Bus Inf Syst Eng 63(3):301–314 (2021)
https://doi.org/10.1007/s12599-020-00641-4

RESEARCH NOTES

Robotic Process Mining: Vision and Challenges

Volodymyr Leno · Artem Polyvyanyy · Marlon Dumas · Marcello La Rosa · Fabrizio Maria Maggi

Received: 21 March 2019 / Accepted: 17 February 2020 / Published online: 24 March 2020

Abstract Robotic process automation (RPA) is an emerging technology that allows organizations automating repetitive clerical tasks by executing scripts that encode sequences of fine-grained interactions with Web and desktop applications. Examples of clerical tasks include opening a file, selecting a field in a Web form or a cell in a spreadsheet, and copy-pasting data across fields or cells. Given that RPA can automate a wide range of routines, this raises the question of which routines should be automated in the first place. This paper presents a vision towards a family of techniques, termed robotic process mining (RPM), aimed at filling this gap. The core idea of RPM is that repetitive routines amenable for automation can be discovered from logs of interactions between workers and Web and desktop applications, also known as user interactions (UI) logs. The paper defines a set of basic concepts underpinning RPM and presents a pipeline of processing steps that would allow an RPM tool to generate RPA scripts from UI logs. The paper also discusses research challenges to realize the envisioned pipeline.

Accepted after two revisions by Jörg Becker.

V. Leno (✉) · A. Polyvyanyy · M. La Rosa
University of Melbourne, Parkville, VIC 3010, Australia
e-mail: vleno@student.unimelb.edu.au

A. Polyvyanyy
e-mail: artem.polyvyanyy@unimelb.edu.au

M. La Rosa
e-mail: marcello.larosa@unimelb.edu.au

M. Dumas · F. M. Maggi
University of Tartu, Liivi 2, 50409 Tartu, Estonia
e-mail: marlon.dumas@ut.ee

F. M. Maggi
e-mail: f.m.maggi@ut.ee

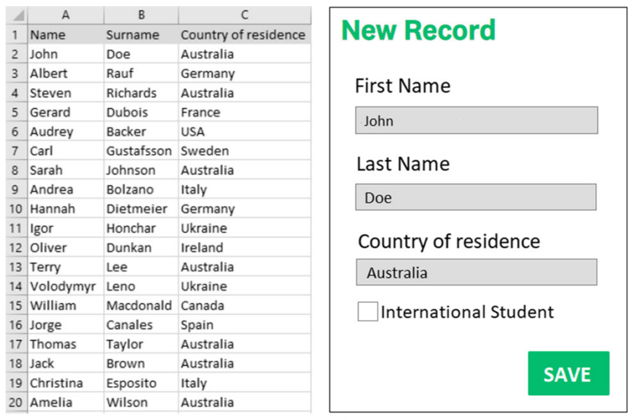
Keywords Robotic process automation · Process mining · Robotic process mining

1 Introduction

Robotic process automation (RPA) tools, such as UiPath Enterprise RPA Platform[1] and Automation Anywhere Enterprise RPA,[2] allow organizations to automate repetitive work by executing scripts that encode sequences of fine-grained interactions with Web and desktop applications (van der Aalst et al. 2018). A typical clerical task that can be automated using an RPA tool is transferring data from one system to another via the user interfaces of these systems. For example, Fig. 1 shows a spreadsheet with student records that need to be transferred one by one into a Web-based study information system. This task involves, for each row in the spreadsheet, selecting the cells, copying the value in a selected cell to the corresponding field in the Web form, and submitting the form after a row has been processed. Routines such as this one can be encoded in an RPA script and executed by an instance of an RPA tool's runtime environment, also known as an *RPA software robot* (or *RPA bot* for short).

A number of case studies have shown that RPA technology can lead to improvements in efficiency and data quality in business processes involving clerical work (Lacity and Willcocks 2016; Aguirre and Rodriguez 2017). However, while existing RPA tools are able to automate a wide range of routines, they cannot determine which routines are candidates for automation in the first place.

[1] https://www.uipath.com/.
[2] https://www.automationanywhere.com/.



	A	B	C
1	Name	Surname	Country of residence
2	John	Doe	Australia
3	Albert	Rauf	Germany
4	Steven	Richards	Australia
5	Gerard	Dubois	France
6	Audrey	Backer	USA
7	Carl	Gustafsson	Sweden
8	Sarah	Johnson	Australia
9	Andrea	Bolzano	Italy
10	Hannah	Dietmeier	Germany
11	Igor	Honchar	Ukraine
12	Oliver	Dunkan	Ireland
13	Terry	Lee	Australia
14	Volodymyr	Leno	Ukraine
15	William	Macdonald	Canada
16	Jorge	Canales	Spain
17	Thomas	Taylor	Australia
18	Jack	Brown	Australia
19	Christina	Esposito	Italy
20	Amelia	Wilson	Australia

New Record

First Name: John

Last Name: Doe

Country of residence: Australia

☐ International Student

SAVE

(a) Student records spreadsheet **(b)** New Record creation form

Fig. 1 Extract of spreadsheet with student data that needs to be transferred to a Web form

The current practice for identifying candidate routines for RPA is through interviews, walk-throughs, and detailed observation of workers conducting their daily work, either in situ or using video-recordings (Agaton and Swedberg 2018). These empirical investigation methods allow analysts to identify candidate routines for automation and to assess the potential benefits and costs of automating the identified routines. However, these methods are time-consuming and, therefore, face scalability limitations in organizations where the number of routines is very high.

In this position paper, we lay down a vision for a new class of tools, namely Robotic Process Mining (RPM) tools, capable of discovering automatable routines from logs of interactions between workers and Web and desktop applications. The envisioned RPM tools take as input logs of user interactions with the applications (so-called *user interaction logs*, or *UI logs*) that contain event records, such as selecting a field or cell, copying and pasting, and editing fields or cells. Given a UI log, RPM tools aim at identifying automatable routines and their boundaries, collect variants of each identified routine, standardize and streamline the identified variants, and discover an executable specification corresponding to a streamlined and standardized variant of the routine. The routines produced as output should be defined in a platform-independent language that can be compiled into a script and executed in an RPA tool.

In this way, RPM tools will assist analysts in drawing a systematic inventory of candidate routines for automation. This input is useful in environments where the number of routines is too large for purely manual identification. We envision that the identified candidate routines will then be analyzed in terms of potential benefit and automation costs using a combination of automatically derived attributes (e.g., frequency, number of steps in the routines, amenability to automation) in conjunction with domain knowledge (e.g., potential financial benefits of automating the routines). Once the candidate routines for RPA have been selected, RPM will then help analysts to produce executable specifications of routines (or sub-routines), which can be used as a starting point for the automation effort.

The paper defines a set of concepts underpinning RPM and presents a pipeline of processing steps that would allow an RPM tool to generate RPA scripts from UI logs. Based on this pipeline, the paper then discusses research challenges and points out possible approaches to address these challenges.

The rest of the paper is structured as follows. Section 2 presents the proposed RPM framework. Section 3 discusses challenges and guidelines to realize this framework. Section 4 positions RPM with respect to related fields, and Sect. 5 draws conclusions and acknowledges ethical considerations.

2 RPM Framework

In this section, we clarify the context and scope of RPM and propose a conceptual framework for RPM as well as a pipeline that decomposes the RPM problem into relatively independent steps.

2.1 Context and Scope

Several partially overlapping definitions of RPA can be found in the research and industry literature. For example, Aguirre and Rodriguez (2017) define RPA as a category of software tools designed "to automate rules-based business processes that involve routine tasks, structured data, and deterministic outcomes". Meanwhile, van der Aalst et al. (2018) define RPA as "an umbrella term for tools that operate on the user interface of other computer systems in the way a human would do". On the other hand, Gartner (Tornbohm 2017) defines RPA as a class of tools that perform [if, then, else] statements on structured data, typically using a combination of user interface interactions, or by connecting to APIs to drive client servers, mainframes or HTML code. An RPA tool operates by mapping a process to the RPA tool language to drive the software robot, with runtime allocated to execute the script by a control dashboard.

Three elements come out from the above definitions. First, RPA tools are designed to automate routine tasks that involve structured data, that are driven by rules (e.g., if-then-else rules), and that have "deterministic outcomes". Second, RPA tools are able to execute tasks that involve user interactons, in addition to other operations accessible via APIs (in any case, automated actions). Third, in RPA tools, it is possible to specify scripts and to operate (i.e., to run and monitor via control dashboards) software bots that execute these scripts.

By synthesizing these elements, we define RPA as *a class of tools that allow users to specify deterministic routines involving structured data, rules, user interface interactions, and operations accessible via APIs. These routines are encoded as scripts that are executed by software bots, operated via control dashboards.*

Depending on how the control dashboard is used, we can distinguish two RPA use cases: *attended* and *unattended* (Tornbohm 2017). In attended use cases, the bot is triggered by a user. During its execution, an attended bot may provide/take in data to/from a user. Also, in these use cases, the user may run the bot's script step-by-step, stop the bot, or otherwise intervene during the execution of the script. Attended bots are suitable for routines where dynamic inputs (i.e., inputs gathered during a routine) are required, where some decisions or checks need to be made that require human judgment, or when the routine is likely to have unforeseen exceptions and it is important to detect such exceptions. Entering data from an invoice in a spreadsheet format into a financial system is an example of a routine suitable for attended RPA, given that in this setting some types of errors may have financial consequences.

Unattended RPA bots, on the other hand, execute scripts without human involvement and do not take inputs during their execution. Unattended RPA bots are suitable for executing deterministic routines where all execution paths (including exceptions) are well understood and can be codified. Copying records from one system into another via their user interfaces through a series of copy-paste operations is an example of a routine that could be executed by an unattended bot.

In light of the above, we can classify RPA as a specific type of process automation technology – a broader class of software tools that include Business Process Management Systems (BPMS), document workflow systems, and other types of workflow automation tools (Dumas et al. 2018). A key difference between RPA on the one hand and BPMS and workflow systems on the other is that RPA is meant to automate deterministic routines that involve automated steps where either an interaction is performed with the UI of an application or an API is called (in both cases the steps are automated). In contrast, BPMS and workflow systems are designed to automate processes that involve combinations of automated tasks and manual tasks. Related to this distinction, BPMS and workflow systems are designed to automate end-to-end processes consisting of multiple tasks, performed by multiple types of participants (e.g., roles, groups). Meanwhile, RPA tools are developed to automate smaller routines, which correspond to individual tasks in a process, or even steps within a task, such as creating an invoice or a student record in an information system. As such, RPA tools and BPMSs are complementary. A BPMS may trigger an RPA tool to perform a given step in a process.

RPA tools are able to automate a wide range of routines, thus raising the following question: Which routines in an organization may be beneficially automated using RPA? We envision a class of tools, namely RPM tools[3], that answer this question. Specifically, we define RPM as *a class of techniques and tools to analyze data collected during the execution of user-driven tasks in order to support the identification and assessment of candidate routines for automation and the discovery of routine specifications that can be executed by RPA bots. In this context, a* user-driven task *is a task that involves interactions between a user (e.g., a worker in a business process) and one or more software applications. Accordingly, the main source of data for an RPM tool is a UI log.*

In line with the above definition, we distinguish three main phases in RPM: (1) collecting and pre-processing UI logs corresponding to executions of one or more tasks; (2) identifying candidate routines for RPA; and (3) discovering executable RPA routines.[4] In the following, we analyze the concepts involved across these three phases and refine these phases into a tool pipeline.

2.2 Concepts

The main input for RPM is a *UI log*, which has to be recorded beforehand. A UI log is a timestamped sequence of events performed by a single user in a single workstation, involving events generated by one or more applications (including Web and desktop applications). An example of a UI log, which we use herein as a running example, is given in Table 1.

Each row in this example corresponds to one event (e.g., accessing url "https://www.unimelb.edu.au", clicking button "New record", etc.). Each event is characterized by an event type (e.g., click button, edit text field), a timestamp and other information (e.g., the label of a button,

[3] Some commercial and open-source tool developers use the term *task mining* to refer to RPM, e.g., in the PM4Py toolset http://pm4py.pads.rwth-aachen.de/task-mining/.

[4] Once an RPA routine has been automated via an RPA bot, a fourth phase is to monitor this bot in order to detect anomalies or performance degradation events that may signal that the bot may need to be adjusted, re-implemented, or retired. While relevant from a practical perspective, this phase is orthogonal to the three previous phases since it is relevant both for bots developed manually and bots developed using RPM techniques. Furthermore, previous work has shown that existing process mining tools are suitable for analyzing logs produced by RPA bots for monitoring purposes (Geyer-Klingeberg et al. 2018).

Table 1 Example of UI log

	Timestamp	Event type	Source	Arg 1	Arg 2	Arg 3
1	2019-03-03T19:02:18	Open file	File System	FileName: student_data.xls		
2	2019-03-03T19:02:23	Go to URL	Web	URL: "https://www.unimelb.edu.au"		
3	2019-03-03T19:02:26	Click button	Web	Label: "New record"		
4	2019-03-03T19:02:28	Go to cell	Worksheet	SheetName: Sheet1	Address: A2	Value: "John"
5	2019-03-03T19:02:31	Click text field	Web	Label: "First Name"	Value: ""	
6	2019-03-03T19:02:37	Edit text field	Web	Label: "First Name"	Value: "John"	
7	2019-03-03T19:02:40	Go to URL	Web	URL: "https://www.distraction.com"		
8	2019-03-03T19:07:33	Open email	Email Client	From: "student@abc.com"	Message: "Dear Course Coordinator,..."	
9	2019-03-03T19:07:40	Click button	Email Client	Label: "Reply"		
10	2019-03-03T19:07:48	Edit text field	Email Client	Label: "Message"	Value: "Dear Student, your request has been processed"	
11	2019-03-03T19:07:50	Click button	Email Client	Label: "Send"		
12	2019-03-03T19:07:55	Go to URL	Web	URL: "https://www.unimelb.edu.au"		
13	2019-03-03T19:08:02	Click text field	Web	Label: "Last Name"	Value: ""	
14	2019-03-03T19:08:05	Edit text field	Web	Label: "Last Name"	Value: "Do3"	
15	2019-03-03T19:08:08	Click text field	Web	Label: "Last Name"	Value: "Do3"	
16	2019-03-03T19:08:12	Edit text field	Web	Label: "Last Name"	Value: "Doe"	
17	2019-03-03T19:08:17	Click text field	Web	Label: "Country of residence"	Value: ""	
18	2019-03-03T19:08:21	Edit text field	Web	Label: "Country of residence"	Value: "Australia"	
19	2019-03-03T19:08:28	Click button	Web	Label: "Save"		
20	2019-03-03T19:08:35	Click button	Web	Label: "New record"		
21	2019-03-03T19:08:38	Go to cell	Worksheet	SheetName: Sheet1	Address: A3	Value: "Albert"
22	2019-03-03T19:08:39	Copy	Worksheet	Content: "Albert"		
23	2019-03-03T19:08:40	Copy	Worksheet	Content: "Albert"		
24	2019-03-03T19:08:42	Click text field	Web	Label: "First Name"	Value: ""	
25	2019-03-03T19:08:43	Paste	Web	Value: "Albert"		
26	2019-03-03T19:08:44	Edit text field	Web	Label: "First Name"	Value: "Albert"	
27	2019-03-03T19:08:47	Go to cell	Worksheet	SheetName: Sheet1	Address: B3	Value: "Rauf"

Table 1 continued

	Timestamp	Event type	Source	Arg 1	Arg 2	Arg 3
28	2019-03-03T19:08:49	Copy	Worksheet	Content: "Rauf"		
29	2019-03-03T19:08:52	Click text field	Web	Label: "Last Name"	Value: ""	
30	2019-03-03T19:08:53	Paste	Web	Value: "Rauf"		
31	2019-03-03T19:08:54	Edit text field	Web	Label: "Last Name"	Value: "Rauf"	
32	2019-03-03T19:08:58	Go to cell	Worksheet	SheetName: Sheet1	Address: C3	Value: "Germany"
33	2019-03-03T19:09:01	Copy	Workseet	Content: "Germany"		
34	2019-03-03T19:09:03	Click on text field	Web	Label: "Country of residence"	Value: ""	
35	2019-03-03T19:09:04	Paste	Web	Value: "Germany"		
36	2019-03-03T19:09:05	Edit text field	Web	Label: "Country of residence"	Value: "Germany"	
37	2019-03-03T19:09:09	Tick box	Web	Label: "International student"		
38	2019-03-03T19:09:14	Click button	Web	Label: "Save"		
...	...	...	...	...	...	...

the value of a cell, etc.), called *payload*, sufficient to reconstruct the performed activity. For example, for an event that refers to clicking a button, it is important to store a unique identifier of this button (e.g., either the element identifier, or its name if this is unique in the page). Likewise, for an event that refers to editing a field, an identifier of the field as well as a new value assigned to that field are required attributes. Events of the same type usually are characterized by the same attributes in the payload. Generally, events recorded by different source applications contain different attributes in the payload. For example, the events generated by a spreadsheet (e.g., an Excel spreadsheet) contain information such as spreadsheet name and position of the involved cell or range of cells, while Web-based events are characterized by the corresponding Web page, name and/or identifier of the involved HTML element. Events in UI logs are chronologically ordered based on their timestamps. Some events may be aggregated into *actions* of higher level. For example, two events *Go to cell* and *Copy cell content* can be merged into one action called *Copy cell*.

In order to obtain a UI log, all user interactions related to a particular task have to be recorded. This recording procedure can be long-running, covering a session of several hours of work, if the user performs multiple instances of the task one after the other. During such a session, a worker is expected to perform a number of actions of the same or different types. The UI log used as running example describes the execution of a task corresponding to transferring student data from a spreadsheet into the Web form of a study information system. The Web form requires information such as the student's first name, last name and country of residence. If the country of residence is not Australia, the user needs to perform one more step, indicating that the student has to be registered as an international student.

Each execution of a task is represented by a *task trace*. In our running example, there are two traces belonging to the new record creation task. From the log, we can see that the user performed the creation of a new record in two different ways. In the first case, they filled in the form manually, while in the second case, they copied the data from a worksheet and pasted it into the corresponding fields.

Given a collection of task traces, the goal of RPM is to identify a repetitive sequence of actions that can be observed in multiple task traces, herein called a *routine*, and identify routines amenable for automation. For each such routine, RPM then aims at extracting an executable specification (herein called a *routine specification*).

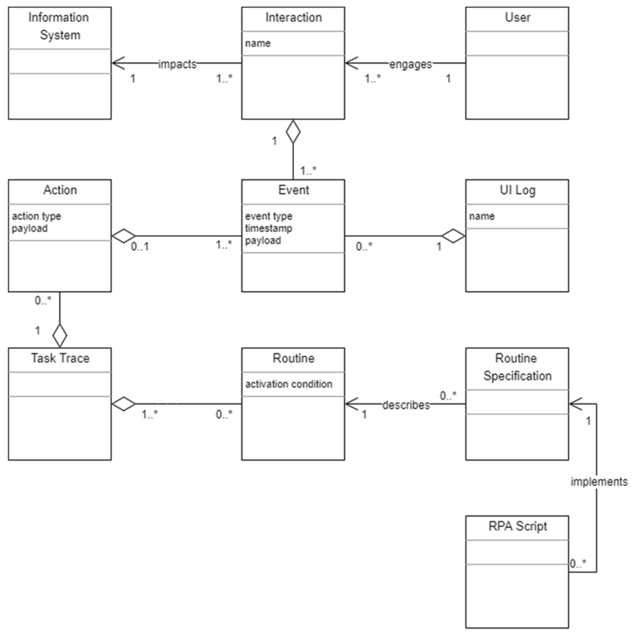


Fig. 2 Class diagram of RPM concepts

This routine specification may initially be captured in a platform-independent manner, and then compiled into a platform-dependent *RPA script* to be executed in a specific RPA tool.

To summarize, Fig. 2 presents a class diagram capturing the above concepts and their relations.

2.3 RPM Pipeline

As mentioned earlier, the three main phases of RPM are: (1) UI log collection and pre-processing; (2) candidate routine identification; and (3) executable routine discovery. In order to provide a more detailed view of the steps required to achieve the goals of RPM, we decompose the first phase into the recording step itself, and three pre-processing steps, namely removal of irrelevant events (noise filtering), segmentation of the log into routine traces, and simplification of the resulting routine traces. We then map the second phase into a single step and we decompose the third phase into two steps: the discovery of platform-independent routine specifications and compilation of the latter into platform-specific specifications (scripts). This decomposition of the three phases into steps is summarized in the RPM pipeline depicted in Fig. 3. In the following, we discuss each of the steps in this pipeline.

The *recording* of a UI log involves capturing low-level UI events, such selecting a field in a form, editing a field, opening a desktop application, or opening a Web page. UI log recording may be achieved by instrumenting the software applications (including the Web browser) used by the workers, via plugin or extension mechanisms. Logs

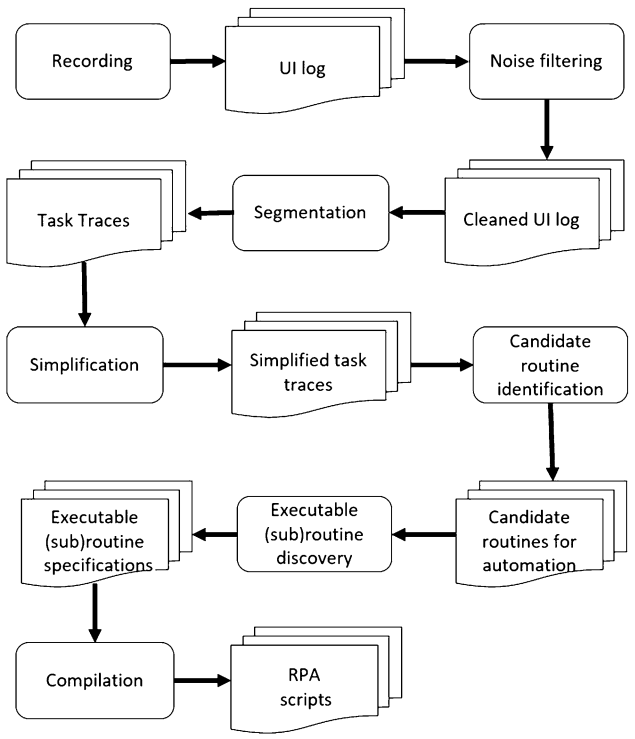

Recording
UI log
Noise filtering
Task Traces
Segmentation
Cleaned UI log
Simplification
Simplified task traces
Candidate routine identification
Executable (sub)routine specifications
Executable (sub)routine discovery
Candidate routines for automation
Compilation
RPA scripts

Fig. 3 RPM pipeline

collected by such plugins or extensions may be merged in order to produce a raw UI log corresponding to the execution of one or more tasks by a user during a period of time. This raw log usually needs to undergo some pre-processing in order to be suitable for RPM.

As shown in Fig. 2, a UI log may contain events that do not belong to an execution of any action, herein called *noise*. Noise may occur for example when the user is interrupted or gets distracted during the execution of a task, leading to performing activities that are not relevant to the task in question (e.g., pausing the transfer of student records to reply to an email). Accordingly, the first step in the pipeline (after the recording step) is dedicated to identifying and filtering out events that do not belong to any action (noise filtering) and as such should not be automated. In our running example, event 7 (visiting https://www.distraction.com) as well as events 8-11 (replying to an email) are examples of noise.

Given a noise-filtered UI log, the next problem is to identify the boundaries of the task traces. We call this problem *segmentation.* Specifically, the purpose of segmentation is to identify sequences of consecutive actions that represent the execution of a task. The input of segmentation is a UI log containing a single sequence of events, while the output is a set of traces each representing the executions of a task. We observe that noise filtering and segmentation are intertwined. By identifying the boundaries of task traces, we also understand which events are not part of any task, hence representing noise. Segmentation can be performed in several ways. For example, it can be performed by using domain knowledge or a UI log can be combined with transactional data recorded by an enterprise system to identify start and end events of a task (Linn et al. 2018).

Task traces may contain events that have no effect on the final outcome. Such events constitute *waste*. For example, a task trace may contain redundant events (e.g., pressing Ctrl-C twice consecutively on the same field, which has the same effect as doing it only once). Another type of waste has to do with defects, e.g., typing in a text field, then deleting the content of the field and typing something different. In our running example, events 13, 14 and 22 represent overprocessing waste. Accordingly, the pipeline includes a *simplification* step, that aims at waste identification and removal. The simplification step includes aggregation of events into higher-level actions. In this way, the task traces will be much more compact and concise, and thus easier to translate into a target language.

Given a set of simplified task traces, the next step is to identify *candidate routines* for automation. This step aims at extracting repetitive sequences of actions that occur across multiple task traces, a.k.a. routines, and at identifying the ones that are amenable for automation. The output of this step is a set of automatable or semi-automatable routines, ranked accordingly to their automation potential (e.g., based on their execution frequency and length).

After the candidate routines for automation are identified, the next step is *executable (sub)routine discovery*. For each candidate routine, this step identifies the *activation condition* (events 3 and 20 in Table 1), which indicates when an instance of the routine should be triggered, and the *routine specification*, which specifies what actions should be performed within that routine.

The *executable (sub)routine discovery* step leads to a platform-independent representation of the routine, which can then be compiled into a script targeted at a specific RPA tool via a final *compilation* step. This step generates an executable script by mapping actions from the routine specification into commands in the scripting language of the target RPA tool.

The generated bot can then be executed in *attended* or *unattended* settings. In attended settings, given an activation condition extracted from the routine specification, it can notify the user about its "readiness" to perform the routine when the condition is met. It can be paused during execution, so the user can make small corrections if needed and then resume the work. In unattended settings, the bot works independently without human involvement.

Let us demonstrate this RPM pipeline on the running example (Table 1):

Noise filtering. Events e7, e8, e9, e10, and e11 are noise and must be filtered out from the log.

Segmentation. The main goal of the task captured in the running example is to create a new record of a student. Thus, the end event of a task trace is the actual creation of such record, achieved as a result of clicking the button "Save". Thus, there are two task traces:

- Trace 1: e1, e2, e3, e4, e5, e6, e12, e13, e14, e15, e16, e17, e18, e19;
- Trace 2: e20, e21, e22, e23, e24, e25, e26, e27, e28, e29, e30, e31, e32, e33, e34, e35, e36, e37, e38;

Simplification. Events e13 and e14 in Trace 1 as well as event e22 in Trace 2 are waste and must be removed. There are three possible events merges:

- Events {e5, e6}, {e15, e16} and {e17, e18} can be merged into action *Write into text field* with payload p = {*Label*, *Value*}.
- Events {e24, e25, e26}, {e29, e30, e31} and {e34, e35, e36} can be merged into action *Paste into text field* with payload p = {*Label*, *Value*}.
- Events {e21, e23}, {e27, e28} and {e32, e33} can be merged into action *Copy cell* with payload p = {*SheetName*, *Address*, *Content*}.

Candidate routine identification. The actions related to the modification of the Web-form fields occur in both traces. Thus, the corresponding sequence of actions constitutes a routine. Note that Trace 1 contains some actions that cannot be automated (the user fills in the form manually), while Trace 2 consists of automatable actions only.

Executable (sub)routine discovery. The activation condition for the extracted routine is *Click button "New Record"* (e3 and e20 of the running example). Figure 4 presents the New Record Creation routine specification.

Compilation. The routine specification is then compiled into an RPA script. Here, each step from the specification model is "translated" into a specific command in the language of the target RPA tool. Figure 5 provides an example of script generated from the discovered routine specification.

3 Challenges and Guidelines

Each step of the RPM pipeline presented in Fig. 3 gives rise to research challenges. Next, we give an overview of some of these challenges and propose approaches to tackle them.

Recording. The main challenge in this step is to identify what actions must be recorded. The same action (e.g., mouse click) can either be important or irrelevant in a

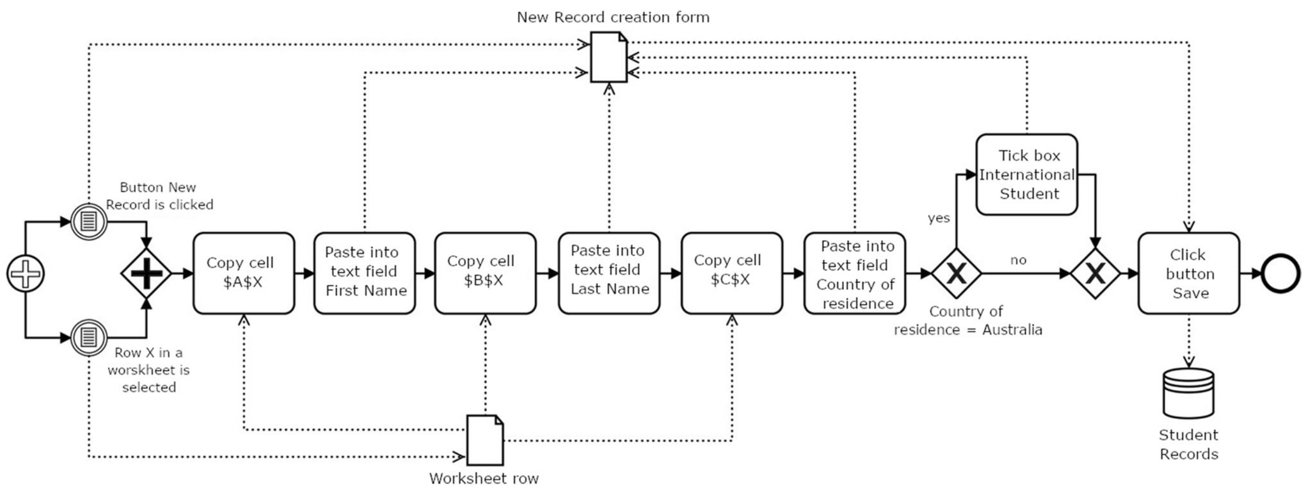

New Record creation form
Button New Record is clicked
Row X in a worksheet is selected
Copy cell AX
Paste into text field First Name
Copy cell BX
Paste into text field Last Name
Copy cell CX
Paste into text field Country of residence
yes
Tick box International Student
no
Country of residence = Australia
Click button Save
Student Records
Worksheet row

Fig. 4 New Record Creation routine specification

```
Assign value of selected row to variable "$X$"
Excel: Get value of cell "A$X$" and assign to
        variable "$Clipboard$"
Assign value of Clipboard to variable "$FirstName.value$"
        in "http://www.unimelb.edu.au"
Excel: Get value of cell "B$X$" and assign to
        variable "$Clipboard$"
Assign value of Clipboard to variable "$LastName.value$"
        in "http://www.unimelb.edu.au"
Excel: Get value of cell "C$X$" and assign to
        variable "$Clipboard$"
Assign value of Clipboard to variable "$CountryOfResidence.value$"
        in "http://www.unimelb.edu.au"
If $CountryOfResidence.value$ Not Equal To (<>) "Australia" Then
        Mouse Click: Left Button on "International Student"
                in "http://www.unimelb.edu.au"
End If
Mouse Click: Left Button on "Save"
        in "http://www.unimelb.edu.au"
```

Fig. 5 New Record Creation script

given context. For example, a mouse click on a button is an important event but a mouse click on the background of a Web page is an irrelevant event. Also, when a worker selects a Web form, we need to record events at the level of the Web page (the Document Object Model – DOM) in order to learn routines at the level of logical input elements (e.g., fields) and not at the level of pixel coordinates, which are dependent on screen resolution, window sizes, etc. Existing UI event recording tools, such as JitBit Macro Recorder,[5] TinyTask,[6] and WinParrot,[7] save all the actions performed by the user at a too low level of granularity, with reference to pixel coordinates (e.g., mouse click at coordinates 748,365). As a result, the UI interaction logs generated by these tools are not suitable for extracting useful routines. RPA tools (e.g., UiPath Enterprise RPA Platform, and Automation Anywhere Enterprise RPA) provide recording functionality. However, this functionality is intended to record RPA scripts. These tools do not capture details about the values of the different fields, as these values are not relevant for RPA script generation. Hence, a new family of recording tools is needed to record UI logs required for RPM.

In recent work, Leno et al. (2019) introduced a tool to record UI logs in a format that is suitable for RPM. The tool records not only the UI actions (selecting a field, editing a field, copying into or pasting from the clipboard), but also the values associated with these actions (e.g., the value of a field after an editing event). The tool supports MS Excel and Google Chrome. The tool also simplifies the recorded UI logs by removing redundant events (e.g., double-copying without pasting, navigation between cells in Excel without modifying or copying their content). The applicability of such tools, however, is limited to desktop applications that provide APIs for listening to UI events and accessing the data consumed and produced by these events. To develop a more general solution, it may be necessary to combine this latter approach with the OCR technology in order to detect UI events and associated data from application screenshots, as outlined in Ramirez et al. (2019); Linn et al. (2018).

Noise filtering. One of the main challenges of this stage is to separate noise from events that contribute to tasks. A possible solution is to treat noise as chaotic events that can happen anywhere during the process execution. A technique for filtering out such chaotic events is described in Tax et al. (2019). However, if noise gravitates towards one particular state or set of states in the task (e.g., towards the start or the end of the task), techniques such as the one mentioned above may not discover it and consequently may not filter it out. Moreover, some events can be mistakenly removed due to the different ways the same task can be performed and induce what may mistakenly appear to be chaotic sequences of events. This can be avoided by

[5] https://www.jitbit.com/macro-recorder/.

[6] https://www.tinytask.net/.

[7] http://www.winparrot.com/.

considering the data perspective of processes, i.e., values of data objects that are manipulated by actions and events. Looking at the data objects, it is possible to identify events and actions that share the same attribute values (e.g., copying a value from a worksheet and then pasting it into a Web form), or have the same source/origin (e.g., all the actions performed on the same web site). The events that do not share any data attributes and/or values or originate from uncommon sources most likely constitute noise.

Segmentation. A UI log, in its raw form, consists of one single sequence of events recorded during a session. During this session, a user may have performed several executions of one or multiple tasks. In other words, a UI log may contain information about several tasks, whose actions and events are mixed in some order that reflects the particular order of their execution by the user. Moreover, the same task can be "spread" across multiple logs, for example if a task is performed by several users working on different work stations. Before identifying candidate routines for automation, we therefore need to segment a UI log into traces, such that each trace corresponds to one execution of a task.

In some scenarios, segmentation may be accomplished by combining transactional data recorded by enterprise information systems together with UI logs, as proposed in Linn et al. (2018). For instance, after pressing button "Save" in our running example, event *Create record* can be generated, which marks the end point of the current task trace. The problem of this approach, however, is that such transactional data may only provide limited information about the task.

The problem of segmentation in RPM is akin to that of Web session mining – widely studied in the field of Web log mining (Liu 2007) – where the input is a set of click-streams and the goal is to extract sessions where a user engages with a web application to fulfill a goal. Most of traditional approaches to session identification can also be used for RPM. However, they can only be used in the context of Web interactions, as they are based on Web organization specifics. For example, one of the key concepts they use is that a page must have been reached from a previous page in the same session. Therefore, one of the challenges in RPM segmentation is that tasks are usually performed across different systems and applications, and the Web browser is just one of these applications. An alternative approach is to use time-based heuristics to set a limit for the total duration of a session or the maximal allowed time difference between two events. However, this approach is unreliable since users may be involved in different activities when performing a task. In addition, tasks are usually performed in batches, and that increases the difficulty of using time-based heuristics for the correct identification of the tasks' boundaries. As an example, let us take the task of filling in Web forms by copying data from a spreadsheet. For each row in the spreadsheet, the user creates a new form, copies the required data from a cell of that row and pastes it into the corresponding text field, then presses the submit button and starts the task again. In this example, the time difference between two different tasks can be smaller than the time difference between events in the same task, leading to an incorrect segmentation.

The problem of UI log segmentation is also related to that of correlating uncorrelated events in event logs used for process mining (Bayomie et al. 2019, 2016; Ferreira and Gillblad 2009). However, this problem has been addressed in restrictive settings. In particular, Ferreira and Gillblad (2009) addressed the problem when the process (in our case the routine) does not have cycles/repetitions, whereas (Bayomie et al. 2016, 2019) assume that a process model is given as input, which means that the the routine specification is known. Also, the approaches in Ferreira and Gillblad (2009) and Bayomie et al. (2016) were shown to produce rather inaccurate results, whereas RPM seeks to identify routines with high levels of confidence, given that replicating a routine inaccurately can lead to costly errors, especially in contexts where unattended bots are used.

Simplification. Even if an event belongs to a task, it may still be redundant. For example, when a user fills in a text field with a mistake, and then has to fill it in again. In this case, the events that belong to the first time of filling in the text field are redundant. Depending on the context, the same event may be integral part of a routine or it may be redundant. Thus, classical frequency-based filtering approaches, like (Conforti et al. 2017), cannot be applied to address this problem. One of the possible solutions is to use sequential pattern mining techniques to distinguish between events that are part of mainstream behavior and outlier events (Sani et al. 2017). However, in case some events are rarely seen during a task execution they can be mistakenly treated as outliers. The outlined problem creates a need for semantic filtering. Groups of events can be combined into actions of a higher semantic meaning. The challenge here is to identify the semantic boundaries of an action and the attributes to form its payload.

Candidate routines identification. This step can be decomposed into two substeps: 1) Routine extraction; 2) Identification of automatable routines. Each of the presented substeps faces its own challenges.

The first substep aims at the identification and extraction of repetitive sequential patterns that represent the execution of routines. One of the challenges here is that, during the execution of a routine, the user can perform other actions that are not part of the routine. When identifying the routines, such actions have to be ignored. In this regard, sequential pattern mining techniques, in particular the ones that work with gapped patterns (Liao and Chen 2013) can be used. Another challenge is that sometimes the actions that constitute a routine can be performed in random order (e.g., when filling in a Web form). Thus, it is difficult to identify frequently-occurring patterns. One possible solution is to use abstraction mechanisms as shown in Bialy et al. (2019). An alternative approach is to use more flexible notions of patterns like *alphabet repeats* (Bose and van der Aalst 2009) that do not take into consideration the order in which the events occur, or even declarative specifications as described in Leno et al. (2020).

The main goal of the second substep is to identify routines amenable for automation. A discovered routine is considered to be a candidate for automation if this routine is either semi- or fully automatable. In this context, the challenge is how to identify whether the routine is automatable or not. In Geyer-Klingeberg et al. (2018), the authors describe how to assess the automation potential of a task. The frequency of execution of a task is presented as the main criterion for automation. However, if the task is frequent there is no guarantee that it is automatable.

Lacity and Willcocks (2016) propose high-level guidelines for determining if a task is a candidate for automation in the context of a case study at Telefonica. However, this work does not provide a formal and precise definition of an automatable task, which would be crucial to automate the identification of automatable routines. In fact, a major challenge is how to formally characterize what makes a routine suitable for RPA, in a sufficiently precise way to enable the design of efficient algorithms to identify these routines from large volumes of UI logs. One possible solution is to use the notion of determinism. A routine can be automated if every event belonging to the routine is deterministically activated and uses data produced by previous actions (e.g., the manual input into a text field is an example of a non-deterministic action). The challenge here is to identify non-deterministic events in a UI log, which reflect non-deterministic actions being performed. One of the problems related to non-deterministic actions that can arise is the identification of partially-automatable routines including automatable sub-routines. If somewhere in the middle of a routine a non-deterministic action happens, this action splits the routine into two automatable sub-routines. We also observe that not every routine is worth to be automated. The automation of some routines can bring much more benefits than the automation of other routines. Thus, a cost-benefit analysis of routine automation is an important task in RPA (Lacity and Willcocks 2016).

Executable routine discovery. Given a set of routines, executable routine discovery consists in constructing a routine specification that represents the entire set of routines in the form of a control-flow model enhanced with data flow. The challenge here is that there may be multiple (alternative) ways of performing the same routine, e.g., different workers may perform the same routine differently. Hence, when discovering a routine specification, we need to focus on capturing all the preconditions under which the routine should be triggered and the effects (postconditions) of the routine. This calls for dedicated quality measures for routine specifications, which capture the extent to which the preconditions and the effects of the observed routines are covered by a given routine specification. Also, in case two different routines produce the same effects, it is important to identify the optimal one. Searching for the best alternative variant of a routine is a challenge in executable routine discovery.

Some repetitive routines may be triggered only under certain conditions. For example, when a purchase order is of type “retail-EU”, then a certain sequence of actions is performed in order to comply with specific EU regulations and this sequence of actions corresponds to a repetitive routine that can be automated. On the other hand, when the order is of type “retail-US” another routine is performed. Or, alternatively, we might find that handling orders of type “retail-EU” follows some specified sequence of steps (that can be captured via an executable process model), whereas for “retail-CN”, handling the order is an ad-hoc procedure and no regularity can be found. Therefore, the handling of “retail-EU” orders can be automated by means of an executable model, whereas the processing of “retail-CN” orders cannot. Recent work (Bosco et al. 2019) has put forward the idea of using rule mining techniques, such as RIPPER, to discover conditions under which a given routine can be automated. However, the applicability of these techniques on real-life RPM scenarios has yet to be tested, and is likely to raise scalability and robustness challenges.

Another challenge in this step is to discover the data transformations that occur within each action in a routine. Indeed, if we want an RPA bot to reproduce the actions of a routine, we need to encode in the bot’s script how the parameters of each action are computed from the routine’s input parameters or from the parameters of previous actions in the routine. Recent work (Bosco et al. 2019) suggests that this step in the discovery of executable routines can be implemented using existing methods for automated discovery of data transformations “by example” (Abedjan et al. 2016; Jin et al. 2017). However, these methods suffer from scalability issues. In addition, their scope (i.e., the types of transformations they can discover) is rather limited. Thus, new advances in the field of automated discovery of data transformations are needed to make data transformation applicable in the context of RPM.

Compilation. Given a routine specification, the compilation step aims at generating an executable RPA script that implements the specification. This step requires the correct identification of the application elements involved during the routine execution (e.g., a button or a text field in a Web form). For example, when converting the action of clicking a button in a Web page into an executable command, we need to identify the HTML element that represents this button and extract its DOM position. Such information can be recorded by a logger during the *recording* step. However, sometimes this information may be missing. For example, some of the Web elements (e.g., the links) do not have any identifiers that can be used to locate them in the page. In cases where Web sites are created dynamically and consist of a large amount of nested containers it is very difficult to extract the correct location of the elements. Therefore, when working with custom applications without an API, it may not be possible to identify the type of an event correctly. For this reason, an intelligent recognition of the elements is required. In this regard, technologies such as OCR may be used, but the challenge here is to preserve the semantics of the actions recorded and to capture all the data involved during their execution.

4 Related Work

The discovery of candidate routines for automation via RPA tools is so far a largely unexplored problem. Recent work (Leopold et al. 2018) sketched an approach to identify passages in textual descriptions of business processes (e.g., work instructions) that might refer to tasks amenable for automation. This approach, however, may lead to imprecise results due to the complexity of natural language analysis. Also, it requires textual documentation of suitable quality and completeness, and assumes that tasks are performed exactly as documented. In reality, workers may perform steps that are not fully documented in order to deal with exceptions and variations. Hence, a task that might appear as automatable according to its work instructions might turn out not to be automatable in practice. Another body of related work includes approaches for auto-completing Web forms with default values or predicted values (Hermens and Schlimmer 1994). These approaches help users during manual form filling, but they do not automate routines in the way RPA tools do.

In addition to the above work, the RPM vision presented in this paper is related to other sub-fields of data mining that seek to discover behavioral models from different types of logs. Below, we discuss the relations between RPM and three of such fields, namely process mining, web usage mining, and user interface log mining.

Process mining. RPM can be positioned as an extension of the field of process mining (van der Aalst 2016). Specifically, discovering RPA routines is closely related to the problem of Automated Process Discovery (APD), which has been widely studied in the field of process mining (Augusto et al. 2019). The purpose of APD techniques is to discover business process models from event logs recording the execution of tasks in enterprise systems. A significant subset of APD algorithms focus on discovering process models from the control-flow perspective. This subset of APD algorithms does not consider the data that is taken as input and produced as output by the tasks of the process, nor the data used by a process execution engine to evaluate branching conditions. Another subset of APD techniques target the problem of discovering process models with data-driven branching conditions (de Leoni et al. 2013) as well as control-flow relations that only hold under certain conditions (Mannhardt et al. 2017). These latter techniques provide a starting point for developing techniques for discovering RPA routines. Indeed, in order to discover RPA routines, we need to discover conditions within the routine like the activation conditions that trigger a routine. Other APD techniques focus on discovering simulation models (Martin et al. 2016). The latter type of models can be given as input to business process simulators, which execute them in a stochastic sense.

Notwithstanding the rich body of work in the field of process mining, we are not aware of techniques that discover executable process models ready to be deployed or compiled (without significant manual enhancement) into a business process execution engine. In particular, we are not aware of any work on automated process discovery that tries to discover data transformations (i.e., mappings between inputs and outputs) in automatically discovered process models. Yet, these data transformations are essential to discover process models that can be executed by a process execution engine or by an RPA tool.

There are similarities between UI logs and event logs used in process mining. Specifically, both types of logs consist of timestamped records, such that each record refers to the execution of an action (or a task) by a user. Also, each record may contain a payload consisting of one or more attribute-value pairs. Some commercial process mining vendors have exploited the similarities between UI logs and business process event logs in order to offer RPM-related features. For example, the Minit[8] process mining tool provides a multi-level process discovery feature to support some RPM tasks. Specifically, given an event log

[8] https://www.minit.io/.

recording the execution tasks and a UI log, Minit is able to generate a two-level process map. The first level shows the tasks recorded in the log extracted from the enterprise system. Each task can be expanded into a second-level process map showing the UI actions and their control-flow relations. In this way, the tool supports the (visual) identification of tasks that have relatively simple internal structures and could, therefore, be potentially automated. However, it cannot determine if a task contains fully deterministic (sub-)routines nor can it produce executable specifications of deterministic routines. Also, the tool assumes that there is a clear relation between the events in the UI log and those in the business process event log. In other words, it does not address the segmentation step in the RPM pipeline.

Another commercial tool, namely Kryon Process Discovery,[9] identifies candidate routines for RPA by analyzing UI logs in conjunction with screenshots taken while users perform their work on one or more applications. However, the candidate routines that Kryon identifies may or may not be automatable, depending on the actual data values that users have entered. If the data values that are entered in a particular step cannot be determined from the values of previously observed attributes, it means that the user is providing inputs either from external data sources (not observed in the UI) or from their own domain knowledge, and hence that step of the routine is not automatable. In other words, not all routines that are identified as candidates for automation by this tool can be automated.

While there are similarities between UI logs on the one hand, and event logs used for process mining on the other hand, there are four some notable differences. First, event logs capture events at a higher level of abstraction. Specifically a record in an event log typically refers to the execution of an entire task within a business process, such as *Check purchase order* or *Transfer student records*. Such tasks can be seen as a composition of lower-level actions, which may be recorded in a UI log. For example, task *Transfer student records* may involve multiple actions to copy the records associated with a student (name, surname, address, course details) from one application to another. Second, UI logs do not come with a notion of *case identifier* (or process instance identifier), whereas event logs typically do. In other words, events in a UI log are not explicitly correlated, and for this reason, they may need to be segmented as discussed in Sect. 2.3. Third, a record in an event log often does not contain all input or output data used or produced during the execution of the corresponding task. For example, a record in an event log corresponding to an execution of task *Transfer student records*, is likely not to contain all attributes of the corresponding student (e.g., their address). On the other hand, the presence of every input and output attribute in a UI log is necessary for RPM purposes. If some input or output attributes are missing in the UI log, the resulting routine specification would be incomplete, and hence the resulting RPA bot would not perform the routine correctly. A fourth difference is that event logs are typically obtained as a by-product of transactions executed in an information system, rather than being explicitly recorded for analysis purposes. The latter characteristic entails that event logs are more likely to suffer from incompleteness, including missing attributes as discussed above, but also missing events. For example, in a patient treatment process in a hospital, it may be that the actual arrival of the patient to the emergency room is not recorded when the patient arrives by themselves, but it is recorded when the patient arrives via an ambulance. In other words, the presence or absence of an event in an event log depends on whether or not the information system is designed to record it, and whether or not the workers actually record it. On the other hand, a UI log is recorded specifically for analysis purposes, which allows all relevant events to be collected subject to the capabilities of the UI recording tool.

Web usage mining. Web usage mining seeks to discover and analyze sequential patterns in Web data, such as click streams capturing user interactions with Web applications (Srivastava et al. 2000). Analyzing such data can help to optimize the functionality of Web-based applications, provide personalized content to users, and find the most effective logical structure for Web pages (Liu 2007). Web usage mining works with data at a similar level of granularity as RPM. Also, the data manipulated in Web log mining is often uncorrelated, meaning that it represents a sequence of actions performed throughout several sessions without explicit assignment of actions to a specific session. Given these similarities, Web usage mining techniques could provide a starting point to realize an RPM pipeline. For example, Web mining techniques for extracting sessions from Web logs could be adapted to address the problem of segmentation discussed above. On the other hand, Web usage mining techniques do not address the problem of discovering candidate routines for automation. Also, RPM differs from Web usage mining in that it is not restricted to Web applications.

User interface log mining. RPM is also related to the topic of user interface log mining. In the context of desktop assistants, research proposals such as TaskTracer and TaskPredictor have tackled the problem of analyzing user interface logs generated by desktop applications in order to identify the current task performed by a user and to detect switches between one task and another (Shen et al. 2007; Dragunov et al. 2005). Other related work in this area has

[9] https://www.kryonsystems.com/process-discovery/.

tackled the problem of task identification and classification from Desktop app user interface logs (Oliver et al. 2006; Rath et al. 2010) as well as the problem of extracting frequent sequences of actions from noisy user interface logs (Dev and Liu 2017) (which could constitute candidate routines for automation). With respect to the previously cited research, the novelty of RPM is that it seeks to discover executable routine specifications by analyzing logs that include inputs and outputs of actions (e.g., data copied to or pasted from the clipboard, data entered into cells), as opposed to purely considering sequences of actions without the associated data.

5 Conclusion

We have exposed a vision for a new class of process mining tools, namely RPM tools, capable of analyzing UI logs of fine-grained user interactions with IT systems in order to identify routines that can be automated using RPA tools. As a first step to concretize this vision, we decomposed it into a pipeline and sketched challenges that need to be overcome to implement each of the pipeline's components. We also provided some guidelines to tackle these challenges.

The proposed RPM pipeline focuses on the discovery of routines that can be executed in an end-to-end manner by an RPA bot. This assumption is constraining. In reality, routines may be automated for a certain subset of cases, but not for all cases (i.e., automation may only be partially achievable). A key challenge beyond the proposed RPM pipeline is how to discover partially deterministic routines. While a fully deterministic routine can be executed end-to-end in all cases, a partially deterministic routine can be stopped if the bot reaches a point where the routine cannot be deterministically continued given the input data and other data that the bot collects during the routine's execution. For example, while copying records of purchase orders from a spreadsheet or an enterprise system, the bot detects that this order comes from China, so it stops because it does not know how to handle such orders, or it does not find a PO number (empty cell), and hence it cannot proceed. Discovering conditions under which a routine cannot be deterministically continued (or started) is a major challenge for RPM.

The vision of RPM exposed in this paper focuses on discovering automatable routines, which is only one operation of a broader set of RPM operations that we foresee, namely *robotic process discovery*. Besides robotic process discovery, we envision that the field of RPM will encompass complementary problems and questions such as performance mining of RPA bots, e.g., "What is the success or defect rate of a bot when performing a given routine?", "What patterns are correlated with or are causal factors of bot failures?", as well as anomaly detection problems, e.g., "Are there cases where the behavior of the bot or the effects of the bot's actions are abnormal and hence warrant manual inspection and rectification?".

Acknowledgements This research was funded by the Australian Research Council (Grant DP180102839) and the European Research Council (PIX Project).

References

Abedjan Z, Morcos J, Ilyas IF, Ouzzani M, Papotti P, Stonebraker M (2016) Dataxformer: a robust transformation discovery system. In: 32nd IEEE international conference on data engineering, ICDE 2016, Helsinki, Finland, May 16–20, 2016. IEEE Computer Society, pp 1134–1145

Agaton B, Swedberg G (2018) Evaluating and developing methods to assess business process suitability for robotic process automation: a design research approach. Master's thesis, Chalmers University of Technology

Aguirre S, Rodriguez A (2017) Automation of a business process using robotic process automation (RPA): a case study. In: Proceedings of the 4th workshop on engineering applications (WEA). Springer, New York, pp 65–71

Augusto A, Conforti R, Dumas M, La Rosa M, Maggi FM, Marrella A, Mecella M, Soo A (2019) Automated discovery of process models from event logs: review and benchmark. IEEE Trans Knowl Data Eng 31(4):686–705

Bayomie D, Awad A, Ezat E (2016) Correlating unlabeled events from cyclic business processes execution. In: Proceedings of the 28th international conference on advanced information systems engineering (CAiSE). Springer, New York, pp 274–289

Bayomie D, Di Ciccio C, La Rosa M, Mendling J (2019) A probabilistic approach to event-case correlation for process mining. In: Proceedings of the international conference on conceptual modeling (ER2019). Springer, New York

Bialy Y, Yehezkel A, Roseberg E, Smutko A (2019) Process mining from low-level user actions. In: Proceedings of the BPM 2019 workshops. Springer, New York

Bosco A, Augusto A, Dumas M, La Rosa M, Fortino G (2019) Discovering automatable routines from user interaction logs. In: Proceedings of the business process management forum (BPM Forum). Springer, New York

Bose RPJC, van der Aalst WMP (2009) Abstractions in process mining: a taxonomy of patterns. In: Dayal U, Eder J, Koehler J, Reijers HA (eds) Business process management. BPM 2009, Springer, Heidelberg

Conforti R, La Rosa M, ter Hofstede AHM (2017) Filtering out infrequent behavior from business process event logs. IEEE Trans Knowl Data Eng 29(2):300–314

de Leoni M, Dumas M, García-Bañuelos L (2013) Discovering branching conditions from business process execution logs. In: Proceedings of the 16th international conference on fundamental approaches to software engineering – (FASE), pp 114–129

Dev H, Liu Z (2017) Identifying frequent user tasks from application logs. In: Proceedings of IUI 2017. Springer, New York, pp 263–273

Dong G, Pei J (2007) Sequence data mining, volume 33 of Advances in Database Systems. Kluwer

Dragunov AN, Dietterich TG, Johnsrude K, McLaughlin MR, Li L, Herlocker JL (2005) Tasktracer: a desktop environment to support multi-tasking knowledge workers. In: St. Amant R, Riedl J, Jameson A (eds) Proceedings of the 10th international conference on intelligent user interfaces, IUI 2005, San Diego, California, USA, January 10–13, 2005. ACM, pp 75–82

Dumas M, La Rosa M, Mendling J, Reijers HA (2018) Fundamentals of business process management, 2nd edn. Springer, New York

Ferreira DR, Gillblad D (2009) Discovering process models from unlabelled event logs. In: Proceedings of the 7th international conference on business process management (BPM). Springer, New York, pp 143–158

Geyer-Klingeberg J, Nakladal J, Baldauf F, Veit F (2018) Process mining and robotic process automation: a perfect match. In: Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018. CEUR-WS.org, pp 124–131

Hermens LA, Schlimmer JC (1994) A machine-learning apprentice for the completion of repetitive forms. IEEE Expert 9(1):28–33

Jin Z, Anderson MR, Cafarella MJ, Jagadish HV (2017) Foofah: transforming data by example. In: Salihoglu S, Zhou W, Chirkova R, Yang J, Suciu D (eds) Proceedings of the 2017 ACM international conference on management of data, SIGMOD conference 2017, Chicago, IL, USA, May 14–19, 2017. ACM, pp 683–698

Lacity Mary, Willcocks Leslie P (2016) Robotic process automation at telefónica O2. MIS Q Exec 15(1)

Leno V, Polyvyanyy A, La Rosa M, Dumas M, Maggi FM (2019) Action logger: enabling process mining for robotic process automation. In: Proceedings of the business process management demonstration track. CEUR

Leno V, Dumas M, Maggi FM, La Rosa M, Polyvyanyy A (2020) Automated discovery of declarative process models with correlated data conditions. Inf Syst 89:101482

Leopold H, van der Aa H, Reijers HA (2018) Identifying candidate tasks for robotic process automation in textual process descriptions. In: Proceedings of BPMDS and EMMSAD. Springer, New York, pp 67–81

Liao VC-C, Chen M-S (2013) Efficient mining gapped sequential patterns for motifs in biological sequences. BMC Syst Biol 7(S–4):S7

Linn C, Zimmermann P, Werth D (2018) Desktop activity mining – a new level of detail in mining business processes. In: Workshops der INFORMATIK 2018 – Architekturen, Prozesse, Sicherheit und Nachhaltigkeit, pp 245–258

Liu B (2007) Web usage mining. In: Web data mining: exploring hyperlinks. contents, and usage data. Springer, Berlin, pp 449–483

Mannhardt F, de Leoni M, Reijers HA, van der Aalst WMP (2017) Data-driven process discovery – revealing conditional infrequent behavior from event logs. In: Proceedings of the 29th international conference on advanced information systems engineering. Springer, New York, pp 545–560

Martin N, Depaire B, Caris A (2016) The use of process mining in business process simulation model construction – structuring the field. Bus Inf Syst Eng 58(1):73–87

Oliver N, Smith G, Thakkar C, Surendran AC (2006) SWISH: semantic analysis of window titles and switching history. In: Paris C, Sidner CL (eds) Proceedings of the 11th international conference on intelligent user interfaces, IUI 2006, Sydney, Australia, January 29–February 1, 2006. ACM, pp 194–201

Ramirez AJ, Reijers HA, Barba I, Del Valle C (2019) A method to improve the early stages of the robotic process automation lifecycle. In: Proceedings of the 31st international conference on advanced information systems engineering (CAiSE). Springer, New York, pp 446–461

Rath AS, Devaurs D, Lindstaedt SN (2010) Studying the factors influencing automatic user task detection on the computer desktop. In: Wolpers M, Kirschner PA, Scheffel M, Lindstaedt SN, Dimitrova V (eds) Sustaining TEL: from innovation to learning and practice – 5th European conference on technology enhanced learning, EC-TEL 2010, Barcelona, Spain, September 28–October 1, 2010. Springer, New York, pp 292–307

Sani MF, van Zelst SJ, van der Aalst WMP (2017) Improving process discovery results by filtering outliers using conditional behavioural probabilities. In: Proceedings of the business process management workshops. Springer, New York, pp 216–229

Shen J, Li L, Dietterich TG (2007) Real-time detection of task switches of desktop users. In: Veloso MM (ed) IJCAI 2007, proceedings of the 20th international joint conference on artificial intelligence, Hyderabad, India, January 6–12, 2007, pp 2868–2873

Srivastava J, Cooley R, Deshpande M, Tan P-N (2000) Web usage mining: discovery and applications of usage patterns from web data. SIGKDD Explor Newsl 1(2):12–23

Tax N, Sidorova N, van der Aalst WMP (2019) Discovering more precise process models from event logs by filtering out chaotic activities. J Intell Inf Syst 52(1):107–139

Tornbohm C (2017) Gartner market guide for robotic process automation software. Report G00319864, Gartner

van der Aalst WMP (2016) Process mining – data science in action, 2nd edn. Springer, New York

van der Aalst WMP, Bichler M, Heinzl A (2018) Robotic process automation. Bus Inf Syst Eng 60(4):269–272