Simone Agostinelli, Andrea Marrella, and Massimo Mecella

Automated Segmentation of User Interface Logs

Abstract: Robotic Process Automation (RPA) tools are able to capture in dedicated User Interface (UI) logs the execution of high volume routines previously performed by a human user on the interface of a computer system, and then emulate their enactment in place of the user by means of a software (SW) robot. A UI log can record information about several routines, whose actions and events are mixed in some order that reflects the particular order of their execution by the user. In addition, the same user action may belong to different routines, making its automated identification far from being trivial. The issue to automatically understand which user actions contribute to a specific routine inside the UI log is also known as *segmentation*. In this contribution, after discussing in detail the issue of segmentation and all its potential variants, we present a novel segmentation technique that leverages trace alignment in Process Mining for automatically deriving the boundaries of a routine by analyzing the UI logs that keep track of its execution, in order to cluster all user actions associated with the routine itself in well bounded routine traces.

Keywords: Robotic Process Automation, Segmentation of User Interface Logs, Trace Alignment in Process Mining

Simone Agostinelli, Andrea Marrella, Massimo Mecella, Sapienza University of Rome - Department of Computer, Control and Management Engineering, Via Ariosto 25 00185 Rome, Italy, e-mail: {agostinelli,marrella,mecella}@diag.uniroma1.it

Contents

1 Introduction

Robotic Process Automation (RPA) uses *software robots* (or simply *SW robots*) to mimic and replicate the execution of highly routine tasks (in the following, called *routines*) performed by humans in their application's User Interface (UI). SW robots encode, by means of executable scripts, sequences of fine-grained interactions with a computer system such as: opening a file, selecting a field in a form or a cell in a spreadsheet, copy and paste data across cells of a spreadsheet, extract semi-structured data from documents, read and write from/to databases, open emails and attachments, make calculations, etc. [Willcocks, 2016]. A typical routine that can be automated by a SW robot using a RPA tool is transferring data from one system to another via their respective UIs, e.g., copying records from a spreadsheet application into a web-based enterprise information system [Leno et al., 2020b].

Commercial RPA tools allow SW robots to automate a wide range of routines in a record-and-replay fashion. The current practice for identifying the single steps of a routine is by means of interviews, walk-throughs, and detailed observation of workers conducting their daily work [Jimenez-Ramirez et al., 2019]. A recent approach proposed by Bosco et al. [Bosco et al., 2019] makes this identification less time-consuming and error-prone, as it enables to automatically extract from a UI log, which records the UI interactions during a routine enactment, those routine steps to be automated with a SW robot. While this approach is effective in case of UI logs that keep track of single routine executions, i.e., there is an exact 1:1 mapping among a recorded user action and the specific routine it belongs to, it becomes inadequate when the UI log records information about several routines whose actions are mixed in some order that reflects the particular order of their execution by the user. In addition, since the same user action may belong to different routines, the automated identification of those user actions that belong to a specific routine is far from being trivial. The challenge to automatically understand which user actions contribute to which routines inside a UI log is also known as *segmentation* [Agostinelli et al., 2019, Leno et al., 2020b].

In this chapter, after discussing in detail the issue of segmentation and all its potential variants, we present a technique for automatically deriving the boundaries of a routine by analyzing the UI log that keeps track of its execution, in order to cluster all user actions associated with the routine itself in well bounded *routine traces*. A routine trace represents an execution instance of a routine within a UI log. To be more precise, as shown in Figure 1, starting from a UI log previously recorded by a RPA tool and an *interaction model* representing the expected behaviour of a routine performed during an interaction session with

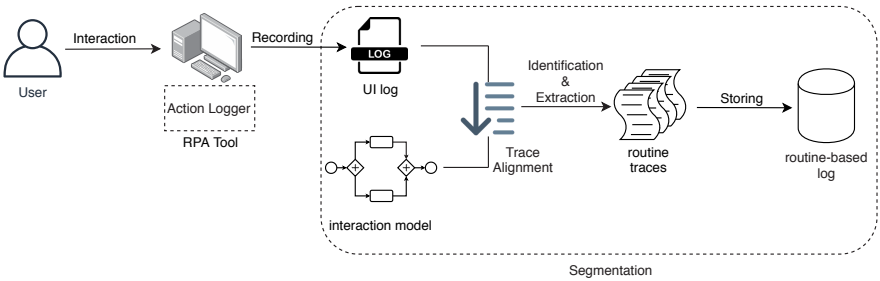

Fig. 1: Overview of the proposed segmentation technique

the UI, we propose a supervised algorithm that leverages *trace alignment* in Process Mining [Adriansyah et al., 2011, de Leoni and Marrella, 2017, de Leoni et al., 2018] to automatically identify and extract the routine traces by the UI log. Such traces are finally stored in a dedicated *routine-based log*, which captures exactly all the user actions happened during many different executions of the routine, thus achieving the segmentation task. 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. 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.

The rest of the chapter is organized as follows. Section 2 introduces a running example that will be used to explain our technique. Section 3 describes the relevant background on interaction models and UI logs. Section 4 illustrates the concept of segmentation and all its peculiarities. Section 5 presents the details of our technique to the automated segmentation of UI logs. Finally, Section 6 discusses the related work, while Section 7 draws conclusions and outlines future works.

2 Running Example

Below, we describe a RPA use case inspired by a real-life scenario at Department of Computer, Control and Management Engineering (DIAG) of Sapienza Universitá di Roma. The scenario concerns the filling of the travel authorization request form made by professors, researchers and PhD students of DIAG for travel requiring prior approval. The request applicant must fill a well-structured Excel spreadsheet (cf. Figure 2(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 (only in this case) 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. Figure 2(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 (simple) 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 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

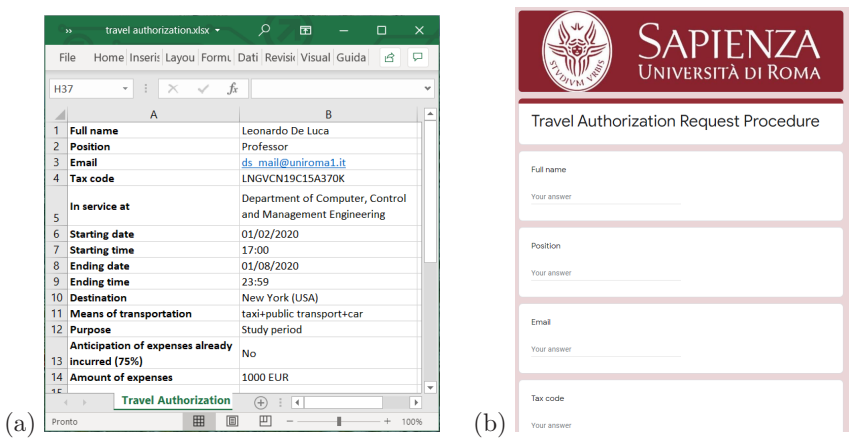

Fig. 2: UIs involved in the running example

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 (this may happen only if the exact sequence of user actions required to achieve the routines' targets on the UI is recorded in the context of controlled training sessions), 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.

3 Preliminaries

In this section, we present some preliminary concepts used throughout the chapter. In Section 3.1, we describe the Petri net modeling language, which will be used to formally specify the interaction models required to represent the structure of the routines of interest, while in Section 3.2 we introduce the notion of UI log.

3.1 Interaction Models as Petri Nets

The research literature is rich of notations for expressing human-computer dialogs as interaction models that allow to see at a glance the structure of a user interaction with a UI [Paternò, 1999, Dix et al., 2004]. Existing notations can be categorized in two main classes: *diagrammatic* and *textual*. Diagrammatic notations include (among the others) various forms of state transition networks (STNs) [Wasserman, 1985], Petri nets [Sy et al., 2000], Harel state charts [Harel, 1987], flow charts [Dix et al., 2004], JSD diagrams [Sutcliffe and Wang, 1991] and ConcurTaskTrees (CTT) [Mori et al., 2002]. Textual notations include regular expressions [Van Den Bos et al., 1983], Linear Temporal Logic (LTL) [Pnueli, 1977], Communicating Sequential Processes (CSPs) [Dignum, 2004], GOMS [John and Kieras, 1996], modal action logic [Campos et al., 2016], BNF and production rules [Feary, 2010].

While there are major differences in expressive power between different notations, an increased expressive power is not always desirable as it may suggest a harder to understand description, i.e., the dialog of a UI can become unmanageable [Dix et al., 2004]. To guarantee a good trade-off between expressive power and understandability of the models, we decided to use *Petri nets* for their specification. Petri nets have proven to be adequate for defining interaction models [Dix et al., 2004, Palanque and Bastide, 1995, Marrella and Catarci, 2018]. They may contain exclusive choices, parallel branches and loops, allowing

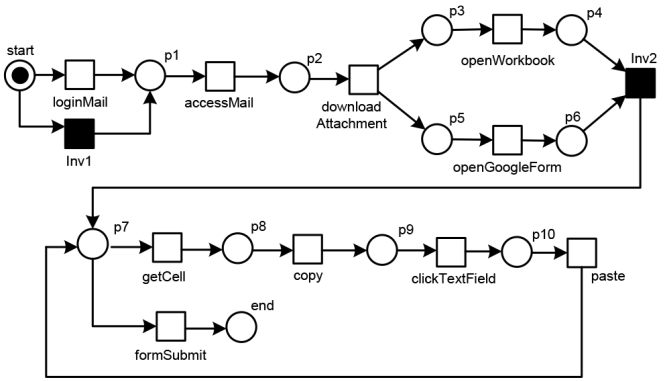

Fig. 3: Interaction model for R1

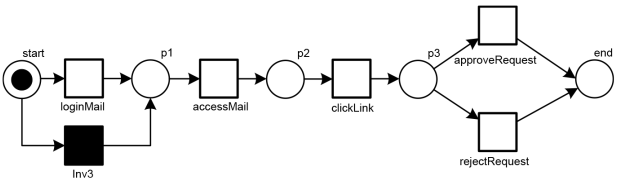

Fig. 4: Interaction model for R2

the representation of extremely complex behaviours in a very compact way. Last but not least, Petri nets provide a formal semantics, which allows to interpret the meaning of an interaction model unambiguously.

From a formal point of view, a Petri net $W = (P, T, S)$ is a directed graph with a set P of nodes called *places* and a set T of *transitions*. The nodes are connected via directed arcs $S \subseteq (P \times T) \cup (T \times P)$. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles. Figures 3 and 4 illustrate the Petri nets used to represent the interaction models of R1 and R2. Transitions are associated with *labels* reflecting the user actions (e.g., system commands executed, buttons clicked, etc.) required to accomplish a routine on the UI. For example, a proper execution of R1 requires a path on the UI made by the following user actions:

- 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;
- paste, to paste the content of the cell into the corresponding text field of the Google form;
- formSubmit, to press the button to finally submit the Google form to the internal database.

Note that, as shown in Figure 3, 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 press the button on the Google form that approves the request;
- rejectRequest, to press the button on the Google form that rejects the request;

Note that the execution of approveRequest and rejectRequest is exclusive. Then, in the interaction models of R1 and R2, there are transitions that do not represent user actions but are needed to correctly represent the structure of such models. These transitions, drawn with a black-filled rectangle, are said to be "invisible", and are not recorded in the UI logs (cf. Inv1, Inv2 and Inv3).

To understand our segmentation technique based on trace alignment in Process Mining, we also need to briefly illustrate the dynamic behaviour of a Petri net, i.e., its operational semantics. Given a transition $t \in T$, ${}^\bullet t$ is used to indicate the set of *input places* of t, which are the places p with a directed arc from p to t (i.e., such that $(p, t) \in S$). Similarly, $t^\bullet$ indicates the set of *output places*, namely the places p with a direct arc from t to p. At any time, a place can contain zero or more *tokens*, drawn as black dots. The state of a Petri net, i.e., its *marking*, is determined by the number of tokens in places. Therefore, a marking m is a function $m : P \to \mathbb{N}$. In any run of a Petri net, the number of tokens in places may change, i.e., the Petri net marking. A transition t is *enabled* at a marking m iff each input place contains at least one token, i.e., $\forall\ p \in {}^\bullet t$, $m(p) > 0$. A transition t can *fire* at a marking m if and only if it is enabled. As result of firing a transition t, one token is "consumed" from each input place

and one is "produced" in each output place. This is denoted as $m \xrightarrow{t} m'$. In the remainder, given a sequence of transition firing $\sigma = \langle t_1, \ldots, t_n \rangle \in T^*$, $m_0 \xrightarrow{\sigma} m_n$ is used to indicate $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m_n$, i.e., m_n is *reachable* from m_0.

Since the executions of a routine have a start and a end, the interaction models represented trough Petri nets need to be associated with an initial and final marking. For example, in both routines of figures 3 and 4, the markings with respectively one token in place *start* or in place *end* are the initial and final marking (and no tokens in any other place). In the remainder of this paper, we assume all Petri nets to be 1-bounded. A Petri net is 1-bounded if in any reachable marking from the initial marking, no place ever contains more than 1 token. One-boundness is not a large limitation as the behavior allowed by interaction models can be represented as 1-bounded Petri nets [Dix et al., 2004, Marrella and Catarci, 2018].

3.2 UI Logs

A single UI log in its raw form consists of a long sequence of user actions recorded during one user session.[1] 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 Figure 5, we show a snapshot of a UI log captured using a dedicated action logger[2] 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 human/sw resource that performed the action, etc.

For the sake of understandability, we assume here that any user action associated to each event recorded in the UI log is mapped at most with one (and only one) Petri net transition, and that the collection of labels associated to the Petri net transitions is defined over the same alphabet as the user actions in the UI log,[3] i.e., the alphabet of user actions in the UI log is a *superset* of that used for defining the labels of Petri net transitions. In the running example, we can recognize in R1 and R2 a universe of user actions of in-

1 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.

2 The working of the action logger is described in [Agostinelli et al., 2020]. The tool is available at:
https://github.com/bpm-diag/smartRPA

3 In [de Leoni and Marrella, 2017], it is shown how these assumptions can be removed.

terest $Z = \{A, B, C, D, E, F, G, H, I, L, M, N, O\}$, 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.

As shown in Figure 5, 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 Figure 5 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 by mistake and then is moved in the right place through a corrective action on the UI. On the other hand, we consider as *noise* all those user actions that do not contribute to the achievement of any routine target, e.g., a window that is resized. In Figure 5, the sequences of user actions that are not surrounded by a blue/red box can be safely labeled as noise.

Based on the foregoing, our segmentation technique aims at extracting from the UI log all those user actions that match a distinguishable pattern as represented by the interaction model of a generic routine R, filtering out redundant actions and noise. To be more specific, any sequence of user actions in the UI log that can be replayed from the initial to the final marking of the Petri net-based interaction model of R is said to be a *routine trace* of R, i.e., a complete execution instance of R within the UI log. For example, a valid routine trace of R1 is $\langle A, B, C, D, E, F, G, H, I, L\rangle$. The interaction model of R1 suggests that valid routine traces are also those ones where: *(i)* 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

	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 ac		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. 5: Snapshot of a UI log captured during the executions of R1 and R2

executed several time before submitting the Google form. On the other hand, two main routine traces can be extracted from R2: $\langle A, B, M, N \rangle$ and $\langle A, B, M, O \rangle$, again with the possibility to skip A, i.e., the access to the client email. Note that, within a routine trace, the concept of time is usually defined in a way that user actions in a trace are sorted according to the timestamp of their occurrence.

We conclude this section by introducing the concept of *routine-based* log as a special container that stores all the routine traces extracted by a UI log and associated to a generic routine R. Thus, the final outcome of our segmentation technique will be a collection of as many routine-based logs as are the interaction models of the routines of interest.

4 Segmentation

Given a UI log consisting of events including user actions having the same granularity[4] and potentially belonging to different routines, in the RPA domain *segmentation* is the task of clustering parts of the log together which belong to the same routine. In a nutshell, the challenge is to automatically understand which user actions contribute to which routines, and organize such user actions in well bounded routine traces Agostinelli et al. [2019], Leno et al. [2020b].

As shown in Section 3.2, in general a UI log stores information about several routines enacted in an interleaved fashion, with the possibility that a specific user action is shared by different routines. Furthermore, actions providing redundant behavior or not belonging to any of the routine under observation may be recorded in the log, generating noise that should be filtered out by a segmentation technique. We can distinguish among three main forms of UI logs, which can be categorized according to the fact that: *(i)* any user action in the log exclusively belongs to a specific routine; *(ii)* the log records the execution of many routines that do not have any user action in common; *(iii)* the log records the execution of many routines, and the possibility exists that some performed user actions are shared by many routines at the same time. In the following, we analyze in detail case by case.

– **Case 1:** This is the case when a UI log captures many executions of the same routine. Of course, in this scenario it is not possible to distinguish

4 The UI logs created by generic action loggers usually consist of low-level events associated one-by-one to a recorded user action on the UI (e.g., mouse clicks, etc.). We will discuss the abstraction issue in Section 6, where state-of-the-art techniques are shown that enable to flatten the content of a log to a same granularity level.

between shared and non-shared user actions by different routines, since the UI log keeps track only of executions associated to a single routine.

Starting from our running example in Section 2, let us consider the simplest case of a UI log U that records a sequence of user actions resulting from many non-interleaved executions of R1: $U = \{A_{11}, B_{11}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11}, \ldots, A_{12}, B_{12}, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12}\}$. For the sake of understandability, we use a numerical subscript ij associated to any user action to indicate that it belongs to the $j-th$ execution of the $i-th$ routine under study. Of course, this information is not recorded in 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. Applying a segmentation technique to the above UI log would trivially produce a routine-based log U_{R1} where the (already well bounded) executions of R1 are organized as different routine traces: $U_{R1} = \{\langle A_{11}, B_{11}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11}, \rangle, \ldots, \langle A_{12}, B_{12}, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12}\rangle\}$.

The same routine-based log U_{R1} would be obtained when the executions of R1 are recorded in an interleaved fashion in the UI log, e.g., $U = \{\boldsymbol{A}, B_{11}, B_{12}, C_{11}, D_{11}, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11}, \ldots\}$. Here, the segmentation task becomes more challenging, not only because the user actions of different executions of a same routine are interleaved among each others, and it is not known a-priori to which execution they belong to, but also for the presence of some user actions that potentially belong *at the same time* to many executions of the routine itself. This is the case of $\boldsymbol{A}$ (that corresponds to loginMail), which can be performed exactly once at the beginning of a user session and can be "shared" by many executions of the same routine.

Another variant is when the execution of a routine is affected by *noise* or *redundant* actions. For example, let us consider the following UI log recorded after many execution of R1: $U = \{A_{11}, B_{11}, C_{11}, \boldsymbol{Y_1}, D_{11}, E_{11}, F_{11}, G_{11}, G_{11}, G_{11}, H_{11}, I_{11}, L_{11}, \ldots, A_{12}, \boldsymbol{Y_{n-1}}, B_{12}, C_{12}, D_{12}, E_{12}, \boldsymbol{Y_n}, F_{12}, G_{12}, H_{12}, I_{12}, I_{12}, I_{12}, L_{12}\}$. This log contains elements of noise, i.e., user actions $\boldsymbol{Y_{k\in\{1,n\}}} \in Z$ (remind that Z is the universe of user actions allowed by a UI log, as introduced in Section 3.2) that are not allowed by R1, and redundant actions like G_{11} (copy action) and I_{12} (paste action) that are unnecessary repeated multiple times. Noise and redundant actions need to be filtered out during the segmentation task because they do not contribute to the achievement of the routine's target.

– **Case 2**: In this case, a UI log captures many executions of different routines, with the assumption that the interaction models of such routines include only transitions associated to user actions that are exclusive for that routines. For example, let us suppose that in both interaction models of R1 and R2 the transitions $\boldsymbol{A}$ and $\boldsymbol{B}$ are not required, and the UI log is as follows: $U = \{C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11}, M_{21}, N_{21}, \ldots, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12}, M_{22}, O_{22}\}$. The output of the segmentation task would consist of two routine-based logs, one per routine, which include the following routine traces:
  – $U_{R1} = \{\langle C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11}\rangle, \ldots, \langle C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12}\rangle\}$
  – $U_{R2} = \{\langle M_{21}, N_{21}\rangle, \ldots, \langle M_{22}, O_{22}\rangle\}$

  Similarly to what already seen in Case 1, it may happen that many executions of the same routine (and, in this case, also of many different routines) are interleaved among each other, and that noise and redundant actions are also recorded in the log. Since we are assuming that there are no shared actions among different routines, the complexity of the segmentation task in presence of interleaved actions, noise and redundancy can be reduced to the case of a single routine, cf. Case 1.
– **Case 3**: In this case, a UI log captures many executions of different routines, and there exist user actions that are shared by such routines. This case perfectly reflects what happens in the running example of Section 2. Let us consider the following UI log: $U = \{\boldsymbol{A}, \boldsymbol{B}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11}, \boldsymbol{B}, M_{21}, N_{21}, \ldots, \boldsymbol{B}, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12}, \boldsymbol{B}, M_{22}, O_{22}\}$. $\boldsymbol{A}$ and $\boldsymbol{B}$ are shared by R1 and R2, as they are included in the interaction models of both routines. 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 executions they belong to. The complexity of the segmentation task here lies in understanding to which routine traces the execution of $\boldsymbol{A}$ and $\boldsymbol{B}$ belong to. The outcome of the segmentation task will be a pair of routine-based logs generated as follows:
  – $U_{R1} = \{\langle \boldsymbol{A}, \boldsymbol{B_{11}}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11}\rangle, \ldots, \langle \boldsymbol{A}, \boldsymbol{B_{12}}, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12}\rangle\}$
  – $U_{R2} = \{\langle \boldsymbol{A}, \boldsymbol{B_{21}}, M_{21}, N_{21}\rangle, \ldots, \langle \boldsymbol{A}, \boldsymbol{B_{22}}, M_{22}, O_{22}\rangle\}$

  Consider that, while $\boldsymbol{A}$ can belong to some routine executions and not to others, making it not possible to understand to which exact routine execution

it can be associated, in case of $\boldsymbol{B}$ it is important to identify the association between its i-th execution and the specific routine execution it belongs to.

The above cases have in common that all the user actions are stored within a single UI log. It may happen that the same routine is spread across multiple UI logs, in particular when there are multiple users that are involved in the execution of the routine on different computer systems. This case can be tackled by "merging" the UI logs where the routine execution is distributed into a single UI log, reducing the segmentation issue to one of the already analyzed cases.

5 Segmentation Technique

In this section, we present our technique to tackle the segmentation issue of UI logs that leverages trace alignment in Process Mining for deriving the boundaries of a routine by analyzing the UI log that keeps track of its execution, in order to cluster all user actions associated with the routine itself in well bounded routine traces. Specifically, in Section 5.1, we first provide the relevant background on trace alignment. Then, in Section 5.2, we present an overview of the general approach underlying our segmentation technique depicting its main steps, and we describe the technical details of the algorithm that implements the technique.

5.1 Alignment between UI Logs and Interaction Models as Petri Nets

Trace alignment [Adriansyah et al., 2011, de Leoni and Marrella, 2017, de Leoni et al., 2018] is a conformance checking technique within Process Mining that is employed to replay the content of any trace of an event log against a process model represented as a Petri net, 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*, together with a *fitness* value, which quantifies how much the trace adheres to the process model. The fitness value can vary from 0 to 1. A fitness value equals to 1 means a perfect matching between the trace and the model.

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 an interaction model W as a Petri net, which allows us to exactly pinpoint where deviations occur. To this aim, the events in U need to be related to transitions

in the model, and vice versa. Building this alignment is far from trivial, since the log may deviate from the model at an arbitrary number of places. To be more specific, we need to relate "moves" in the log to "moves" in the model in order to establish an alignment between an interaction model and a UI log. However, it may be that some of the moves in the log cannot be mimicked by the model and vice versa. We explicitly denote such "no moves" by $\gg$.

Definition 5.1 (Alignment Moves)**.** Let $W = (P, T, S)$ be a Petri net and U be a UI log. A legal *alignment move* for W and U is represented by a pair $(q_U, q_W) \in (T \cup \{\gg\} \times T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$ such that:

- (q_U, q_W) is a *move in log* if $q_U \neq \gg$ and $q_W = \gg$,
- (q_U, q_W) is a *move in model* if $q_U = \gg$ and $q_W \in T$,
- (q_U, q_W) is a *synchronous move* if $q_U = q_W$.

An alignment is a sequence of alignment moves:

Definition 5.2 (Alignment)**.** Let $W = (P, T, S)$ be a Petri net with an initial marking and final marking denoted with m_i and m_f. Let also U be a UI log. Let Γ_W be the universe of all alignment moves for W and U. Sequence $\gamma \in \Gamma_W^*$ is an *alignment* of W and U if, ignoring all occurrences of $\gg$, the projection on the first element yields U and the projection on the second yields a sequence $\sigma'' \in T^*$ such that $m_i \xrightarrow{\sigma''} m_f$.

A move in log for a transition t indicates that t occurred when not allowed; a move in model for a visible transition t indicates that t did not occur, when, conversely, expected. Many alignments are possible for the same UI log and a Petri net. For example, Figure 6 shows two possible alignments for a UI log consisting of the following sequence of user actions $\langle A, B, M, N \rangle$ and the Petri net in Figure 4, representing the interaction model of R2. Note how moves are represented vertically. For example, as shown in Figure 6, the first move of γ_1 is (A, A), i.e., a *synchronous move* of A, while the first and second move of γ_2 are a move in log and model, respectively. We aim at finding a complete alignment of U and W with minimal number of deviations (i.e., of moves in log/model) for visible transitions, also known in literature as *optimal alignments*. With reference to the alignments in Figure 6, γ_1 have four synchronous moves and γ_2 have one move in log for visible transitions and one move in model for the invisible transition $Inv3$ (that does not count for the computation

$$\gamma_1 = \begin{array}{|c|c|c|c|} \hline A & B & M & N \\ \hline A & B & M & N \\ \hline \end{array}$$

$$\gamma_2 = \begin{array}{|c|c|c|c|c|} \hline A & \gg & B & M & N \\ \hline \gg & Inv3 & B & M & N \\ \hline \end{array}$$

Fig. 6: Alignments of $\langle A, B, M, N \rangle$ and the Petri net in Figure 4.

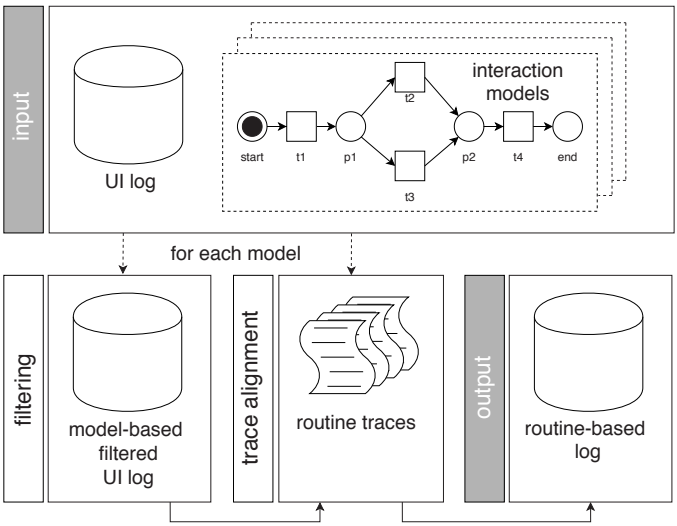

Fig. 7: Overview of the general approach underlying the proposed segmentation technique

of the fitness value). As a consequence, γ_1 is an optimal alignment and can be returned. Note that its fitness value is exactly equal to 1, since it is consists only of synchronous moves enabling U to be completely replayed from the initial to the final marking of W. For the sake of simplicity, we are assuming here that all the deviations have the same severity. However, the severity of a deviation can be customized on a ad-hoc basis [de Leoni and Marrella, 2017].

5.2 The General Approach and the Segmentation Algorithm

The general approach underlying our segmentation techniques consists of two methodological phases, *filtering* and *trace alignment*, to be applied in sequence, as shown in Figure 7. Algorithm 1 shows the technical details of the algorithm that concretely implements such phases.

The algorithm takes in input a UI log U, a set of interaction models W_{set} and returns a set of routine-based logs U_{set}. For each interaction model $w \in W_{set}$ (one for each routine of interest) represented as Petri nets, the algorithm performs the following steps:

1. **Filtering**: The filtering phase is used to filter out noisy actions from the UI log. Specifically, for each interaction model $w \in W_{set}$, a local copy of the UI

Algorithm 1: Algorithm implementing our segmentation technique

Parameters: a UI log U, a set of interaction models W_{set}
Result: A set U_{set} of routine-based logs

```
1  U_set ← ∅;
2  forall w ∈ W_set do
3      U^w ← duplicate(U);
4      U^w_φ ← filter(U^w);
5      U^w_R ← ∅;
6      repeat
7          γ^opt ← trace alignment (U^w_φ, w);
8          γ^opt_sm ← extract(γ^opt);
9          if γ^opt_sm is not empty then
10             create a trace τ_sm from γ^opt_sm;
11             create a temporary UI log U^w_sm from τ_sm;
12             fitness ← compute fitness from trace alignment (U^w_sm, w);
13             if fitness is 1 then
14                 add τ_sm to U^w_R;
15             else
16                 discard τ_sm;
17             end
18             remove the events associated to τ_sm from U^w_φ;
19         end
20     until γ^opt_sm is not empty;
21     add U^w_R to U_set;
22 end
23 return U_set
```

log U^w is created (`line 3`). Then, all user actions that appear in U^w but that can not be replayed by any transition of w are removed from U^w. The output of this step is a *model-based filtered UI log* U^w_ϕ (`line 4`). Working with U^w_ϕ rather than with U^w will allow us to apply the trace alignment technique neglecting all the potential moves in log with user actions that could never be replayed by w. As a consequence, this will drastically reduce the number of alignment steps required to find optimal alignments, and at the same time optimize the performance of the algorithm. Before moving to the next step, a new routine-based log U^w_R is initialized (`line 5`).

2. **Trace Alignment**: The second step consists of applying the trace alignment technique discussed in Section 5.1 for any interaction model $w \in W_{set}$ and

its associated model-based filtered UI log U^w_ϕ. This enables to extract from U^w_ϕ all those user actions that match a distinguishable pattern with w in the form of an optimal alignment γ^{opt} (`line 7`). Trace alignment allows to pinpoint the *synchronous moves* between U^w_ϕ and w. If they exist, the user actions involved in synchronous moves are extracted and stored into γ^{opt}_{sm} (`line 8`). Note that focusing just on synchronous moves allows us to exclude all redundant user actions from the analysis. Then, the algorithm:

(a) creates a trace τ_{sm} consisting of the user actions associated with the synchronous moves stored in γ^{opt}_{sm} (`line 10`);
(b) creates a (temporary) UI log U^w_{sm} containing only the trace τ_{sm} (`line 11`), which is required to properly run (again) trace alignment;
(c) performs a new alignment between U^w_{sm} and w with the goal to compute the fitness value (`line 12`).

In case the fitness value is equal to 1, this means that the U^w_{sm} (and, consequently, τ_{sm}) can be replayed from the start to the final marking of w, making τ_{sm} a valid routine trace of w. In such a case, τ_{sm} is stored into U^w_R (`line 14`) and all the events associated to the synchronous moves in τ_{sm} are removed by U^w_ϕ (`line 18`). On the contrary, a fitness value lower than 1 indicates the presence of at least one move in the model in τ_{sm} with respect to w, i.e., τ_{sm} can not be completely replayed by w and is not a valid routine trace, meaning that we can discard it (`line 16`).

The above two steps can be repeated until γ^{opt}_{sm} is not empty (`line 20`), i.e., until there are synchronous moves in the computed alignment. At the end of the iteration, the routine-based log U^w_R is stored into U_{set} (`line 21`), and the algorithm starts to analyze the next interaction model into W_{set}. In conclusion, the algorithm computes a number of routine-based logs equal to the number of interaction models under study.

It is worth to notice that: (*i*) for the computation of the trace alignment, the algorithm relies on the highly-scalable planning-based alignment technique implemented in our previous work [de Leoni and Marrella, 2017], which we have properly customized for our purposes; and (*ii*) the algorithm is able to achieve cases 1, 2 and 3 discussed in Section 4, except when there are interleaved executions of shared user actions by different routines. In that case, the risk exists that a shared user action is associated to a wrong routine execution, i.e., clustered in a wrong routine trace.

5.2.1 An execution instance of the segmentation algorithm

We show now an execution instance of Algorithm 1 applied to the following UI log $U = \{\boldsymbol{A}, \boldsymbol{B}, C, \boldsymbol{Y_1}, D, E, F, G, G, G, H, I, L, \boldsymbol{B}, M, N, \ldots, \boldsymbol{B}, \boldsymbol{Y_{n-1}}, C, D, E, \boldsymbol{Y_n}, F, G, H, I, I, I, L, \boldsymbol{B}, M, O\}$. 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. In addition, $\boldsymbol{A}$ and $\boldsymbol{B}$ are shared by R1 and R2, as they are included in the interaction models of both routines. In particular, $\boldsymbol{A}$ 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.

The algorithm takes in input: *(i)* the UI log U and *(ii)* the interaction models of R1 and R2, and computes a set of routine-based logs U_{set} by executing the following steps:

- (`line 1`): initializes the set of interaction models U_{set};
- (`line 2`): iterates on the interaction models of R1 and R2. For the sake of space, we focus only on the steps computed in the case of R1;
- (`line 3`): creates a local copy of U, namely U^w;
- (`line 4`): filters U^w from noise, so $U^w_\phi = \{\boldsymbol{A}, \boldsymbol{B}, C, D, E, F, G, G, G, H, I, L, \boldsymbol{B}, \ldots, \boldsymbol{B}, C, D, E, F, G, H, I, I, I, L, \boldsymbol{B}\}$. In this step, the user actions $\boldsymbol{Y_{k\in\{1,n\}}}$ and M, N, M (being exclusively related to R2) are filtered out by the log. On the other hand, redundant actions still remain in the log;
- (`line 5`): initializes the routine-based log U^w_R;
- (`line 7`): computes the trace alignment between U^w_ϕ and the interaction model of R1, namely w.

$$\gamma^{opt} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline A & B & C & D & E & F & G & G & G & H & I & L & B & \ldots & B \\ \hline A & B & C & D & E & F & G & \gg & \gg & H & I & L & \gg & \ldots & \gg \\ \hline \end{array}$$

- (`line 8`): extracts the synchronous moves from γ^{opt} into γ^{opt}_{sm}.
- (`line 9`): evaluates to $True$, as γ^{opt}_{sm} is not empty;
- (`line 10`): computes the trace τ_{sm} starting from γ^{opt}_{sm}. So $\tau_{sm} = \langle \boldsymbol{A}, \boldsymbol{B}, C, D, E, F, G, H, I, L \rangle$;
- (`line 11`): adds the trace τ_{sm} in U^w_{sm};
- (`line 12`): computes trace alignment between U^w_{sm} and w.

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline A & B & C & D & E & F & G & H & I & L \\ \hline A & B & C & D & E & F & G & H & I & L \\ \hline \end{array}$$

U^w_{sm} can be replayed without deviations from the start to the final marking of w, meaning a perfect fitness between the log and the interaction model;
- (`line 13`): evaluates to $True$, as the fitness of the alignment (cf. `line 12`) is equal to 1;
- (`line 14`): adds τ_{sm} in U^w_R, i.e., τ_{sm} is recognized as a valid routine trace;
- (`line 18`): removes all the events associated with the synchronous moves in τ_{sm} from U^w_ϕ. Thus, $U^w_\phi = \{G, G, \boldsymbol{B}, \ldots, \boldsymbol{B}, C, D, E, F, G, H, I, I, I, L, \boldsymbol{B}\}$;
- (`line 20`): Since γ^{opt}_{sm} is not empty, the algorithm comes back to `line 6`. After repeating the above steps from `line 7` to `line 14`, the algorithm discovers a second routine trace $\tau_{sm} = \langle \boldsymbol{B}, C, D, E, F, G, H, I, L \rangle$ and adds it in U^w_R. Like before, all the events associated with the synchronous moves in τ_{sm} are removed from U^w_ϕ. Thus, $U^w_\phi = \{G, G, \boldsymbol{B}, \ldots, I, I, \boldsymbol{B}\}$. The subsequents iterations of the algorithm do not discover new routine traces for R1, since the only synchronous moves extracted in the various alignment steps between w and U^w_ϕ are the $\boldsymbol{B}s$, Gs and Is that are discarded (due to the fitness value of γ^{opt}_{sm} that is < 1). It is worth to notice that redundant user actions G and I are removed from U^w_ϕ during these iterations. The algorithm ends to iterate when γ^{opt}_{sm} is empty, that is, when there are no more synchronous moves to extract;
- (`line 21`): After the last iteration ends, the routine-based log U^w_R is stored into U_{set}, and the algorithm starts to analyze the interaction model of R2.

The outcome of the segmentation task will be a set of routine-based logs (in this case two, since the number of interaction models under study is two) generated as follows: $U_{set} = \{\{\langle \boldsymbol{A}, \boldsymbol{B_{11}}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, I_{11}, L_{11} \rangle, \ldots, \langle \boldsymbol{B_{12}}, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, H_{12}, I_{12}, L_{12} \rangle\}, \{\langle \boldsymbol{A}, \boldsymbol{B_{21}}, M_{21}, N_{21} \rangle, \ldots, \langle \boldsymbol{B_{22}}, M_{22}, O_{22} \rangle\}\}$.

6 Related work

In the field of RPA, segmentation is an issue still not so explored, since the current practice adopted by commercial RPA tools for identifying the routine steps often consists of detailed observations of workers conducting their daily work. Such observations are then "converted" in explicit flowchart diagrams [Jimenez-Ramirez et al., 2019], which are manually modeled by expert RPA analysts to depict all the potential behaviours (i.e., the traces) of a specific

routine. In this setting, as the routine traces have been already (implicitly) identified, segmentation can be neglected.

On the other hand, following a similar trend that has been occurring in the Business Process Management (BPM) domain [Marrella et al., 2018, Marrella, 2019], the research on RPA is moving towards the application of intelligent techniques to automate all the steps of a RPA project, as proven by many recent works in this direction (see below). In this context, segmentation can be considered as one of the "hot" key research effort to investigate [Agostinelli et al., 2019, Leno et al., 2020b].

With regard to previous works, even if more focused on traditional business processes in BPM rather than on RPA routines, [Fazzinga et al., 2018] comes closest to our technique. This work proposes a probabilistic interpretation approach that employs predefined behavioural models to establish which process activities (generated by an arbitrary number of process instances) belong to which process model. Similarly to [Fazzinga et al., 2018], our segmentation technique falls in the supervised category, as it can be applied only in presence of pre-defined interaction models in input. On the other hand, differently by [Fazzinga et al., 2018], our approach is not probabilistic, but is thoroughly quantitative, based on the computation of fitness values.

In [Bosco et al., 2019], the authors provide a method to analyze UI logs in order to discover routines that are fully deterministic and thus amenable for automation. The method combines a technique for compressing a set of sequences of user actions into an acyclic automaton using rule mining techniques and data transformations. However, this approach is effective in case of UI logs that keep track of well-bounded routine executions, and becomes inadequate when the UI log records information about several routines whose actions are potentially interleaved.

In [Leno et al., 2020a], the authors propose a technique to identify candidate routines to be automated starting from an unsegmented UI log. The technique is able to discover the execution traces of a specific routine relying on the automated synthesis of a control-flow graph that describes the observed directly-follow relations between the user actions. The technique in [Leno et al., 2020a] is effective to tackle some simple variants of Case 1 and Case 2 (cf. Section 4), while loses in accuracy in presence of recurrent noise and interleaved routine executions.

In [Gao et al., 2019], the authors propose a self-learning approach to automatically detect high-level RPA-rules from captured historical low-level behaviour logs. An if-then-else deduction logic is used to infer rules from behaviour logs by learning relations between the different routines performed in the past. Then, such rules are employed to facilitate the SW robots instantiation. A similar ap-

proach is adopted in [Le and Gulwani, 2014], where the *FlashExtract* framework is presented. FlashExtract allows to extract relevant data from semi-structured documents using input-output examples, from which one can derive some relations underlying the working of a routine. Differently from our segmentation technique, which is able to extract the routine traces, i.e., the concrete behaviours of a routine, the above works allow to discover partial views of the working of a routine.

There exist other approaches that focus on learning the anatomy of routines not analyzing UI logs but from natural language descriptions of the procedures underlying such routines. In this direction, the work [Ito et al., 2020] defines a new grammar for complex workflows with chaining machine-executable meaning representations for semantic parsing. In [Leopold et al., 2018], the authors provide an approach to learn activities from text documents employing supervised machine learning techniques such as feature extraction and support vector machine training. Similarly, in [Han et al., 2020] the authors adopt a deep learning approach based on Long Short-Term Memory (LSTM) recurrent neural networks to learn the relationship between user actions.

Moreover, even if the target is not to 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. In [Baier et al., 2014], the authors present a semi-automated approach for finding a set of candidate mappings between the user actions stored in a UI log and instances of high-level activities. This scenario requires a human-in-the-loop to be involved in the filtering phase to resolve the ambiguities on the mapping between user actions and activities. The works [Baier et al., 2015] proposes 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 interaction model. This method leverages constraint-satisfaction techniques to reduce the set of candidate mappings. Similarly, in [Ferreira et al., 2014], starting from a state-machine model describing the routine of interest in terms of high-level activities, the authors 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, in [Mannhardt et al., 2018], a technique is presented that maps 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 (corresponding to our UI actions) to multiple routines.

In addition to the above supervised techniques, there are unsupervised techniques [Günther et al., 2009, Bose and van der Aalst, 2009, Folino et al., 2014, 2015] that try to convert each sequence of user actions into a sequence of higher level activities without any background knowledge on the structure of the routines whose execution generates the UI log. Starting from the UI log, such works exploit clustering techniques to aggregate user actions into clusters, where any cluster represents a high-level activity associated to a well-defined sequence of user actions. The final outcome is an abstracted view of the UI log, obtained by replacing each user action with a label identifying the cluster containing it.

7 Conclusion

To tackle the *segmentation* challenge, in this chapter we have presented a technique, coupled with a supervised algorithm, leveraging trace alignment in Process Mining to identify sequences of user actions in a UI log that belong to specific routine executions, clustering them in well bounded routine traces. Our work is based on a supervised assumption since we know a-priori the structure of the routines, namely the interaction models. Despite this limitation, we consider this contribution as an important first step towards the development of a more complete and unsupervised technique to the segmentation of UI logs.

In this direction, as a future work, we are going to perform a robust evaluation of the algorithm on synthetic and real-world case studies with heterogeneous UI logs. In addition, we aim at relaxing the supervised assumption in different ways: *(i)* by employing declarative rules [Pesic et al., 2007] rather than Petri nets to represent interaction models, allowing us to reason over a partial knowledge of the working of the routines; *(ii)* by investigating *sequential pattern mining* techniques [Dong, 2009] to examine frequent sequences of user actions having common data attributes; *(iii)* by analyzing *web log mining* techniques [Mobasher and Nasraoui, 2011], whose input is a set of clickstreams and the goal is to extract sessions where a user engages with a web application to fulfill a goal; *(iv)* by employing *machine learning* techniques to automatically identify sequences of user actions associated with a routine execution without any previous knowledge of the routines' structure.

Bibliography

Arya Adriansyah, Natalia Sidorova, and Boudewijn F. van Dongen. Cost-Based Fitness in Conformance Checking. In *Int. Conf. on Application of Concurrency to System Design*, pages 57–66. IEEE, 2011.

Simone Agostinelli, Andrea Marrella, and Massimo Mecella. Research Challenges for Intelligent Robotic Process Automation. In *Business Process Management Workshops - BPM 2019 Int. Workshops*, pages 12–18, 2019.

Simone Agostinelli, Marco Lupia, Andrea Marrella, and Massimo Mecella. Automated Generation of Executable RPA Scripts from User Interface Logs. In *Business Process Management: Blockchain and Robotic Process Automation Forum - BPM 2020 Blockchain and RPA Forum*, 2020.

Thomas Baier, Jan Mendling, and Mathias Weske. Bridging abstraction layers in process mining. *Information System*, 46:123–139, 2014.

Thomas Baier, Andreas Rogge-Solti, Jan Mendling, and Mathias Weske. Matching of events and activities: an approach based on behavioral constraint satisfaction. In *ACM Symp. on Applied Computing*, pages 1225–1230, 2015.

Antonio Bosco, Adriano Augusto, Marlon Dumas, Marcello La Rosa, and Giancarlo Fortino. Discovering Automatable Routines From User Interaction Logs. In *Int. Conf. on Business Process Management (BPM'19), Forum track, Vienna, Austria*, pages 144–162. Springer, 2019.

RP Jagadeesh Chandra Bose and Wil MP van der Aalst. Abstractions in Process Mining: A Taxonomy of Patterns. In *Int. Conf. on Business Process Management*, pages 159–175. Springer, 2009.

José Creissac Campos, Manuel Sousa, Miriam C Bergue Alves, and Michael D Harrison. Formal Verification of a Space System's User Interface with the IVY Workbench. *IEEE SMC*, 46(2), 2016.

Massimiliano de Leoni and Andrea Marrella. Aligning Real Process Executions and Prescriptive Process Models through Automated Planning. *Expert System with Application*, 82: 162–183, 2017.

Massimiliano de Leoni, Giacomo Lanciano, and Andrea Marrella. Aligning Partially-Ordered Process-Execution Traces and Models Using Automated Planning. In *Proc. of the Twenty-Eight Int. Conf. on Automated Planning and Scheduling (ICAPS 2018)*, pages 321–329, 2018.

MV Dignum. *A model for organizational interaction: based on agents, founded in logic*. SIKS, 2004.

Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. Human-Computer Interaction. *Pearson*, 2004.

Guozhu Dong. *Sequence Data Mining*. Springer-Verlag, 2009. ISBN 1441943528, 9781441943521.

Bettina Fazzinga, Sergio Flesca, Filippo Furfaro, Elio Masciari, and Luigi Pontieri. Efficiently interpreting traces of low level events in business process logs. *Information Systems*, 73: 1–24, 2018.

Michael S Feary. A toolset for supporting iterative human automation: Interaction in design. *NASA Ames Research Center*, 2010.

Diogo R Ferreira, Fernando Szimanski, and Célia G Ralha. Improving process models by mining mappings of low-level events to high-level activities. *Intelligent Information Systems*, 43(2):379–407, 2014.

Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Mining Predictive Process Models out of Low-level Multidimensional Logs. In *Int. Conf. on Advanced Information Systems Engineering*, pages 533–547. Springer, 2014.

Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Mining multi-variant process models from low-level logs. In *Int. Conf. on Business Information Systems*, pages 165–177. Springer, 2015.

Junxiong Gao, Sebastiaan J. van Zelst, Xixi Lu, and Wil M. P. van der Aalst. Automated Robotic Process Automation: A Self-Learning Approach. In *On the Move to Meaningful Internet Systems: OTM 2019 Conf.*, pages 95–112. Springer, 2019. ISBN 978-3-030-33246-4.

Christian W Günther, Anne Rozinat, and Wil MP van Der Aalst. Activity Mining by Global Trace Segmentation. In *Int. Conf. on Business Process Management*, pages 128–139. Springer, 2009.

Xue Han, Lianxue Hu, Yabin Dang, Shivali Agarwal, Lijun Mei, Shaochun Li, and Xin Zhou. Automatic Business Process Structure Discovery using Ordered Neurons LSTM: A Preliminary Study. *arXiv CoRR abs/2001.01243*, 2020. URL https://arxiv.org/abs/2001.01243.

David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

Nobuhiro Ito, Yuya Suzuki, and Akiko Aizawa. From Natural Language Instructions to Complex Processes: Issues in Chaining Trigger Action Rules. *arXiv CoRR abs/2001.02462*, 2020. URL https://arxiv.org/abs/2001.02462.

Andres Jimenez-Ramirez, Hajo A. Reijers, Irene Barba, and Carmelo Del Valle. A Method to Improve the Early Stages of the Robotic Process Automation Lifecycle. In *31st Int. Conf. on Advanced Information Systems Engineering (CAiSE'19)*, pages 446–461, 2019. ISBN 978-3-030-21290-2.

Bonnie E John and David E Kieras. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM TOCHI*, 3(4), 1996.

Vu Le and Sumit Gulwani. FlashExtract: A Framework for Data Extraction by Examples. In *ACM SIGPLAN PLDI '14*, pages 542–553, 2014.

Volodymyr Leno, Adriano Augusto, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Artem Polyvyanyy. Identifying Candidate Routines for Robotic Process Automation from Unsegmented UI Logs. In *2nd International Conference on Process Mining, ICPM'20*, pages 153–160, 2020a.

Volodymyr Leno, Artem Polyvyanyy, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. Robotic Process Mining: Vision and Challenges. *Business & Information Systems Engineering*, 2020b.

Henrik Leopold, Han van der Aa, and Hajo A Reijers. Identifying Candidate Tasks for Robotic Process Automation in Textual Process Descriptions. In *Int. Conf. on Bus. Proc. Mod., Dev. and Supp. (BPMDS'18)*. 2018.

Felix Mannhardt, Massimiliano de Leoni, Hajo A Reijers, Wil MP van der Aalst, and Pieter J Toussaint. Guided process discovery–a pattern-based approach. *Information Systems*, 76:1–18, 2018.

Andrea Marrella. Automated Planning for Business Process Management. *J. Data Semant.*, 8 (2):79–98, 2019.

Andrea Marrella and Tiziana Catarci. Measuring the Learnability of Interactive Systems Using a Petri Net Based Approach. In *2018 Designing Interactive Systems Conf.*, DIS '18, pages 1309–1319. ACM, 2018. ISBN 978-1-4503-5198-0.

Andrea Marrella, Massimo Mecella, and Sebastian Sardiña. Supporting adaptiveness of cyber-physical processes through action-based formalisms. *AI Commun.*, 31(1):47–74, 2018.

B Mobasher and O Nasraoui. Web Usage Mining, Web Data Mining. Exploring Hyperlinks, Contents, and Usage Data, B. Liu, 2011.

Giulio Mori, Fabio Paternò, and Carmen Santoro. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Trans. Sof. Eng.*, 28(8), 2002.

Philippe A Palanque and Remi Bastide. Petri Net Based Design of User-Driven Interfaces Using the Interactive Cooperative Objects Formalism. In *Interactive Systems: Design, Specification, and Verification*. Springer, 1995.

Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, 1st edition, 1999. ISBN 1852331550.

Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *IEEE Int. Enterprise Distributed Object Computing Conf. (EDOC 2007)*, pages 287–300, 2007.

Amir Pnueli. The temporal logic of programs. In *F. of Comp. Sc.*, 1977.

Alistair G. Sutcliffe and I Wang. Integrating Human Computer Interaction with Jackson System Development. *The Computer journal*, 34(2), 1991.

Ousmane Sy, Rémi Bastide, Philippe Palanque, D Le, and David Navarre. PetShop: a CASE tool for the Petri Net based specification and prototyping of CORBA systems. In *Petri nets*, volume 2000, 2000.

Jan Van Den Bos, Marinus J. Plasmeijer, and Pieter H. Hartel. Input–Output Tools: A Language Facility for Interactive and Real-Time Systems. *IEEE Trans. Sof. Eng.*, (3), 1983.

Anthony I. Wasserman. Extending State Transition Diagrams for the Specification of Human-Computer Interaction. *IEEE Trans. on Soft. Eng.*, (8):699–713, 1985.

Leslie Willcocks. *Service automation: robots and the future of work*. Steve Brookes Publishing, Warwickshire, United Kingdom, 2016. ISBN 0956414567.