by: (1) proposing a novel approach to systematically analyze the screen captures to extract information which is, then, incorporated into the UI Logs (cf. step 3 in Fig. 1) using image-processing techniques [26] (e.g., Optical Character Recognition), (2) presenting a method to discover decision models which explain the variability that is found in the UI Log (cf. step 4 in Fig. 1) using a Machine Learning (ML) approach, and (3) evaluating the approach with synthetics problems of different complexity to demonstrate the efficacy and efficiency of the framework.

It should be noted that industrial RPA platforms often do incorporate sophisticated task mining techniques, as well as features for image processing. The point is that these capabilities are not integrated in approaches to analyze task variability.

The rest of the paper is organized as follows. Section 2 provides the background in topics like behavior monitoring, ML, and image processing. Section 3 introduces a synthetic business case to motivate the proposal. Section 4 elaborates on the novel method to discover the decision models. Section 5 reports the empirical evaluation performed to validate the method. Sect. 6 presents a critical discussion. Section 7 reviews similar approaches in the literature. Finally, Sect. 8 summarizes the work and describes future research lines.

2 Background

The approach that is presented in this paper builds on behavior monitoring techniques, process discovery, Graphical User Interface (GUI) analysis, and ML.

For *behavioral monitoring*, there are several industrial solutions for keylogging[1] that capture the interaction of a human interacting with a system. In addition, other approaches have been proposed in academia, taking a further step in how to automate certain stages of robotization [3,11,20,24,25]. It should be noted that there are different formats proposed for capturing events, although the most representative for this work is the *UI Log* from [15] which defines it as an extension of the XES format—standard for event logs in Process Mining—which incorporate attributes like the *app_name* (i.e., the name of the app), *event_type* (i.e., mouse click or keystroke), *click_type* (i.e., left, right, or middle), *click_coords* (i.e., position of the mouse on the screen), the *keystroke* (i.e., the keys that are typed), and the *screenshot* (i.e., the screen capture associated to this event path).

Using a UI Log, many proposals exist for *process discovery*, i.e., to automatically or semi-automatically discover the underlying process model that is associated with human behavior [5,6,13,15,19,23]. Moreover, these proposals include functionalities to clean the UI Log from irrelevant information, so that noise in the resulting process model is filtered; and to select variants/cases/activities

[1] Availabe at: www.spyrix.com and bestxsoftware.com/es/.

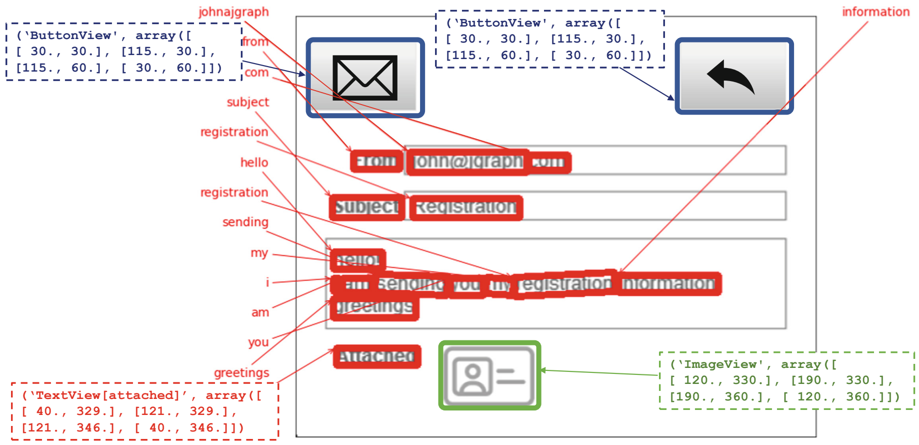
according to the frequency, length, and other criteria that are useful to identify process candidates to robotize. The resulting process model may contain decision points and separate branches for different process variants.

In the field of *GUI analysis*, approaches exist than can identify the GUI components within an image [28,34]. GUI components are atomic graphical elements with predefined functionality, displayed within a GUI of a software application [28]. Besides locating the element on the screen (i.e., calculating the bounding boxes), they can classify them by the type of element, e.g., image, button, or text. Nevertheless, when dealing with texts, Optical Character Recognition (OCR) techniques are more appropriate. For instance, KerasOCR [17] allows extracting words and their bounding boxes from screen captures. Such an image-based technique allows for extracting the content and structure of an image in a computer-readable way (cf. Fig. 2).

In the *ML* domain, supervised algorithms exist that focus on finding rules to explain a given dataset, e.g., extracting a decision tree [27]. Datasets are commonly represented in tabular form: each column is an input variable or a label (i.e., what needs to be predicted or classified), and each row is a member of the dataset. In a classification problem, the algorithm tries to find patterns in the input variables that help to explain the labels. Decision trees are an example of classification algorithms that, besides just providing a classification, do so in a human-interpretable way [14].

3 Running Example

This section describes an artificial business case to explain and motivate the problem addressed in this paper. Figure 3 depicts an excerpt of the elementary user interfaces for registering a customer in the context of a telecom company.



```
('ButtonView', array([
[ 30., 30.], [115., 30.],
[115., 60.], [ 30., 60.]])
```

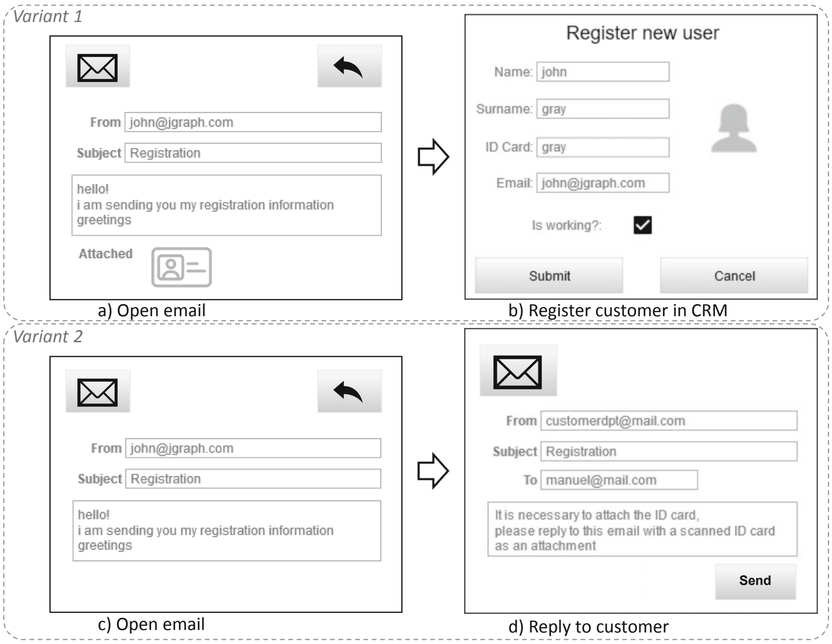
```
('ButtonView', array([
[ 30., 30.], [115., 30.],
[115., 60.], [ 30., 60.]])
```

```
('TextView[attached]', array([
[ 40., 329.], [121., 329.],
[121., 346.], [ 40., 346.]])
```

```
('ImageView', array([
[ 120., 330.], [190., 330.],
[190., 360.], [ 120., 360.]])
```

Fig. 2. Example of GUI analysis applied to a sample screen capture. Extracted buttons are in blue, images are in green, and texts are in red. To the sake of readability, some extracted coordinate are not shown. (Color figure online)

In this registration process, the human operator checks her email inbox and reviews the pending emails regarding the registration tasks. After opening the email, the operator has to validate that all provided information related to the new customer is correct. More precisely, the customer *ID card* is expected to be included as an attachment. If it is indeed included (cf. Fig. 3a), all the customer data has to be registered into a CRM system (cf. Fig. 3b). Otherwise, in case the *ID card* is missing (cf. Fig. 3c), an email has to be sent to the customer requesting such data (cf. Fig. 3d). Regardless which of these two situations occurred (i.e., Variant 1 or Variant 2 of Fig. 3), the operator returns to their inbox to process the next mail. This process must be repeated several times during the day to process the entire queue of emails in the operator's inbox.



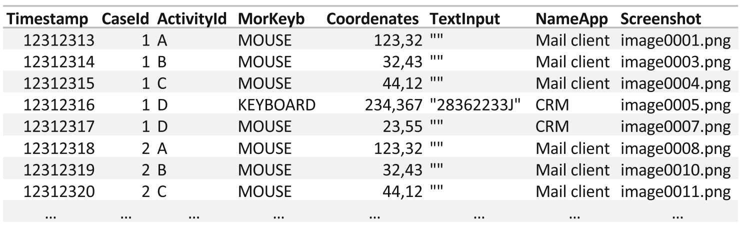
From: john@jgraph.com
Subject: Registration
hello!
i am sending you my registration information
greetings

Register new user
Name: john
Surname: gray
ID Card: gray
Email: john@jgraph.com

From: john@jgraph.com
Subject: Registration
hello!
i am sending you my registration information
greetings

From: customerdpt@mail.com
Subject: Registration
To: manuel@mail.com
It is necessary to attach the ID card, please reply to this email with a scanned ID card as an attachment

Fig. 3. Mockups of motivating example



Timestamp	CaseId	ActivityId	MorKeyb	Coordenates	TextInput	NameApp	Screenshot
12312313	1	A	MOUSE	123,32	""	Mail client	image0001.png
12312314	1	B	MOUSE	32,43	""	Mail client	image0003.png
12312315	1	C	MOUSE	44,12	""	Mail client	image0004.png
12312316	1	D	KEYBOARD	234,367	"28362233J"	CRM	image0005.png
12312317	1	D	MOUSE	23,55	""	CRM	image0007.png
12312318	2	A	MOUSE	123,32	""	Mail client	image0008.png
12312319	2	B	MOUSE	32,43	""	Mail client	image0010.png
12312320	2	C	MOUSE	44,12	""	Mail client	image0011.png
...	...	...	...	...	...	...	...

Fig. 4. Excerpt of a UI Log obtained form a keylogger.

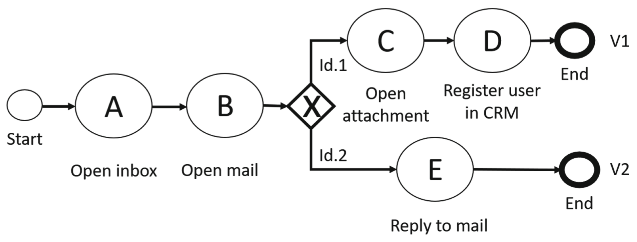


Fig. 5. Process model discovered from the UI Log.

Task Mining techniques can be applied to discover the operator process model [3,15,22]. In essence, these techniques start by monitoring the operator behavior (e.g., with a key logger [25]) and, then, analyze it to extract the relevant activities and cases using image-similarity comparison [15]. This results in a *UI Log* (cf. Fig. 4) which includes, at a minimum, the time when each event is produced, the activityId, the caseId, the type of event (i.e., mouse click or keystroke), the text which is introduced, the name of the application where the event has occurred, and a screen capture taken just before the event. The process model discovered from such a log would be similar to the one shown in Fig. 5, which correctly depicts that it contains a single decision point after activity "B" (i.e., after seeing the email), where the process branches off into two different variants.

Despite the simplicity of this process, disclosing the condition which rules the decision point of this process is challenging, i.e., *why* is it that the operator choses for decision *Id1* or *Id2*? Existing techniques are unable to find a meaningful correlation between the events and the decision since it is missing from the UI Log – it only appears in the screen captures.

4 Decision Discovery Framework

It is clearly challenging to automatically identify the factors behind variable human behavior on the basis of a UI Log that lacks certain key information. The proposed framework includes a first step to enrich the UI Log with features extracted from the screen captures, recorded along with the UI Log (cf. Sect. 4.1). The second step that is carried out by the framework leverages this new information, which is derived from the screen captures, to deliver a decision model. In our opinion, it is crucial that such a model can be interpreted by a machine and also be understood by a human (cf Sect. 4.2).

4.1 Feature Extraction

This step transforms the screen captures that are taken for each event in the UI Log into structured information (i.e., features), which can be incorporated back into the log. Since a variety of features can be extracted from a screen capture, the current framework offers a common interface for these extractors (cf. Definition 1) which can be implemented accordingly to the project necessities.

Definition 1. *A **Feature Extractor** is a tuple* $<Name, Function>$ *that represents a software component named* $Name$*, with a* $Function$ *that receives an image and returns a list of pairs* $<key, value>$*, where the key is the name of a feature that presents a value in the given image.*

Once implemented, the framework applies these feature extractors to each event in the UI Log, i.e., the *Function* of each extractor is applied to the screen capture of each event. If the feature extracted is new in the UI Log (i.e., none of the columns of the UI Log have the same name as the *key*), a new column is appended to the log, and the *value* is assigned to this event. The rest of the log events have an empty value for this new column. Otherwise, if the feature already exists, the *value* is assigned to this event at the existing column *key*.

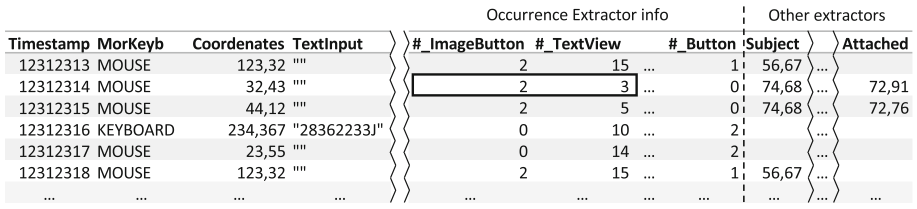
To illustrate this process, we will describe the **UI Element Occurrence** extractor in more detail. It extracts the occurrences of UI elements in the sense that the output of its *Function* contains as many *keys* as different UI elements are found in the screen capture. The *value* associated with each *key* is a number greater than 0, which expresses the number of occurrences of this *key* in the screen capture. The extractor includes the detection and classification of each component to determine which type of UI Element they belong to. For this purpose, each screen capture is processed in three phases by the *Function*:

1. To detect the UI elements, image-processing techniques are applied to find the elements within the image. In this phase, edge detection algorithms can be used, such as Canny's algorithm [7], which we applied. Each detected UI element is then cropped to deal with these separately in the next phase.
2. To classify the detected UI elements according to the type of GUI component they belong to, an ML model—previously trained for conducting such a task—is used.[2] Specifically, we adapted the convolutional neural network proposed in Moran's work [28], which is able to detect 14 different types of UI elements.
3. To return the number of occurrences for each type of UI element, the detected UI elements in the first phase are first grouped, according to the class determined during the second phase, and then summed.

Consequently, after applying the UI Element Occurrence extractor, 14 columns—one for each type of UI element—are appended to the resulting UI Log. They are then available, along with the additional columns from other extractors.

Running Example. When considering the screen capture in Fig. 3.a, which relates to our running example, 2 image buttons can be observed (i.e., the 'envelope' for returning to the email home page, and the 'arrow' for reply), as well as 3 texts. Therefore, the returning list of the *Function* of the UI Element Occurrence extractor is: $\{<\#\_ImageButton, 2>, <\#\_TextView, 3>\}$. This results in an UI Log that includes the $\#\_ImageButton$ column, which gets value 2, and the $\#\_TextView$ column, which gets value 3. The values are shown in Fig. 6 under the header 'Occurrence Extractor info'.

[2] We trained the model with this dataset: https://doi.org/10.5281/zenodo.2530277.



				Occurrence Extractor info				Other extractors		
Timestamp	MorKeyb	Coordenates	TextInput	#_ImageButton	#_TextView		#_Button	Subject		Attached
12312313	MOUSE	123,32	""	2	15	...	1	56,67	...	
12312314	MOUSE	32,43	""	2	3	...	0	74,68	...	72,91
12312315	MOUSE	44,12	""	2	5	...	0	74,68	...	72,76
12312316	KEYBOARD	234,367	"28362233J"	0	10	...	2		...	
12312317	MOUSE	23,55	""	0	14	...	2		...	
12312318	MOUSE	123,32	""	2	15	...	1	56,67	...	
...	...	...	...	...	...	...	...	...	...	...

Fig. 6. A sample UI Log including features (columns) extracted from the screen captures.

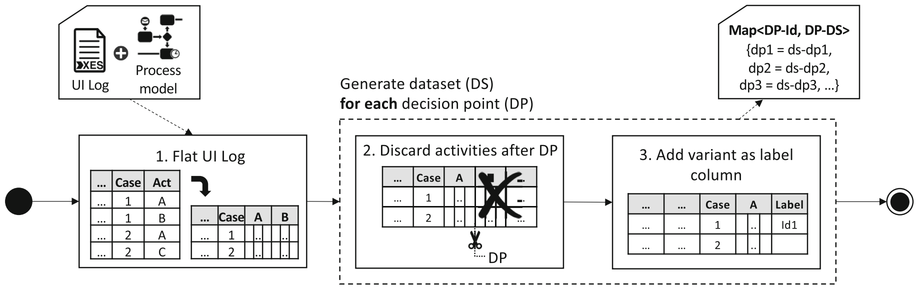


Fig. 7. Proposed algorithm for generating labeled datasets

4.2 Decision Model Discovery

Once the UI Log is enriched with the extracted features, the decision model discovery takes place. To enable this, first a labeled dataset is generated for each decision point in the process model. Secondly, a classification algorithm is applied to disclose the rules existing in the dataset, which will help to explain the decision in the process model. In the remainder of this section, each of these steps will be explained.

Generating the Labeled Dataset. This step processes the enriched UI Log to convert it into a labeled dataset usable by supervised ML algorithms (e.g., classification trees) in the next step. Specifically, one dataset is created for each decision point that appears in the process model.

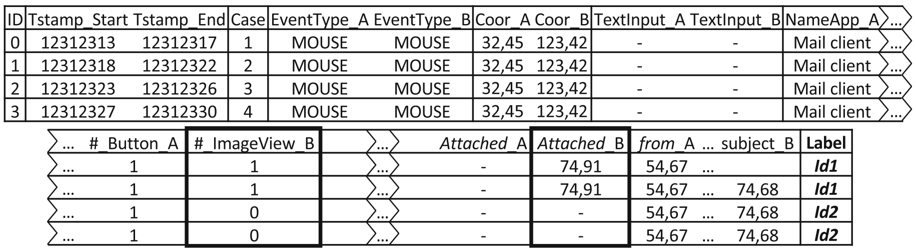
Given a decision point, the objective is to determine which branch is chosen, providing a detailed explanation. Therefore, the *label* of the resulting dataset is the branch of the decision point. With this aim, Fig. 7 illustrates, in an activity diagram, the way of creating an appropriate dataset for each decision point that appears in a process model.

First, it receives both the UI Log and the discovered process model and returns a map where the keys are the *IDs* of the decision point in the process model, and the values are the *dataset* extracted to each one. The algorithm starts by flattening the UI Log (cf. step 1 in Fig. 7). This operation is done by putting all UI Log events of each case in the same row of the dataset (i.e., the dataset will contain one row for each different case in the UI Log). In this way, the dataset will include columns for an *ID* (i.e., auto-incremental number starting in 0), the *caseID*, a *TimeStampEnd* (i.e., corresponding to the timestamp of the last event of this case), and a *TimeStampStart* (i.e., corresponding to the timestamp of the first event of this case), and the rest of the UI Log attributes for each activity of this case (i.e., event type, click coordinates, each of the extracted features, etc.). Regarding these latter attributes, the column names are the attribute names prefixing its *activityID*. For instance, the EventType attribute of activity A will be stored as: *EventType_A*.

Then, for each decision point in the process model, the columns of the dataset are filtered in such a way that only the columns related to the activities that precede that decision point are kept (cf. step 2 in Fig. 7). Note that the columns of the events *after* the decision point relate to events in the *future*, so they should not be considered when discovering the decision model of that particular decision. Aferwards, the *label* column is added to the dataset. For each row (i.e., each process case), its value is the branch which is taken for this decision point, (cf. step 3 in Fig. 7).

Discovering the Decision Model with Classification Algorithms. As we explained, the labeled dataset generated at this point will be used to train a supervised ML model. This model will classify the *label* column based on the rest of the dataset columns. There is a wide range of algorithms that can be used for this purpose. For our framework, we use decision trees since both humans and machines can easily interpret them. This kind of model expresses the discovered rules of the classification in the form of a tree: the tree nodes are a column of the dataset (i.e., UI Log attributes) while the tree edges are non-overlapping conditions as evaluated over the node. Our framework implements four common used algorithms to construct decision trees: CART [8], ID3 [29], C4.5 [30] and CHAID [16]. Although these techniques are known to be similar, they are implemented to explore their behavior in this context.

To make the tree even more understable for humans, the framework takes the discovered features and rules to **highlight** them into the associated screen captures, in this way linking them with their visual information.



ID	Tstamp_Start	Tstamp_End	Case	EventType_A	EventType_B	Coor_A	Coor_B	TextInput_A	TextInput_B	NameApp_A	...
0	12312313	12312317	1	MOUSE	MOUSE	32,45	123,42	-	-	Mail client	...
1	12312318	12312322	2	MOUSE	MOUSE	32,45	123,42	-	-	Mail client	...
2	12312323	12312326	3	MOUSE	MOUSE	32,45	123,42	-	-	Mail client	...
3	12312327	12312330	4	MOUSE	MOUSE	32,45	123,42	-	-	Mail client	...

...	#_Button_A	#_ImageView_B	...	*Attached*_A	*Attached*_B	*from*_A	...	subject_B	Label
...	1	1	...	-	74,91	54,67	...		***Id1***
...	1	1	...	-	74,91	54,67	...	74,68	***Id1***
...	1	0	...	-	-	54,67	...	74,68	***Id2***
...	1	0	...	-	-	54,67	...	74,68	***Id2***

Fig. 8. Dataset extracted from the UI Log of Fig. 4

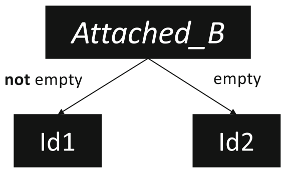


Fig. 9. Decision Tree for the decision point in the process model of Fig. 5

Running Example. In our running example, only one decision point is discovered from the UI Log (cf. Fig. 5). Therefore, only one dataset is generated (cf. Fig. 8). The dataset contains a series of columns that describe the attributes for each case and a *label* column that indicates the decision which is made at the decision point. The decision tree that is obtained when trained on this dataset is shown in Fig. 9. As can be inferred from the framed columns in Fig. 8, the decision *Id2* is made when there is nothing in column *Attached_B*; otherwise, the decision *Id1* is taken. When looking at the screen capture of activity B (cf. Fig. 3a and 3c), we can see that these actions correspond to the absence or presence of the word "Attachment" in the email, respectively. Note that the classification algorithm provides one tree, although other alternatives exist. For instance, the number of ImageView UI elements in activity B explains the behavior too.

5 Empirical Evaluation

Purpose: This empirical evaluation aims to analyze our proposal to discover the conditions that drive decisions from a UI Log. Particularly, it focuses on situations where the conditions depend on information that does appear on the screen but not in the log itself.

Objects: The evaluation is based on a set of synthetic problems, which resemble realistic use cases in the administrative domain. More precisely, 3 different processes (P) are created, each of them with a different level of complexity. Complexity is measured in terms of the number of activities, the number of variants to execute the process, and the number of visual features that affect the decision to choose between variants. The processes are:

- P1 Client creation. A process with 5 activities and 2 variants. The single decision in this process is made based on the existence of an attachment in the reception email.
- P2 Client validation. A process with 7 activities and 2 variants. The decision is made based on the user's response to a query.
- P3 Client deletion. A process with 7 activities and 4 variants. The decisions are made based on two conditions: (1) the existence of pending invoices and (2) the existence of an attachment to justify the payment of the invoices.

These processes all contain a single decision point, although the one in P3 is rather complex. All processes include (1) synthetic screen captures for their activities and (2) a sample event log with a single instance for each variant. To generate the objects for the evaluation, we generate event logs of different sizes ($|L|$) for each of these processes by deriving events from the sample event log. We consider log sizes in the range of $\{10, 25, 50, 100\}$ events. Note that we consider complete instances in the log and, thus, we remove the last instance if it goes beyond $|L|$. Some of these logs are generated with a balanced number of instances, while others are unbalanced ($B?$) in the sense that more than a 20% frequency difference exists between the most frequent and less frequent variants. To average the result over a collection of problems, 30 instances are randomly generated for each tuple $<P, |L|, B?>$.[3]

Independent Variables: The independent variables of this empirical evaluation are (1) the process (i.e., P), (2) the log size (i.e., $|L|$), and (3) whether the log is balanced or not (i.e., $B?$).

Response Variables: The efficiency and efficacy of the approach are evaluated in terms of: (1) the average time spent on each of the framework phases, i.e., feature extraction (tFE) and decision model discovery (tDD), (2) the number of columns included in the log after the feature extraction phase ($\#CL$), (3) the number of columns included in the dataset after the flattening phase ($\#CD$), and (4) the average accuracy of the discovered model (Ac).

Evaluation Design: For each of the 3 processes, 30 instances are randomly generated by varying the graphical look&feel of the screen captures, including/excluding some UI elements that do not affect the process, as well as replacing texts. For each of these 90 instances, 8 different logs are generated considering the different values of $|L|$ and $B?$. To do this, the instances of the sample event log of the process are used as templates to create similar ones by applying changes in the events while keeping their logic (e.g., a mouse click at a random place inside the same button). The framework is then executed for each of the 720 objects and the response values are calculated considering the average values for the 30 instances. To measure accuracy, we record 100% if the model correctly identifies the condition and 0% otherwise.

Execution Environment: The evaluation was run on a machine with Windows 10, an Intel i9-7900X processor at 3.30 GHz, 64 Gb of RAM, and 10 cores.

Evaluation Results and Data Analysis: Table 1 shows the experiment results. For each problem, identified by P, $B?$, and $|L|$, the average values for the 30 scenarios are shown for each response variable. Some of them are calculated for the feature extraction phase (i.e., tFE and $\#CL$,) and the others are calculated for the decision model discovery (i.e., tDD and Ac for each algorithm).

Regarding the feature extraction phase, we can observe that it is a time-consuming task (i.e., tFE), which takes longer as the number of activities of the process increase (i.e., P) and the size of the log (i.e., $|L|$) grows. This behavior is expected since the extraction algorithms need to be applied to each screen capture that exists in the log before the decision point. Nonetheless, the number of

[3] The set of problems are available at: https://doi.org/10.5281/zenodo.5734323.

Table 1. Experiment results

P	$B?$	$\|L\|$	tFE[a]	$\#CL$	$\#CD$	tDD_{CART}[b]	tDD_{ID3}[b]	$tDD_{C4.5}$[b]	tDD_{CHAID}[b]	Ac_{CART}[c]	Ac_{ID3}[c]	$Ac_{C4.5}$[c]	Ac_{CHAID}[c]
P1	Yes	10	108	25	39	323	339	330	335	12.9	12.9	12.9	12.9
		25	258	25	39	332	361	356	350	83.9	83.9	83.9	83.9
		50	497	25	39	352	394	393	379	100.0	100.0	100.0	100.0
		100	1,029	25	39	334	360	358	352	100.0	100.0	100.0	100.0
	No	10	112	25	39	324	340	329	334	12.9	12.9	12.9	12.9
		25	273	25	39	330	354	351	344	83.9	83.9	83.9	83.9
		50	501	25	39	350	390	388	376	96.8	96.8	96.8	96.8
		100	977	25	39	331	359	358	349	100.0	100.0	100.0	100.0
P2	Yes	10	79	25	79	704	716	708	701	0.0	0.0	0.0	0.0
		25	137	25	79	399	421	404	819	31.0	31.0	31.0	31.0
		50	319	25	79	1,323	1,572	1,405	2,102	93.1	93.1	93.1	93.1
		100	588	25	79	3,495	3,781	3,593	3,531	100.0	100.0	100.0	100.0
	No	10	70	25	79	308	321	307	316	0.0	0.0	0.0	0.0
		25	158	25	79	563	597	570	583	6.9	6.9	6.9	6.9
		50	363	25	79	1,787	1,853	2,175	2,463	75.9	75.9	75.9	75.9
		100	659	25	79	4,627	4,584	3,737	4,764	100.0	100.0	100.0	100.0
P3	Yes	10	51	25	79	1,086	1,090	1,096	1,090	0.0	0.0	0.0	0.0
		25	189	25	79	1,155	1,101	1,058	1,014	0.0	0.0	0.0	0.0
		50	333	25	79	1,818	2,200	1,981	2,748	32.3	48.4	32.3	48.4
		100	724	25	79	2,848	3,039	3,008	5,950	100.0	100.0	96.8	100.0
	No	10	51	25	79	1,085	1,102	1,088	1,087	0.0	0.0	0.0	0.0
		25	219	25	79	1,378	1,094	1,489	1,082	0.0	0.0	0.0	0.0
		50	394	25	79	1,735	2,359	1,893	2,284	19.4	45.2	19.4	45.2
		100	737	25	79	5,473	5,626	3,762	5,908	77.4	100.0	93.6	100.0

[a]Expressed in seconds; [b]Expressed in milliseconds; [c]Expressed in %.

features that are extracted and included in the UI Log (i.e., $\#CL$) only depends on the extractor itself that, for this experiment, the *UI Element Occurrence* extractor (cf. Sect. 4.1) obtains a fixed number of features for each screenshot, i.e., 14—one for each UI element. The other 11 columns are the standard ones defined for the UI Log.

Regarding the decision model discovery phase, the number of columns included in the dataset for training the classification algorithm (i.e., $\#CD$) depends on the number of activities before the decision point. More precisely, $P1$ has 2 activities, while $P2$ and $P3$ both have 4 activities. In this phase, it can be observed that the time for decision model discovery, which is expressed in *ms* (i.e., tDD), is negligible in comparison with tFE, which is expressed in *s*. Moreover, tDD depends on both $|L|$, since the flattener algorithm needs to run over all the events, and $\#CD$, since the tree's training runs over the entries in the dataset. In turns, tDD seems not to be influenced by the algorithm. When analyzing the accuracy of the classification tree (i.e., Ac), it is clear that the framework has a better performance with higher values of $|L|$ since there are more entries in the dataset. However, as expected, the accuracy decreases when the process becomes more complex (i.e., P): more columns in the dataset are not relevant for the decision, i.e., can be considered noise. In addition, for $P3$, we observe more differences between the performance of the algorithms while in $P2$ and $P3$ all the algorithms present the same behavior. This situation may be caused since $P3$ decision depends on more than one feature. Furthermore, having more possible variants (i.e., 4) may affect the performance of the algorithms because fewer rows are obtained for each variant to train the decision models. Unlike the previous response variables, Ac is influenced by whether the log is balanced or not (i.e., $B?$). This behavior can be expected since an unbalanced dataset offers fewer opportunities to distinguish between data and noise. It is important to note that the accuracy is 100% for most cases with $|L| = 100$, which is a reasonably small number for this kind of logs. This provides the insight that the framework *can generally explain the variability within these processes.* This result is very encouraging, particularly when considering the small log sizes included in this experimental set-up.

6 Discussion

Discovering why decisions are made in a process is of utmost importance when the aim is to analyze the variability or even to automate such decisions. The framework proposed in this paper was motivated by the fact that existing Task Mining approaches fail to leverage screen captures when mining UI Logs. In general, alternatives exist to make a transparent analysis of the UI, e.g., navigating the DOM tree or accessing Windows GUI API. Yet, we discovered that considering screen captures is necessary for specific situations like in Business Process Outsourcing scenarios [15], where access to the front-end of the information systems is usually secured or virtualized by systems like Citrix. Although combining both sources of information—transparent analysis of UIs and screen captures—could bring benefits when they are available, this proposal focuses on these situations based exclusively on screenshots proposing a framework that can support a process analyst to (1) accelerate the analysis phase since the automatically-generated decision models include rules linked to the screen captures, which are readable by the analyst, (2) accelerate the development phase since the rules are in a computer-readable format too, and (3) unleash candidates to automate that would be discarded otherwise because no rule could be found to describe the human variability in UI Log. Our experimental evaluation, as conducted with synthetic problems of a variety of complexities, showed positive results. It is noteworthy that these problems use logs that includes one event for each activity while, in practice, there might be more than one events for a single activity, i.e., several events performed over the same screen. However, the approach has not been tested on real UI Logs nor real screen captures; these are clearly limitations to our work.

There are further limitations that can be observed. Specifically, the feature extraction phase highly relies on the image-processing techniques (i.e., the OCR and GUI analysis), which may occasionally present wrong classifications or detections. Moreover, several alternatives exist for these techniques, and they evolve fast due to highly active communities. Nonetheless, the proposed framework is not limited to any of these techniques. After all, it defines a high-level interface to incorporate improved techniques easily. In addition, it is noteworthy that the number of extracted features is directly related to the number of columns in the dataset to be used to train the classifier. Although the aim is to not overlook any relevant feature from the screen captures, non-relevant features may just produce noise that would negatively affect its performance unless more entries are included in the dataset (i.e., longer UI Logs). Fortunately, if the number of dataset columns increases, the evaluation has demonstrated that the framework keeps a reasonably good performance if the size of the log increases as well.

Finally, the validation has intentionally considered problems that rely on the screen captures' content to make the decision. This was done to ensure that the evaluation covers the two phases of the framework. However, the classification algorithm may take into account *all* the information contained in the UI Log, which makes it compatible with existing related approaches, e.g., SmartRPA [4], which is very suitable when screen captures are not required.

7 Related Work

There exist other proposals in the literature that can be applied for decision model discovery. Rozinat and Van der Aalst [32] use decision trees to analyze choices made based on data dependencies that affect the routing of a case. However, this approach does not consider information on the screen nor provides the possibility of showing graphically to a non-expert user why a decision was made.

Agostinelli et al. [3] and Leno et al. [22] cover the complete RPA lifecycle, from event capture to the automatic generation of scripts. Data capture is based on an *Action Logger* that captures the information through plugins [23] or separately within the system [3]. Furthermore, although this capture is mainly focused on the keyboard and mouse events, they capture the DOM tree in those events which interact with the web browser. In contrast to these approaches, our work focuses on screen captures as the primary information source.

Leno et al. [21] present an algorithm that generates a kind of "association rules" between events and results. The information gathering is based on tailor-made plugins. Their approach proposes similar solutions as is the case for our framework. However, in their approach it is not possible to capture the information that the user generates outside the context of these plugins. Finally, Gao et al. [12] propose a solution based on the implementation of decision trees for the algorithmic deduction of RPA rules, based on the captured user behavior. Similar to [21], screen captures and their features are left out from this analysis.

8 Conclusion and Future Work

This paper proposes a framework for discovering decision models from event logs that are extended with screen captures. The framework includes a phase to extract features from such screen captures, as well as a phase to discover decision models using these features. Furthermore, we illustrated our proposal by means of a running example and provided an extensive empirical evaluation. Our proposal advances the state-of-the-art on Task Mining for its application to RPA by providing insights into the drivers for variability in human behavior that is targeted to be automated.

For future work, we plan: (1) to validate the current approach in a real-life context with people with different profiles. So that both the feature extraction algorithm and the decision discovery algorithm can be tested with industrial data and scenarios; (2) to investigate the robustness of the proposal against noise injected at the event level; (3) to investigate mechanisms to reduce noise by filtering/selecting the appropriate features, e.g., by using eye-tracking technologies, which could focus the attention of the feature extraction algorithms on those screen regions where human attention is devoted to; (4) to analyze further classification algorithms to provide more alternatives to express the observed variability; in this way, they can be compared in terms of usability and understandability; and (5) analyze other UI elements detection and classification techniques to compare them with the accuracy of Canny's algorithm and CNN.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Scheppler, B., Weber, C.: Robotic process automation. Informatik Spektrum **43**(2), 152–156 (2020). https://doi.org/10.1007/s00287-020-01263-6
3. Agostinelli, S., Lupia, M., Marrella, A., Mecella, M.: Automated generation of executable RPA scripts from user interface logs. In: BPM, pp. 116–131 (2020)
4. Agostinelli, S., Lupia, M., Marrella, A., Mecella, M.: SmartRPA: a tool to reactively synthesize software robots from user interface logs. In: CAiSE, pp. 137–145 (2021)
5. Augusto, A., et al.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2018)
6. Bazhenova, E., Bülow, S., Weske, M.: Discovering decision models from event logs. In: BIS, pp. 237–251 (2016)
7. Brahmbhatt, S.: Shapes in Images, pp. 67–93. Apress, Berkeley, CA (2013)
8. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA (1984)
9. Capgemini, C.: Robotic Process Automation - Robots conquer business processes in back offices (2017)
10. Denagama Vitharanage, I.M., Bandara, W., Syed, R., Toman, D.: An empirically supported conceptualisation of robotic process automation (RPA) benefits. In: ECIS (2020)
11. Egger, A., ter Hofstede, A.H., Kratsch, W., Leemans, S.J., Röglinger, M., Wynn, M.T.: Bot log mining: using logs from robotic process automation for process mining. In: ER, pp. 51–61 (2020)
12. Gao, J., van Zelst, S.J., Lu, X., van der Aalst, W.M.: Automated robotic process automation: a self-learning approach. In: OTM, pp. 95–112 (2019)
13. Geyer-Klingeberg, J., Nakladal, J., Baldauf, F., Veit, F.: Process mining and robotic process automation: a perfect match. In: BPM, pp. 124–131 (2018)
14. Gilpin, L.H., Bau, D., Yuan, B.Z., Bajwa, A., Specter, M., Kagal, L.: Explaining explanations: an overview of interpretability of machine learning. In: 2018 IEEE 5th DSAA, pp. 80–89 (2018)

15. Jimenez-Ramirez, A., Reijers, H.A., Barba, I., Del Valle, C.: A method to improve the early stages of the robotic process automation lifecycle. In: CAiSE, pp. 446–461 (2019)
16. Kass, G.V.: An exploratory technique for investigating large quantities of categorical data. J. R. Statist. Soc. Ser. C (Appl. Statist.) **29**(2), 119–127 (1980)
17. Keras OCR. https://github.com/faustomorales/keras-ocr. Accessed Mar 2022
18. Lacity, M., Willcocks, L.: What knowledge workers stand to gain from automation. Harv. Bus. Rev. **19**, 1–6 (2015)
19. Leno, V., Armas-Cervantes, A., Dumas, M., La Rosa, M., Maggi, F.M.: Discovering process maps from event streams. In: ICSSP, pp. 86–95 (2018)
20. Leno, V., Augusto, A., Dumas, M., La Rosa, M., Maggi, F.M., Polyvyanyy, A.: Identifying candidate routines for robotic process automation from unsegmented UI logs. In: ICPM, pp. 153–160 (2020)
21. Leno, V., Dumas, M., La Rosa, M., Maggi, F.M., Polyvyanyy, A.: Automated discovery of data transformations for robotic process automation. arXiv preprint arXiv:2001.01007 (2020)
22. Leno, V., Polyvyanyy, A., Dumas, M., La Rosa, M., Maggi, F.M.: Robotic process mining: vision and challenges. Bus. Inf. Syst. Eng. **63**, 1–14 (2020)
23. Leno, V., Polyvyanyy, A., La Rosa, M., Dumas, M., Maggi, F.M.: Action logger: enabling process mining for robotic process automation. In: CEUR Workshop Proceedings (2019)
24. Leshob, A., Bourgouin, A., Renard, L.: Towards a process analysis approach to adopt robotic process automation. In: ICEBE, pp. 46–53 (2018)
25. López-Carnicer, J.M., del Valle, C., Enríquez, J.G.: Towards an opensource logger for the analysis of RPA projects. In: BPM, pp. 176–184 (2020)
26. Majumder, B.P., Potti, N., Tata, S., Wendt, J.B., Zhao, Q., Najork, M.: Representation learning for information extraction from form-like documents. In: ACL, pp. 6495–6504 (2020)
27. Mitchell, T.: Machine Learning. McGraw Hill, New York (1997)
28. Moran, K., Bernal-Cárdenas, C., Curcio, M., Bonett, R., Poshyvanyk, D.: Machine learning-based prototyping of graphical user interfaces for mobile apps. IEEE Trans. Softw. Eng. **46**(2), 196–221 (2018)
29. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
30. Quinlan, J.R.: C4. 5: Programs For Machine Learning. Elsevier, Amsterdam (2014)
31. Reinkemeyer, L.: Process Mining in Action: Principles, Use Cases and Outlook. Springer, Switzerland (2020). https://doi.org/10.1007/978-3-030-40172-6
32. Rozinat, A., van der Aalst, W.M.P.: Decision mining in ProM. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 420–425. Springer, Heidelberg (2006). https://doi.org/10.1007/11841760_33
33. Willcocks, L., Lacity, M.: A new approach to automating services. MIT Sloan Manage. Rev. **58**(1), 40–49 (2016)
34. Xu, Z., Baojie, X., Guoxin, W.: Canny edge detection based on open CV. In: 2017 13th ICEMI, pp. 53–56 (2017)