Interactive Segmentation of User Interface Logs

Simone Agostinelli, Francesco Leotta, and Andrea Marrella

Sapienza Università di Roma, Rome, Italy
{agostinelli,leotta,marrella}@diag.uniroma1.it

Abstract. Robotic Process Automation (RPA) is an emerging technology that relies on software (SW) robots to automate intensive and repetitive tasks (i.e., *routines*) performed by human users on the application's User Interface (UI) of their computer systems. RPA tools are able to capture in dedicated *UI logs* the execution of many routines of interest. A UI log consists of user actions that are mixed in some order that reflects the particular order of their execution by the user, thus potentially belonging to different routines. In the RPA literature, the challenge to understand which user actions contribute to which routines and cluster them into well-bounded *routine traces* is known as *segmentation*. In this paper, we present a novel approach to the discovery of routine traces from unsegmented UI logs, which relies on: *(i)* a frequent-pattern identification technique to automatically derive the routine behaviors (a.k.a. *routine segments*) as recorded into a UI log, *(ii)* a human-in-the-loop interaction to filter out those segments not allowed (i.e., wrongly discovered from the UI log) by any real-world routine under analysis, and *(iii)* a trace alignment technique to cluster all those user actions belonging to a specific segment into routine traces. We evaluate our in terms of supported segmentation variants.

1 Introduction

Robotic Process Automation (RPA) [1] is an emerging technology in the field of Business Process Management (BPM) that relies on software (SW) robots to automate intensive and repetitive tasks (in the following, called *routines*) performed by human users on the application's User Interface (UI) of their computer systems. Similarly to traditional BPM Systems (BPMSs), RPA tools are able to act as effective *service orchestrators*, but without the need of performing the manual configuration steps required by whatever BPMS to run a process, e.g., the definition of specific business rules, the association of resources to the process activities, etc. Since many routine tasks can be implemented through scripting or intelligent recording techniques, RPA projects typically involve comparably little cost than traditional BPM projects [1]. Overall, the target of existing RPA tools is to boost the productivity of organizations by reducing manual labor while improving the operational quality and reducing user input errors.
To take full advantage of this technology, organizations leverage the support of skilled human experts that: *(i)* preliminarily observe how routines are executed on the UI of the involved SW applications (by means of walkthroughs, etc.), *(ii)*

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by doi: 10.1007/978-3-030-91431-8_5

convert such observations in explicit flowchart diagrams, which are specified to depict all the potential behaviors (i.e., *segments*) of the routines of interest, and *(iii)* finally implement the SW robots that automate the routines enactment on a target computer system. However, the current practice is time-consuming and error-prone, as it strongly relies on the ability of human experts to correctly interpret the routines to automate [14]. Consequently, if SW robots are not designed for the appropriate scope of their work, then their implementation cost will increase while no clear business improvement effect will be achieved [13].
To tackle this challenge, in their Robotic Process Mining framework [16], Leno et al. propose to exploit the User Interface (UI) logs recorded by RPA tools to automatically discover the candidate routines that can be later automated with SW robots. UI logs are sequential data of user actions performed on the UI of a computer system during many routines' executions. Typical user actions are: opening a file, selecting/copying a field in a form or a cell in a spreadsheet, read and write from/to databases, open emails and attachments, etc.
To date, when considering state-of-the-art RPA technology, it is evident that the RPA tools available in the market are not able to learn how to automate routines by only interpreting the user actions stored into UI logs [3]. The main trouble is that in a UI log there is not an exact 1:1 mapping among a recorded user action and the specific routine segment it belongs to. In fact, the UI log usually records information about several routines whose actions are mixed in some order that reflects the particular order of their execution by the user. The issue to automatically understand which user actions contribute to a particular routine segment inside a UI log and cluster them into well-bounded *routine traces* (i.e., complete execution instances of a routine) is known as *segmentation* [3,16]. The majority of state-of-the-art segmentation approaches are able to properly extract routine segments (i.e., repeated routine behaviors) from unsegmented UI logs when the routine executions are not interleaved from each others. Only few works are able to partially untangle unsegmented UI logs consisting of many interleaved routines executions, but with the assumption that any routine provides its own, separate universe of user actions. This is a relevant limitation, since it is quite common that real-world routines may share the same user actions (e.g., copy and paste data across cells of a spreadsheet) to achieve their objectives.
In this paper, we propose a novel approach to the segmentation of UI logs that aims to mitigate the aforementioned issue showing its effectiveness in terms of supported segmentation variants. The approach relies on three key ingredients:

1. a *frequent-pattern identification technique* to automatically discover the observed segments of the routines as recorded into the UI log. In this phase, the risk exists that some wrong segments are discovered, i.e., not allowed from the real-world routines that are known to be valid at the outset.
2. a *human-in-the-loop interaction* that enables human experts to visualize the *declarative constraints* inferred by the discovered routine segments. Such constraints describe the temporally extended relations between user actions that must be satisfied throughout a routine segment (e.g., an action a_1 must be eventually followed by an action a_2). In a nutshell, they collectively deter-

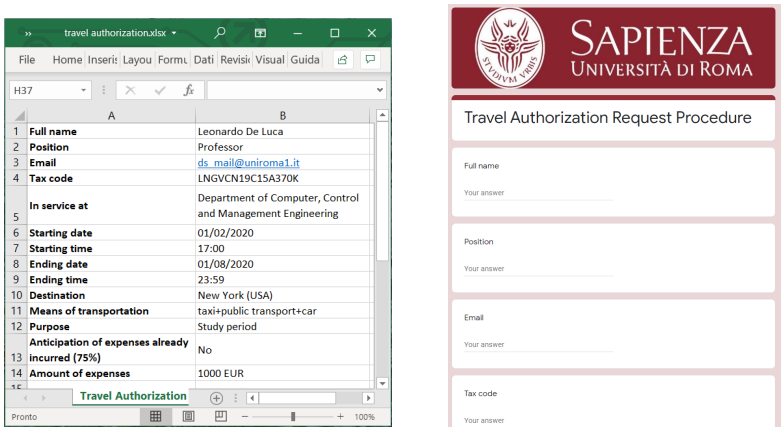
mine the observed behaviors of the routine segments from the UI log. This knowledge allows human experts to identify and remove those constraints that should not be compliant with any real-world routine behavior, thus filtering out the not valid (i.e., wrongly discovered) routine segments;
3. a *trace alignment technique* to cluster all the user actions associated to a valid routine segment into well-bounded routine traces.

We show the feasibility of our approach by employing a dataset of 144 synthetic UI logs covering different segmentation cases to measure to what extent the approach is able to (re)discover the valid routine segments from such UI logs.
The rest of the paper is organized as follows. Section 2 introduces a running example that will be used to explain our approach, and discusses the relevant background on the segmentation of UI logs with all its potential variants. In Section 3, we present the details of our approach to the automated segmentation of UI logs. Section 4 evaluates the feasibility of the proposed approach against synthetic UI logs. Finally, Section 5 discusses the novelty of our approach against literature works, while Section 6 draws conclusions, traces future works and outlines a critical discussion about the general applicability of the approach.

2 Background

2.1 Running Example

In this section, we describe a RPA use case inspired by a real-life scenario at Department of Computer, Control and Management Engineering (DIAG) of Sapienza Università di Roma. The scenario concerns the filling of the travel authorization request form made by personnel of DIAG for travel requiring prior approval. The request applicant must fill a well-structured Excel spreadsheet (cf. Fig. 1(a)) providing some personal information, such as her/his bio-data and the email address, together with further information related to the travel, including the destination, the starting/ending date/time, the means of transport to be used, the travel purpose, and the envisioned amount of travel expenses, associated with the possibility to request an anticipation of the expenses already incurred (e.g., to request in advance a visa). When ready, the spreadsheet is sent via email to an employee of the Administration Office of DIAG, which is in charge of approving and elaborating the request. Concretely, for each row in the spreadsheet, the employee manually copies every cell in that row and pastes that into the corresponding text field in a dedicated Google form (cf. Fig. 1(b)), accessible just by the Administration staff. Once the data transfer for a given travel authorization request has been completed, the employee presses the "Submit" button to submit the data into an internal database.
In addition, if the request applicant declares that s/he would like to use her/his personal car as one of the means of transport for the travel, then s/he has to fill a dedicated web form required for activating a special insurance for the part of the travel that will be performed with the car. This further request will be delivered to the Administration staff via email, and the employee in charge of



(a) Excel spreadsheet (b) Google form

Fig. 1: UIs involved in the use case

processing it can either approve or reject such request. At the end, the applicant will be automatically notified via email of the approval/rejection of the request. The above procedure, which involves two main routines (in the following, we will denote them as R1 and R2), is performed manually by an employee of the Administration Office of DIAG, and it should be repeated for any new travel request. Routines such as these ones are good candidates to be encoded with executable scripts and enacted by means of a SW robot within a commercial RPA tool. However, unless there is complete a-priori knowledge of the specific routines that are enacted on the UI and of their concrete composition, their automated identification from an UI log is challenging, since the associated user actions may be scattered across the log, interleaved with other actions that are not part of the routine under analysis, and potentially shared by many routines. Based on the above description, it becomes clear that a proper execution of R1 requires a path on the UI made by the following user actions:[1]

- loginMail, to access the client email;
- accessMail, to access the specific email with the travel request;
- downloadAttachment, to download the Excel file including the travel request;
- openWorkbook, to open the Excel spreadsheet;
- openGoogleForm, to access the Google Form to be filled;
- getCell, to select the cell in the i-th row of the Excel spreadsheet;
- copy, to copy the content of the selected cell;
- clickTextField, to select the specific text field of the Google form where the content of the cell should be pasted;

[1] Note that the user actions recorded in a UI log can have a finer granularity than the high-level ones used here just with the purpose of describing the routine's behavior.

– paste, to paste the content of the cell into a text field of the Google form;
– formSubmit, to finally submit the Google form to the internal database.

Note that the user actions openWorkbook and openGoogleForm can be performed in any order. Moreover, the sequence of actions ⟨getCell, copy, clickTextField, paste⟩ will be repeated for any travel information to be moved from the Excel spreadsheet to the Google form. On the other hand, the path of user actions in the UI to properly enact R2 is as follows:

– loginMail, to access the client email;
– accessMail, to access the specific email with the request for travel insurance;
– clickLink, to click the link included in the email that opens the Google form with the request to activate the travel insurance on a web browser;
– approveRequest, to approve the request on the Google form;
– rejectRequest, to reject the request on the Google form;

Note that the execution of approveRequest and rejectRequest is exclusive.
In the rest of the paper, we concisely represent the universe of user actions of interest for R1 and R2 as follows: $Z = \{A, B, C, D, E, F, G, H, I, L, M, N, O\}$, such that: A = loginMail, B = accessMail, C = downloadAttachment, D = openWorkbook, E = openGoogleForm, F = getCell, G = copy, H = clickTextField, I = paste, L = formSubmit, M = clickLink, N = approveRequest, O = rejectRequest.

2.2 Segmentation of UI logs

In this section, we provide the relevant background on UI logs and we explain in detail the issue of segmentation of UI logs with all its potential variants.
A UI log typically consists of a long sequence of user actions recorded during one user interaction session.[2] Such actions include all the steps required to accomplish one or more relevant routines using the UI of one or many sw application/s. For instance, in Fig. 2, we show a snapshot of a UI log captured using a dedicated action logger[3] during the execution of R1 and R2. The employed action logger enables to record the *events* happened on the UI, enriched with several data fields describing their "anatomy". For a given event, such fields are useful to keep track the name and the timestamp of the user action performed on the UI, the involved sw application, the resource that performed the action, etc.
As shown in Fig. 2, a UI log is not specifically recorded to capture pre-identified routines. A UI log may contain multiple and interleaved executions of one/many routine/s (cf. in Fig. 2 the blue/red boxes that group the user actions belonging to R1 and R2, respectively), as well as redundant behavior and noise. We consider as *redundant* any user action that is unnecessary repeated during the execution of a routine, e.g., a text value that is first pasted in a wrong field and then is moved in the right place through a corrective action on the UI. On the other hand, we

[2] We interpret a user session as a group of interactions that a single user takes within a given time frame on the UI of a specific computer system.
[3] https://github.com/bpm-diag/smartRPA.

	A	B	C	D	E	F	G	H	I	J
1	timestamp	user	category	application	event_type	event_src_path	clipboard_content	workbook	worksheet	cell_content
2	2020-04-06 13:47	Simone	Mail	Outlook	loginMail					
3	2020-04-06 13:47	Simone	Mail	Outlook	accessMail					
4	2020-04-06 13:47	Simone	Mail	Outlook	downloadAttachment					
5	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	openWorkbook	C:\Users\Simone\Desktop\richiesta missione a		richiesta missione.xlsx	Foglio1	
6	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	openWindow	C:\Users\Simone\Desktop		richiesta missione.xlsx	Foglio1	
7	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	afterCalculate					
8	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	resizeWindow	C:\Users\Simone\Desktop		richiesta missione.xlsx	Foglio1	
9	2020-04-06 13:47	Simone	Browser	Chrome	openGoogleForm					
10	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	getCell			richiesta missione.xlsx	Foglio1	Simone Agostinelli
11	2020-04-06 13:47	Simone	Clipboard	Clipboard	copy		Simone Agostinelli			
12	2020-04-06 13:47	Simone	Browser	Chrome	clickTextField					
13	2020-04-06 13:48	Simone	Mail	Outlook	clickLink					
14	2020-04-06 13:48	Simone	Browser	Chrome	paste		Simone Agostinelli			
15	2020-04-06 13:48	Simone	Browser	Chrome	changeField					
16	2020-04-06 13:48	Simone	Browser	Chrome	approveRequest					
17	2020-04-06 13:48	Simone	MicrosoftOffice	Microsoft Excel	getCell			richiesta missione.xlsx	Foglio1	Dottorando
18	2020-04-06 13:48	Simone	Clipboard	Clipboard	copy		Dottorando			
19	2020-04-06 13:48	Simone	MicrosoftOffice	Microsoft Excel	resizeWindow	C:\Users\Simone\Desktop		richiesta missione.xlsx	Foglio1	
20	2020-04-06 13:48	Simone	Browser	Chrome	clickTextField					
21	2020-04-06 13:48	Simone	Browser	Chrome	paste		Dottorando			

Fig. 2: Snapshot of a UI log captured during the executions of R1 and R2

consider as *noise* all those actions that do not contribute to the achievement of any routine target, e.g., a window that is resized. In Fig. 2, the sequences of user actions that are not surrounded by a blue/red box can be safely labeled as noise.

In this context, *segmentation* techniques aim first to extract from a UI log all those user actions that are compliant with a specific *routine segment*, i.e., with a repetitive routine behavior as observed in the UI log. Then, the target is to cluster such user actions into well-bounded *routine traces*, which are complete and independent execution instances of the routine within the UI log. Such traces are finally stored in a dedicated *routine-based logs*, which capture all the user actions happened during many different executions of the routine and compliant with a specific routine segment, thus achieving the segmentation task. It is worth noticing that a routine-based log obtained in this way can eventually be employed by the commercial RPA tools to synthesize executable scripts in form of SW robots that will emulate the routine behavior.

For example, an allowed routine segment of R1 is $\langle \boldsymbol{A}, \boldsymbol{B}, C, D, E, F, G, H, I, L \rangle$. From the description of the use case, allowed routine segments are also those ones where: *(i)* $\boldsymbol{A}$ is skipped (if the user is already logged in the client email); *(ii)* the pair of actions $\langle D, E \rangle$ is performed in reverse order; *(iii)* the sequence of actions $\langle F, G, H, I \rangle$ is executed several time before submitting the Google form. On the other hand, two allowed routine segments can be observed from R2: $\langle \boldsymbol{A}, \boldsymbol{B}, M, N \rangle$ and $\langle \boldsymbol{A}, \boldsymbol{B}, M, O \rangle$, again with the possibility to skip $\boldsymbol{A}$, i.e., the access to the client email. Note that $\boldsymbol{A}$ and $\boldsymbol{B}$ can be employed by both R1 and R2 to achieve their targets. By analyzing the log, it can be noted that: $\boldsymbol{A}$ is *potentially involved* in the enactment of any execution of R1 and R2, while $\boldsymbol{B}$ is *required by all* executions of R1 and R2, but it is not clear the association between the single executions of $\boldsymbol{B}$ and the routine segments they belong to. Any observed execution of user actions in the UI log that matches with one of the above routine segments can be considered as a valid routine trace.

According to [5], we can distinguish between three major forms of UI logs, which can be categorized as follows:

- **Case 1**. A UI log captures many not interleaved (*case 1.1*) or interleaved (*case 1.2*) executions of the same routine.
- **Case 2**. A UI log captures many executions of different routines, but with the assumption that different routines do not have any user action in common. Four variants of this case can be identified: clear separation in the UI log between the routines' executions (*case 2.1*); many executions of the same routine can be recorded in an interleaved fashion, but the executions of different routines are separated from each others (*case 2.2*); the executions of different routines can be recorded in an interleaved fashion, but the executions of a specific routine can not be enacted in an interleaved way (*case 2.3*); the executions of any routine can be always interleaved from each others (*case 2.4*).
- **Case 3**. Similarly to Case 2, it provides four variants (cases *3.1*, *3.2*, *3.3*, and *3.4*)), with the only difference that a same kind of user action can be employed by many different routines to achieve their objectives, e.g., the UI log associated with the running example in Section 2.1 belongs to Case 3.

While the literature does not provide works able to properly segment UI logs including user actions "shared" by many routine executions, in this paper we propose an approach that is able to relax this assumption and to achieve the following segmentation cases: *1.1*, *2.1*, *2.3*, *3.1* and *3.3*.

3 Approach

Our approach to the segmentation of UI logs can be considered a *semi-supervised* one, as it integrates the usage of automated techniques with the intervention of human experts in some specific points of the approach. To be more precise, as shown in Fig. 3, starting from an unsegmented UI log previously recorded by a RPA tool, the first step is to inject into the UI log the *end-delimiters* of the routines under examination. An end-delimiter is a dummy action added to the UI log immediately after the user action that is known to complete a routine execution. If we consider our running example in Section 2.1, an end-delimiter is always required after the final action of R1, i.e., formSubmit, and after one of the final actions or R2, i.e., approveRequest or rejectRequest. In this paper, we assume that the knowledge of the final action(s) of a routine is given at the outset. Such information can be obtained, for example, by interviewing the users that are in charge to execute the routines of interest.
The second step of the approach consists of automatically extracting the observed routines' behaviors (i.e., the routine segments) directly from the UI log with the end-delimiters. To this aim, we employ a *frequent-pattern identification technique* [9], which has been properly customized for this purpose.
Since from the previous step there is the possibility that some (not allowed) segments are identified as if they would be valid, the third step of the approach involves a *human-in-the-loop interaction* to filter out these segments. Specifically, we automatically infer the declarative constraints (i.e., the temporally extended relations between user actions) that must be satisfied throughout a

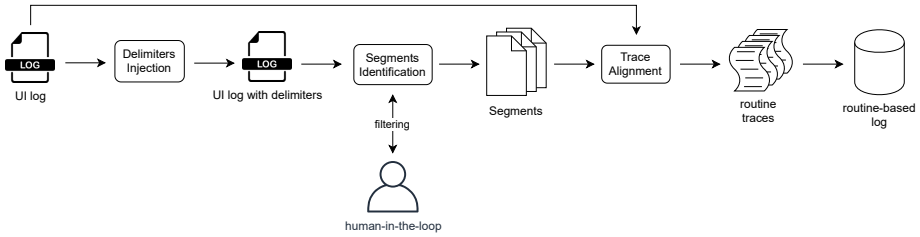


Fig. 3: Overview of our general approach to the segmentation of UI logs

routine segment. In this way, we enable human experts to identify and remove those constraints that should not be compliant with any real-world routine behavior, thus removing the wrongly discovered routine segments from the UI log. Finally, starting from any of the remaining (valid) routine segments, we employ a customized version of a *trace alignment* technique in Process Mining [2] to automatically detect and extract the *routine traces* by the original UI log. Such traces will be stored in a dedicated *routine-based log*. Therefore, the final outcome of our segmentation approach will be a collection of as many routine-based logs as are the number of valid routine segments. By identifying the routine traces, we are also able to filter out those actions in the UI log that are not part of the routine under observation and hence are redundant or represent noise.
In the following sections, we discuss in detail all the steps of our approach, instantiating them over the running example of Section 2.1.

3.1 Segments Discovery through Frequent-Pattern Identification

Pattern identification is a common task in data sequences analysis. As an example, in the field of smart spaces, patterns are identified in sensor logs representing human routines [17]. These patterns are then used to learn models of human behavior that can be used at runtime for activity recognition or anomaly detection. In such a scenario, authors in [9] proposed an approach based on minimum description length (MDL) principle. In this paper, we have customized the technique presented in [9] for automatically identifying the routine segments from UI logs with the end-delimiters properly converted into ad-hoc datasets.
The algorithm takes as input a dataset of a sequence of sensor events witnessing human interactions with the environment. At each step, the algorithm looks for patterns that best compress the dataset. A pattern consists of a specific sequence of sensor events and all of its occurrences in the dataset. In our RPA application scenario, the sensor events represent the user actions involved in each routine(s) execution(s), and the frequent patterns are the discovered routine segments.
Starting from a single pattern for each different sensor event, the algorithm at each step tries to extend patterns aiming at the best compression possible. Every instance of the pattern, in particular, is replaced by a symbol associated to the pattern. The compression of a dataset D given a pattern P is given

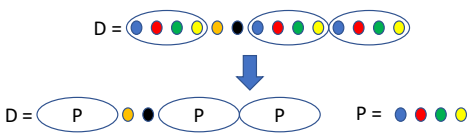


Fig. 4: A dataset compression step in segments discovery

by the formula $\frac{DL(D)}{DL(D|P)+DL(P)}$, where $DL(D)$ represent the description length, measured for example in bits of the dataset with the current patterns, $DL(D|P)$ represents the description length of D if all of the occurrences of P are replaced with a symbol, and $DL(P)$ represents the description length of the pattern, which must be taken into account in compression evaluation. The algorithm stops as soon as no further compression is possible, returning all the patterns found (i.e., all the discovered routine segments). Fig. 4 shows a compression step where a pattern P of repeating events (for simplicity colors have been used instead of labels) is identified and the dataset is compressed accordingly. Noteworthy, for certain parts of the dataset, no pattern is found whose definition improve compression (with the exception of the initial patterns of length one).

We show now how an execution instance of the above algorithm can be applied to the following UI log (that already includes the end-delimiters) generated from the running example of Section 2.1: $U = \{\boldsymbol{A}, \boldsymbol{B}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11}, X, \boldsymbol{B}, M_{21}, N_{21}, Z, \boldsymbol{B}, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12}, X, \boldsymbol{B}, M_{22}, O_{22}, Z, \ldots, \boldsymbol{A}, \boldsymbol{B}, C_{1(i-1)}, \boldsymbol{Y_1}, D_{1(i-1)}, E_{1(i-1)}, F_{1(i-1)}, G_{1(i-1)}, G_{1(i-1)}, G_{1(i-1)}, H_{1(i-1)}, I_{1(i-1)}, L_{1(i-1)}, X, \boldsymbol{B}, M_{2(i-1)}, N_{2(i-1)}, Z, \ldots, \boldsymbol{B}, \boldsymbol{Y_{n-1}}, C_{1i}, D_{1i}, E_{1i}, \boldsymbol{Y_n}, F_{1i}, G_{1i}, H_{1i}, I_{1i}, I_{1i}, I_{1i}, L_{1i}, X, \boldsymbol{B}, M_{2i}, O_{2i}, Z\}$. For the sake of understandability, we use a numerical subscript ji associated to any user action to indicate that it belongs to the $i-th$ execution of the $j-th$ routine under study. This information is not recorded into the UI log, and discovering it (i.e., the identification of the subscripts) is one of the "implicit" effects of segmentation when routine traces are built. Note that $\boldsymbol{A}$ and $\boldsymbol{B}$ are not decorated with subscripts since they can potentially belong to executions of R1 or R2. The log contains elements of noise, i.e., user actions $\boldsymbol{Y_{k\in\{1,n\}}}$ that are not allowed by R1 and R2, and redundant actions like G and I that are unnecessary repeated multiple times. X and Z are the end-delimiters for the executions of R1 and R2. The delimiters injection stage is crucial to drive the discovery of the largest possible set of valid routine segments, otherwise the technique would detect only a small subset of them. For example, let us suppose that the UI log includes only user actions related to two routines A and B without the presence of any end-delimiter. In this case, the UI log will likely include different sequences of consecutive routine segments of the kind A*, B* or AB*. In this condition, any compression algorithm will likely merge multiple routine segments into cumulative symbols (e.g., AAA, BB, ABAB) rather than highlighting single routine executions. This issue becomes less relevant when between the execution of two separate routines there are no repetitive actions. However, while the latter assumption is reasonable in case of recording of human habits, it is far from being

realistic in case of UI logs recording low-level user actions performed during the interaction with a computer system.
Based on the foregoing, the output of the segments discovery stage is represented by a set of identified frequent segments (some of them may not be compliant with the real-world routine behaviors, see the next section), as follows:

- $\{\langle F, G\rangle, \langle C, D, E\rangle, \langle H, I, L\rangle, \langle C, D, E, F, G, H, I, L\rangle, \langle \boldsymbol{B}, C, D, E, F, G, H, I, L\rangle, \langle \boldsymbol{A}, \boldsymbol{B}, C, D, E, F, G, H, I, L\rangle\}$
- $\{\langle \boldsymbol{A}, \boldsymbol{B}\rangle, \langle \boldsymbol{B}, M\rangle, \langle \boldsymbol{B}, M, O\rangle, \langle \boldsymbol{B}, M, N\rangle\}$

3.2 Human-in-the-loop Interaction

Once the routine segments have been discovered, the possibility exists that many of them represent not allowed routine behaviors. This happens because a UI log combines the execution of several routines that are usually interleaved from each others. In addition, in case of routines that make use of the same kinds of user actions to achieve their goals, it may happen that new patterns of repeated user actions, which represent potential not allowed routine segments, are rather detected as valid ones within the UI log.
On the basis of the experiments performed in Section 4, it becomes clear that the employed frequent-pattern identification algorithm is able to (re)discover the allowed routine segments that are known to be recorded in the input UI logs. However, since there is the possibility that some (not allowed) segments are identified as if they would be valid, a *human-in-the-loop interaction* is required to filter out all those routine segments representing behaviors that should not be allowed by any real-world routine of interest. Specifically, starting from the discovered routine segments, we invoke for any of these segments the Declare Miner algorithm implemented in [6] to infer the declarative constraints (i.e., the temporally extended relations between user actions) that must be satisfied throughout the segments. The constraints are represented using Declare, a well-known declarative process modeling language introduced in [10]. Declare constraints can be divided into four main groups: existence, relation, mutual and negative constraints. We notice that the use of declarative notations has been already demonstrated as an effective tool to visually support expert users in the analysis of event logs [21].
At this point, one or more human expert(s) may be involved to evaluate the constraints derived for any routine segment and remove those ones that are considered not compliant with any real-world routine behavior. Detecting and removing these constraints means to filter out all the not allowed (i.e., wrongly discovered) routine segments.
For example, if we consider the discovered segment $\langle C, D, E\rangle$, the following (simple) Declare constraints (among the others) hold: $Init(C)$ and $End(E)$, meaning that routines' executions starting with C or ending with E have been discovered into the UI log. An expert user that is aware of the behavior of the real-world routines under analysis can immediately understand that the above Declare constraints should not hold in reality, since R1 and R2 can start only with **A** or

B and end with L, O or N. For this reason, the above Declare constraints can be considered both as wrongly representative of the routines under analysis. As a consequence, all the discovered segments for which one of the above Declare constraints hold can be immediately discarded. For the sake of space, we do not show here all the Declare constraints that hold for any of the discovered segments. However, we point out that the iterative analysis of the Declare constraints associated to the discovered segments will support the human experts to easily detect and filter out those segments that must not be later emulated by SW robots. The list of allowed segments for our running example is the following:

- $\{\langle \boldsymbol{B}, C, D, E, F, G, H, I, L\rangle, \langle \boldsymbol{A}, \boldsymbol{B}, C, D, E, F, G, H, I, L\rangle\}$
- $\{\langle \boldsymbol{B}, M, O\rangle, \langle \boldsymbol{B}, M, N\rangle\}$

3.3 Trace Alignment

Trace alignment [2] is a conformance checking technique within Process Mining that replays the content of any trace in a log against a process model, one event at a time. For each trace in the log, the technique identifies the closest corresponding trace that can be parsed by the model, i.e., an *alignment*.
We perform trace alignment by constructing an alignment of a UI log U (note that we can consider the entire content of the UI log as a single trace) and a process model W (representing a valid routine segment) as a Petri Net, which allows us to exactly pinpoint where deviations occur. Specifically, we relate "moves" in the log to "moves" in the model in order to establish an alignment between U and W. However, it may be that some of the moves in the log cannot be mimicked by the model and vice versa. In particular, we are interested in synchronous moves between U and W. If they exist, the user actions involved in such synchronous moves are extracted and stored into a routine-based log.
We have implemented a customized version of the above trace alignment algorithm as a supervised segmentation technique [4] that is able to segment a UI log and achieve all variants of cases 1, 2 and (partially) 3 except when there are interleaved executions of shared user actions by many routines. In that case, the risk exists that a shared user action is associated to a wrong routine execution (i.e., case 3.3 and 3.4 are not covered). Thus, while in [4], to make the algorithm works, it is required to know a-priori the structure (i.e., the flowchart) of the routines to identify in the UI log (cf. [20]), the novelty of the proposed approach is to semi-automatically discover such structures in the form of routine segments, and then used them as input for the supervised segmentation technique in [4].
In the case of our running example, starting from the outcome of the previous step (i.e., the valid routine segments), the output of the trace alignment will be a set of four routine-based logs generated as follows:

- $U_{W1} = \{\langle \boldsymbol{A_{11}}, \boldsymbol{B_{11}}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11}\rangle, \ldots, \langle \boldsymbol{A_{1(i-1)}}, \boldsymbol{B_{1(i-1)}}, C_{1(i-1)}, D_{1(i-1)}, E_{1(i-1)}, F_{1(i-1)}, G_{1(i-1)}, H_{1(i-1)}, I_{1(i-1)}, L_{1(i-1)}, \rangle\}$
- $U_{W2} = \{\langle \boldsymbol{B_{12}}, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12}, \rangle, \ldots, \langle \boldsymbol{B_{1i}}, C_{1i}, D_{1i}, E_{1i}, F_{1i}, G_{1i}, H_{1i}, I_{1i}, L_{1i}\rangle\}$
- $U_{W3} = \{\langle \boldsymbol{B_{21}}, M_{21}, N_{21}\rangle, \ldots, \langle \boldsymbol{B_{2(i-1)}}, M_{2(i-1)}, N_{2(i-1)}\rangle\}$
- $U_{W4} = \{\langle \boldsymbol{B_{22}}, M_{22}, O_{22}\rangle, \ldots, \langle \boldsymbol{B_{2i}}, M_{2i}, O_{2i}\rangle\}$

Table 1: Experiments' results. For each segmentation case the number of actions is 28, 21 and 20 (resp.). Only logs with 20 different allowed segments are shown here, and the number of valid routine behaviors is the 70% of the 1000s that were introduced in the UI logs, while the other 30% may be affected by noise.

Case 1	# discovered segments (valid/wrong)		
Noise	0%	10%	20%
no repetitive actions	20/2	20/88	20/118
repetitive actions	20/11	16/161	16/179
Case 2	# discovered segments (valid/wrong)		
Noise	0%	10%	20%
no repetitive actions	20/2	20/59	20/69
repetitive actions	20/10	20/132	20/136
Case 3	# discovered segments (valid/wrong)		
Noise	0%	10%	20%
no repetitive actions	20/6	20/53	20/67
repetitive actions	20/13	20/146	20/170

4 Evaluation

To investigate the feasibility of our approach to the automated segmentation of UI logs, we assessed to what extent it is able to (re)discover routine segments that are known to be recorded into the input UI logs. Specifically, we have synthetically generated 144 different UI logs, in a way that each UI log consisted of 1000 routine executions and was characterized by a unique configuration by varying the following inputs:

- *valid_routine_segments*: number of different routines segments (5/10/15/20), in terms of allowed behaviors, included in the UI log.
- *alphabet_size*: size of the alphabet of user actions for each segmentation case: Case 1 (13/18/23/28); Case 2 (15/16/18/21); Case 3 (13/15/17/20).
- *valid_traces*: percentage of allowed behaviors recorded into the UI log (50%/70%/100%). The remaining portion of the UI log (50%/30%) may be dirty, i.e., it contains routine executions potentially affected by noise.
- *percentage of noise* in the remaining (dirty) portion of the UI log (10%/20%).

The synthetic UI logs generated for the test and the complete list of results can be analyzed at: http://tinyurl.com/icsoc2021. The implementation of our approach is available: https://github.com/bpm-diag/INTSEG. For the sake of space, we present in Table 1 only a view of the results in one of the most complex cases to tackle. The results indicate that the approach scales very well in case of an increasing number of different routine segments to be discovered and with an alphabet of user actions of growing size. The computation time is not shown,

since it ranges from milliseconds for UI logs with 5 different routine segments up to few seconds for UI logs with 20 segments. This result was expected, since more segments in a UI log means more executions to analyze and interpret.
By analyzing the results, we can infer that the approach is able to discover the same allowed routine segments that were synthetically introduced in the routine executions recorded in the UI logs, achieving the following segmentation cases: *1.1*, *2.1*, *2.3*, *3.1* and *3.3*. On the other hand, our approach seems to lack in the computation of valid routine segments in presence of repetitive user actions (i.e., user actions that are repeated in a loop), when there are several routine segments generated by different executions of the same routine. This is due to the fact that similar sequences of user actions tend to be compressed together, and since they are generated from the same routine, the risk exists that different sequences are wrongly recognized as the same and bounded together, thus leading to a number of routine segments lower than ones that were synthetically introduced.

5 Related work

Segmentation is currently considered as one of the "hot" key research effort to investigate [3,16] in the RPA field. Concerning RPA-related techniques, Bosco et al. [8] provide a method that exploits rule mining and data transformation techniques, able to discover routines that are fully deterministic and thus amenable for automation directly from UI logs. This approach is effective in case of UI logs that keep track of well-bounded routine executions (cases 1.1 and 2.1), and becomes inadequate when the UI log records information about several routines whose actions are potentially interleaved. In this direction, Leno et al. [15] propose a technique to identify execution traces of a specific routine relying on the automated synthesis of a control-flow graph, describing the observed directly-follow relations between the user actions. This technique is able to achieve cases 1.1, 1.2 and 2.1, and (only) partially the cases 2.2, 2.3 and 2.4, losing in accuracy in presence of recurrent noise and interleaved routine executions. The main limitation of the above techniques is tackled in [4], which presents a supervised segmentation technique that is able to achieve all variants of cases 1, 2 and (partially) 3 except when there are interleaved executions of shared user actions by many routines. In this paper, we exploit the technique presented in [4] to the discovery of routine traces given a set of input routine segments.
Even if more focused on traditional business processes in BPM rather than on RPA routines, Fazzinga et al. [11] employ predefined behavioral models to establish which process activities belong to which process model. The technique works well when there are no interleaved user actions belonging to one or more routines, since it is not able to discriminate which event instance (but just the event type) belongs to which process model. This makes [11] effective to tackle cases 1.1, 2.1 and 3.1. Closely related to [11], there is the work of Liu [18]. The author proposes a probabilistic approach to learn workflow models from interleaved event logs, dealing with noises in the log data. Since each workflow is assigned with a disjoint set of operations, the work [18] is able to achieve both

cases 1.1 and 2.1, but partially cases 2.2, 2.3 and 2.4 (the approach can lose accuracy in assigning operations to workflows).
There exist other approaches whose the target is not to exactly resolve the segmentation issue. Many research works exist that analyze UI logs at different levels of abstraction and that can be potentially useful to realize segmentation techniques. With the term *"abstraction"* we mean that groups of user actions to be interpreted as executions of high-level activities. Baier et al. [7] propose a method to find a global one-to-one mapping between the user actions that appear in the UI log and the high-level activities of a given model. Similarly, Ferreira et al. [12], starting from a state-machine model describing the routine of interest in terms of high-level activities, employ heuristic techniques to find a mapping from a "micro-sequence" of user actions to the "macro-sequence" of activities in the state-machine model. Finally, Mannhardt et al. [19] present a technique that map low-level event types to multiple high-level activities (while the event instances, i.e., with a specific timestamp in the log, can be coupled with a single high-level activity). However, segmentation techniques in RPA must enable to associate low-level event instances (i.e., user actions) to multiple routines, making abstractions techniques ineffective to tackle all those cases where is the presence of interleaving user actions of many routines. As a consequence, all abstraction techniques are effective to achieve cases 1.1 and 2.1 only.

6 Discussion and Concluding Remarks

In this paper we have presented an approach that tackles the segmentation challenge relying on three main steps: *(i)* a frequent-pattern identification technique to automatically derive the observed routine behaviors from a UI log, *(ii)* a human-in-the-loop interaction to filter out those behaviors not allowed by any real-world routine execution, and *(iii)* a trace alignment technique in Process Mining to cluster all user actions belonging to a specific routine behavior into well-bounded routine traces. Our approach is based on a semi-supervised assumption, since we know a-priori the end-delimiters to be associated to any user action that ends a routine execution. On the other hand, the approach is not aware of the concrete behavior of the routines of interest, which will be discovered by the approach itself. For this reason, we consider this contribution as an important step towards the development of a more complete and unsupervised technique to the segmentation of UI logs.
The presented approach is able to extract routine traces from unsegmented UI logs that record in an interleaved fashion many different routines but not the routine executions, thus losing in accuracy when there is the presence of interleaving executions of the same routine. In addition, it is also able to properly deal with shared user actions required by all routine executions in the UI log, thus achieving the cases *1.1*, *2.1*, *2.3*, *3.1*, and *3.3*.
As a future work, we are going to perform a robust evaluation: *(i)* on real-world case studies with heterogeneous UI logs, and *(ii)* on the impact of the human-in-the-loop interaction to filter out wrongly discovered routine segments. In addi-

tion, we aim at relaxing the semi-supervised assumption by employing *machine learning* and *DNN* techniques to automatically identify the end-delimiters.

Acknowledgments. This work has been supported by the "Dipartimento di Eccellenza" grant, the H2020 project DataCloud and the Sapienza grant BPbots.

References

1. Van der Aalst, W.M., Bichler, M., Heinzl, A.: Robotic process automation. Bus Inf Syst Eng **60** (2018)
2. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-Based Fitness in Conformance Checking. In: ACSD'11. pp. 57–66. IEEE (2011)
3. Agostinelli, S., Marrella, A., Mecella, M.: Research Challenges for Intelligent Robotic Process Automation. In: BPM 2019 Int. Workshops, AI4BBPM'19 (2019)
4. Agostinelli, S., Marrella, A., Mecella, M.: Automated Segmentation of User Interface Logs. In: RPA. Management, Technology, Applications. De Gruyter (2021)
5. Agostinelli, S., Marrella, A., Mecella, M.: Exploring the Challenge of Automated Segmentation in Robotic Process Automation. In: 15th Int. Conf. on Research Challenges in Information Science, RCIS'21 (2021)
6. Alman, A., Di Ciccio, C., Haas, D., Maggi, F.M., Nolte, A.: Rule mining with RuM. In: ICPM'20. IEEE (2020)
7. Baier, T., Rogge-Solti, A., Mendling, J., Weske, M.: Matching of Events and Activities: an Approach Based on Behavioral Constraint Satisfaction. In: SAC (2015)
8. Bosco, A., Augusto, A., Dumas, M., La Rosa, M., Fortino, G.: Discovering Automatable Routines from User Interaction Logs. In: 17th Int. Conf. on Business Process Management (Forum track). pp. 144–162 (2019)
9. Cook, D.J., Krishnan, N.C., Rashidi, P.: Activity Discovery and Activity Recognition: A New Partnership. IEEE Transactions on Cybernetics **43**(3), 820–828 (2013)
10. van Der Aalst, W.M., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Comp. Sc.-Res. and Dev. **23**(2) (2009)
11. Fazzinga, B., Flesca, S., Furfaro, F., Masciari, E., Pontieri, L.: Efficiently Interpreting Traces of Low Level Events in Business Process Logs. Inf. Syst. **73** (2018)
12. Ferreira, D.R., Szimanski, F., Ralha, C.G.: Improving Process Models by Mining Mappings of Low-Level Events to High-Level Activities. Inf. Syst. **43**(2) (2014)
13. Gao, J., van Zelst, S.J., Lu, X., van der Aalst, W.M.P.: Automated Robotic Process Automation: A Self-Learning Approach. In: CoopIS'19. pp. 95–112. Springer (2019)
14. 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 (2019)
15. 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: 2nd Int. Conf. on Process Mining. pp. 153–160 (2020)
16. Leno, V., Polyvyanyy, A., Dumas, M., La Rosa, M., Maggi, F.M.: Robotic Process Mining: Vision and Challenges. Bus. & Inf. Sys. Eng. pp. 1–14 (2020)
17. Leotta, F., Mecella, M., Sora, D., Catarci, T.: Surveying Human Habit Modeling and Mining Techniques in Smart Spaces. Future Internet **11**(1), 23 (2019)
18. Liu, X.: Unraveling and Learning Workflow Models from Interleaved Event Logs. In: 2014 IEEE Int. Conf. on Web Services. pp. 193–200 (2014)
19. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M., Toussaint, P.J.: Guided Process Discovery – A Pattern-based Approach. Inf. Syst. **76** (2018)
20. Marrella, A.: What Automated Planning Can Do for Business Process Management. In: BPM 2017 International Workshops, AI4BBPM'17 (2017)
21. Rovani, M., Maggi, F.M., de Leoni, M., van der Aalst, W.M.: Declarative process mining in healthcare. Expert Systems with Applications **42**(23) (2015)