Robotic Process Mining

Marlon Dumas[1,3], Marcello La Rosa[2,3], Volodymyr Leno[1,2,3], Artem Polyvyanyy[2], and Fabrizio Maria Maggi[4]

[1] University of Tartu, Tartu, Estonia
marlon.dumas@ut.ee
[2] University of Melbourne, Melbourne, Australia
{marcello.larosa,artem.polyvyanyy}@unimelb.edu.au
[3] Apromore, Melbourne, Australia
volodymyr.leno@apromore.com
[4] University of Bozen-Bolzano, Bolzano, Italy
maggi@inf.unibz.it

Abstract. User interaction logs allow us to analyze the execution of tasks in a business process at a finer level of granularity than event logs extracted from enterprise systems. The fine-grained nature of user interaction logs open up a number of use cases. For example, by analyzing such logs, we can identify best practices for executing a given task in a process, or we can elicit differences in performance between workers or between teams. Furthermore, user interaction logs allow us to discover repetitive and automatable routines that occur during the execution of one or more tasks in a process. Along this line, this chapter introduces a family of techniques, called Robotic Process Mining (RPM), which allow us to discover repetitive routines that can be automated using robotic process automation technology. The chapter presents a structured landscape of concepts and techniques for RPM, including techniques for user interaction log preprocessing, techniques for discovering frequent routines, notions of routine automatability, as well as techniques for synthesizing executable routine specifications for robotic process automation.

1 Introduction

The rigidity and complexity of legacy applications, particularly in large organizations, engender situations in which workers are required to perform repetitive routines to transfer data from one application to another via their user interfaces. Examples of such repetitive routines include:

- Downloading and opening an Excel workbook attached to an inbound email (e.g. a list of academic credentials of a prospective student) and copying data records from one of the sheets in this workbook (e.g. the credential details of the student) into a student admission system accessed via a web browser.
- Accesing a legacy ERP system to retrieve one or more purchase orders of a given customer, copying data from each of these purchase orders into a consolidated sheet, and sending the resulting spreadsheet to a customer by email.

Postprint, September 2022

The automation of such routines can eliminate tedious and demotivating manual work, reduce cycle times, and enhance data quality. Advances in Robotic Process Automation (RPA) technology [1, 38] make it possible to automate routines like the above ones. However, building and maintaining RPA bots requires a significant investment and hence, it is important for organizations to make the right decisions as to which bots they should build. In a typical organization, there may be tens of thousands of types of tasks, and any of them may involve one or more repetitive routines. Some routines are sufficiently frequent and widespread across the organization that they can be identified and scoped via interviews, focus groups, and workshops with workers. Other routines, however, may be less widespread or performed sporadically, but still sufficiently often that it is beneficial to automate them.

Robotic Process Mining (RPM) is a family of techniques to discover repetitive routines that can be automated using RPA technology, by analyzing interactions between one or more workers and one or more software applications, during the performance of one or more tasks in a business process. In general, RPM techniques take as input User Interaction logs (*UI logs*).[5] These UI logs are recorded while workers interact with one or more applications, typically desktop applications. Based on these logs, RPM techniques produce specifications of one or more routines that can be automated using RPA or related tools.

Depending on the type of technique, the discovered routine specifications may be conceptual (i.e. non-executable) or *executable*. A conceptual routine specification provides guidance to analysts and developers to help them scope a repetitive routine and to build an executable script to fully or partially automate the routine. For example, a non-executable specification of a routine could take the form of a textual description (in natural language), or a sequence of screenshots corresponding to repetitive sequences of interactions, or a sequence of user interactions (e.g. ["open sheet","select cell", "edit cell", "copy cell contents", ...]). An executable routine specification is a specification that contains all the information required to fully reproduce the routine via a dedicated execution engine or to synthesize a script that can be executed using an RPA tool or a similar type of automation tool.

This chapter reviews the state of the art in the field of RPM and provides a structured overview of the steps of a typical RPM pipeline, the techniques that may be employed in each of these steps, as well as open research challenges on the way to realizing mature RPM tool sets.

The chapter is partially based on a previous journal article [29]. The chapter extends this journal article by positioning the vision of RPM within the broader context of task mining and process mining, and by providing an updated review of related work in the field.

The rest of the chapter is structured as follows. Section 2 gives an overview of techniques related to robotic process mining, including task mining and process mining, and gives an overview of existing work on identification of task automa-

[5] In this chapter, we use the acronym *UI* to refer to a *user interaction*, not to be confused with a *user interface* which is another common use of this acronym.

tion opportunities. Section 3 presents a framework for robotic process mining and introduces techniques covering each component of the framework. Finally, Section 4 discusses open challenges in the field of robotic process mining.

2 Background

2.1 Robotic Process Automation

RPA is a class of tools to automatically execute sequences of steps (herein called *routines*) involving interactions between a user and a software application, or interactions between multiple applications via Application Programming Interfaces (APIs). In an RPA tool, the execution of a routine is driven by a pre-specified script, which consists of atomic steps corresponding to individual interactions, assembled together via control-flow structures (if-then-else statements, repeat-until loops, etc.) [40]. A common characteristic of RPA tools is that they are able to "operate on the user interfaces of computer systems in the way a human would do" [1]. For example, an RPA tool may perform clicks or keystrokes on the user interface of a desktop application to mimic a sequence of steps that would normally be performed by a human operator. Examples of RPA tools, as of the time of writing this chapter, include Automation Anywhere RPA Workspace[6], Blue Prism Intelligent Automation Platform[7], Microsoft Power Automate Desktop[8], RocketBot[9], and UiPath Platform.[10]

Typically, RPA tools include a design environment, where different types of users, ranging from software developers to business users, may specify and test scripts to automate one or more routines. Each such script is then embedded into a so-called *software bot*. A bot is a unit of execution in an RPA tool. A bot is responsible for executing a given script whenever a given type of trigger occurs. Bots are operated via so-called control dashboards, which allow human operators to oversee the work performed by a collection of bots.

Depending on how the control dashboard is used, we can distinguish two RPA use cases: *attended* and *unattended* [40]. In attended use cases, the bot is triggered by a user. During its execution, an attended bot may provide data to a user and take in data from a user. In these use cases, the user may run the bot's script step-by-step, pause or stop the bot, or otherwise intervene during the script's execution. Attended bots are suitable for routines where dynamic inputs are required (i.e. inputs gathered during a routine execution), where some decisions or checks require human judgment, or when the routine is likely to have unforeseen exceptions. For example, 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

[6] `https://www.automationanywhere.com/`
[7] `https://www.automationanywhere.com/`
[8] `https://powerautomate.microsoft.com/`
[9] `https://www.rocketbot.com/`
[10] `https://www.uipath.com/`

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 an unattended bot could execute. In this chapter, we focus on unattended RPA bots.

Figure 1 presents a simple lifecycle model of RPA bots, which we use below to position the role of robotic process mining.[11] According to this lifecycle model, an RPA bot goes through four phases:

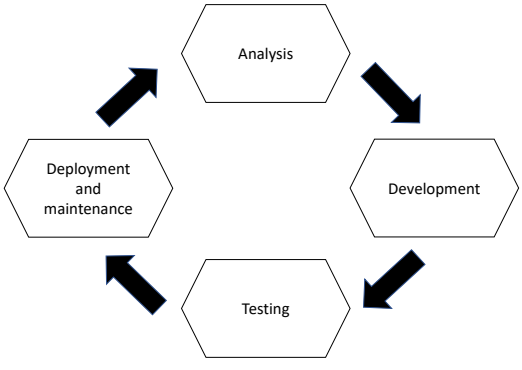

Fig. 1. Simple RPA bot lifecycle [20]

- ***Analysis.*** In this phase, analysts identify candidate routines for automation, examine the current ways of their execution (e.g. by constructing the *as-is* process model), assess the costs and benefits of their automation as well as the related risks, and analyze whether the identified routines can be automated without being redesigned.
- ***Development.*** In this phase, the routines identified earlier are automated. This involves constructing a process model representing the desired execution of the routines to be automated (i.e. the *to-be* process model). Then RPA developers implement the routine using a specialized development environment by creating an executable software script, a.k.a. RPA bot. Depending on the complexity of the task to be automated, this requires a different amount of coding. Large enterprise RPA tools such as UiPath or Automation Anywhere allow for the creation of the scripts by dragging and dropping the required functions (e.g. open a file, copy a cell). Since this step requires a

[11] For the sake of conciseness, the RPA bot lifecycle model discussed here consists of four coarse-grained phases. A finer-grained RPA bot lifecycle can be found, for example, in [13].

large amount of manual, error-prone work, a code review and script evaluation are required.
- ***Testing.*** In this phase, the implemented bot undergoes testing in a pre-production environment. It is evaluated in the different scenarios to examine whether it works as intended and how it handles exceptions. If the tests are successful, the bot proceeds to the deployment phase. If the tests fail, it is sent back to the developers to identify and fix the identified issues.
- ***Deployment and maintenance.*** After successful testing, the bot is deployed in the production environment and is ready to be used via a control dashboard. As the bot performs its work, certain issues may arise. In this case, the bot may be sent back to the testing or development phases.

In this chapter, we focus on techniques that leverage UI logs to support the analysis and development phases of RPA bots.

2.2 Task Mining

Task mining is a collection of techniques for analyzing the execution of tasks performed by human workers, based on records of interactions between these workers and one or more software applications. Depending on the goal of the analysis, we can distinguish between three use cases of task mining [23]: (i) task discovery and optimization; (ii) resource and workforce optimization; and (iii) task automation.

Task discovery and optimization. In this use case, the goal is to discover how a task is performed by one or more workers, to identify deviations with respect to policies or work instructions related to that task, and/or to uncover ways of improving the performance of the task. By applying task mining techniques to a task, we may discover that different workers perform the task in different ways. For example, one worker might open all the desktop windows required to perform a task upfront (e.g. an email client, a spreadsheet application, and a browser window connected to a CRM system), and only once all windows are open, they start navigating across these windows to complete the task. Another worker might start performing the task in one desktop window (e.g. the email client's window) and then open the other windows incrementally. Similarly, one worker might usually execute a task in a single go, without interruptions, while another might interleave the execution of the task with other work, or might multitask.

Having identified how a task is performed by one or more resources, task mining can help us to identify steps in a task that are responsible for delays (bottlenecks), as well as common rework loops or workarounds with respect to normative work instructions. Task mining also allows us to relate the sequences of steps that different workers perform with performance measures, such as the mean cycle time of a task or the defect rate of a task. For example, task mining may help us to identify that when a given step, such as clicking on a given cell number in a sheet, is repeated multiple times, the mean cycle time of a task is significantly higher than when this cell is visited only once.

Resource and workforce optimization. In this use case, the goal is to identify inefficiencies in the way tasks are assigned to resources, or conversely, to uncover ways to improve the assignment of tasks. For example, by analyzing UI logs, we may find that when an invoice entry task relates to an invoice from a company in country X, it takes more time for worker A to perform the task (rather than another worker B) whereas the opposite holds for invoices coming from country Y. We might also find that when worker A performs an invoice data entry task after 4:30pm, the task gets completed faster, but when this happens, some fields in the invoice are left unfilled, which might then be causing issues downstream.

Task automation and robotic process mining. In this use case, the goal is to discover opportunities to automate a task or part of a task. The automation of a task can be achieved using a variety of technologies. For example, if a task involves information flows between multiple applications, one could use middleware technology to programmatically connect these applications, thus replacing the manual information flow with an automated (programmatic) flow. Another approach is to develop and RPA bot to transfer data from one application to another by replicating the user interactions that a human worker would do to achieve this. Robotic Process Mining (RPM) refers to the use case of task mining where UI logs are analyzed in order to identify frequent routines that can be automated by means of one or more RPA bots. The rest of this chapter focuses on this latter use case of task mining.

2.3 Relations Between Task Mining and Process Mining

Task mining is in many ways related to process mining, particularly to techniques for automated process discovery (cf. Chapters 2 and 3). However, task mining and process mining differ in several respects. These differences stem from the differences in the inputs of these techniques. Process mining take as input event logs extracted from enterprise systems that support the execution of one or more business processes in an organization – e.g. Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) systems, as discussed in Chapter 1. Meanwhile, task mining techniques take as input UI logs, consisting of records of micro-steps performed by workers while they interact with software applications to perform individual tasks in a process. Both types of logs consist of timestamped records, such that each record refers to the execution of an action (or task) by a user. Also, each record may contain a payload consisting of one or more attribute-value pairs. However, UI logs and event logs differ in at least four ways.

First, event logs intended for process mining consist of events at a finer level of granularity than UI logs. An event in an event log typically refers to the start, completion or other significant state change in the execution of a 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 (micro-)steps, which may be

recorded in an 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. In other words, an UI log may contain dozens or even hundreds of entries per task execution, whereas an event log would typically only contain one or a handful of entries per task execution. Also, the payload of the events in an event log may contain low-level information such as the specific cell or the pixel coordinates involved in a user interaction, or it may be associated to a screenshot taken during a user interaction. In contrast, event logs contain business-relevant attributes, such as the amount of a loan offer, the interest rate, the repayment term, etc.

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 an UI log are not explicitly correlated. A typical UI log consists of thousands of user interactions recorded during a period of several hours on the workstation(s) of one or more workers. Prior to being used, such UI logs needs to be segmented into logical units corresponding to task executions, as discussed later in this chapter.

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. address). Meanwhile, an UI log typically collects all the data observed during the execution of a task, particularly when the UI log is intended to be used for RPM purposes. Indeed, 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 a patient arrives by themselves, but it is recorded when a 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. Meanwhile, an 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.

The above differences in the input entail that it is often not possible nor desirable to use the same techniques for process mining as for task mining. In the field of process mining, a typical visualization consists of a graph with one node per activity. The emphasis of these techniques is to show the most frequent control-flow dependencies between the activities of the process. This approach is not feasible in the context of task mining because the steps are fine-

grained and therefore too numerous to be displayed in their entirety. Besides, only certain steps are relevant for a given use-case, specifically those that are part of a frequent routine. Accordingly, a task mining technique typically starts by pre-processing the UI log in order to extract only the most frequent sequences of steps (i.e. the most frequent routines) using sequence pattern mining techniques, or using event abstraction techniques such as those developed in the field of process mining [41].

Notwithstanding these differences, several commercial process mining vendors, such as Apromore[12], Celonis[13], and Minit[14], take advantage of the commonalities between UI logs and business process event logs to offer task mining features. Typically, these tools discover directly-follows graphs (cf. Chapter 2) from UI logs or from combinations of event logs and UI logs. For example, these tools may discover directly-follows graphs to visualize the sequences of screens visited by a user during the performance of one or more tasks, or to visualize the most frequent or the slowest steps during the performance of a task.

These visualizations are suitable when analyzing tasks for the purpose of task optimization and workflow optimization (cf. the first two use-cases above). They can also help users to visually detect candidate routines for automation, when those routines have a simple structure (e.g. perfect sequences of steps). However, beyond simple scenarios, these visualizations do not allow users to determine if a given task contains routines that can be automated by means of an RPA bot. In this respect, RPM techniques complement task mining techniques by explicitly addressing the questions of: (1) how to identify candidate routines for automation? and (2) how to derive an executable specification of a routine that has been identified as a candidate for automation?

3 Robotic Process Mining: A Framework

RPA tools are able to automate a wide range of routines, raising the question *how to identify routines in an organization that may be beneficially automated using RPA?* [38] To address this question, we envision a new class of tools, namely Robotic Process Mining (RPM) tools.

We define RPM as *a class of techniques and tools to analyze data collected during the execution of user-driven tasks to support identifying and assessing candidate routines for automation and discovering routine specifications that RPA bots can execute.* 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 primary source of data for RPM tools consists of user interaction (UI) logs. RPM aims at assisting the analysts in drawing a systematic inventory of candidate routines for automation and help them to produce executable specifications that can be used as a starting point for their automation.

[12] `https://apromore.com`
[13] `https://celonis.com`
[14] `https://minit.io`

3.1 UI Logs and Routines

Fig. 2 presents a class diagram capturing the core concepts and RPM and their relations. In this class diagram, the two main concepts are User Interaction log (*UI log*) and Routine. UI logs are the input of RPM, while routines (represented as routine specifications or as RPA scripts) are the output of RPM.

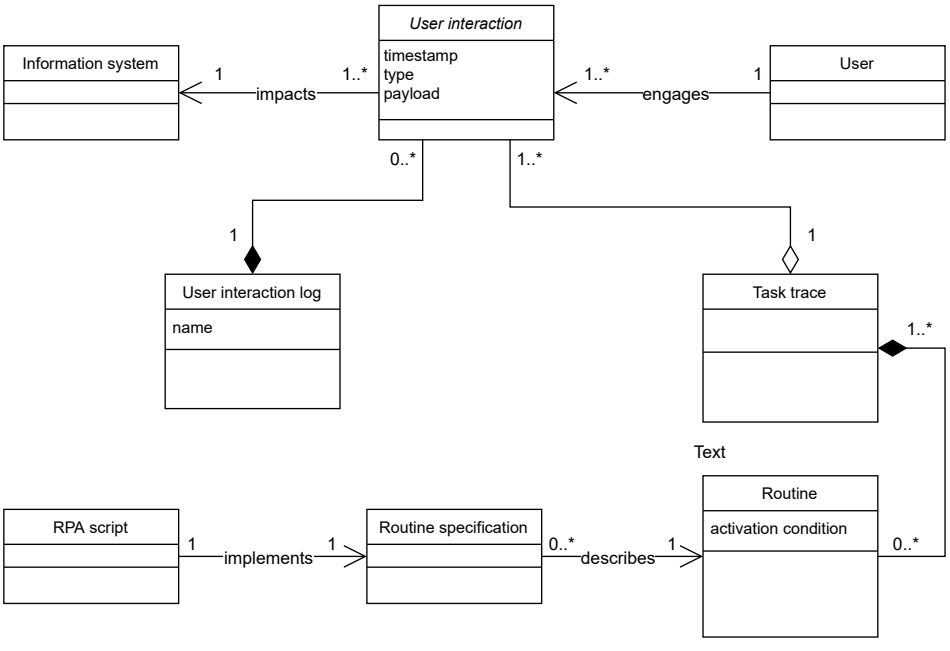

Fig. 2. Class diagram of RPM concepts

An UI log is a chronologically ordered sequence of user interactions, or UIs in short, performed by a single user in a single workstation and involving interactions across one or more applications (including web and desktop applications). An example of an UI log, which we use herein as a running example, is given in Table 1.

Each row in this example corresponds to one UI (e.g. clicking a button or copying the content of a cell). Each UI is characterized by a *timestamp*, a *type*, and a set of *parameters*, or *payload* (e.g. application, button's label or value of a field). To be useful in the context of RPA, the payload should contain sufficient information for a software bot to reproduce the performed activity. For example, for a UI 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. The

Row	UI Timestamp	UI Type	Payload P_1	P_2	P_3	P_4	P_5	P_6
1	2019-03-03T19:02:23	Navigate to (web)	https://www.unimelb.au	204	google search	–	–	–
2	2019-03-03T19:02:26	Click Button (web)	https://www.unimelb.au	New record	newRecord	Button	–	–
3	2019-03-03T19:02:28	Select Cell (Excel)	StudentRecords	Sheet1	A	2	“John”	–
4	2019-03-03T19:02:31	Select Field (web)	https://www.unimelb.au	First Name	first	input	“”	–
5	2019-03-03T19:02:37	Edit Field (web)	https://www.unimelb.au	First Name	first	input	“John”	–
6	2019-03-03T19:03:56	Create New Tab (web)	https://chrome/new-tab/	219	New Tab	–	–	–
7	2019-03-03T19:03:56	Select Tab (web)	https://chrome/new-tab/	219	New Tab	–	–	–
8	2019-03-03T19:04:05	Navigate to (web)	https://www.facebook.com	219	New Tab	–	–	–
9	2019-03-03T19:07:50	Select Tab (web)	https://www.unimelb.au	204	New record	–	–	–
10	2019-03-03T19:08:02	Select Field (web)	https://www.unimelb.au	Last Name	last	input	“”	–
11	2019-03-03T19:08:05	Edit Field (web)	https://www.unimelb.au	Last Name	last	input	“Do3”	–
12	2019-03-03T19:08:08	Select Field (web)	https://www.unimelb.au	Last Name	last	input	“Do3”	–
13	2019-03-03T19:08:12	Edit Field (web)	https://www.unimelb.au	Last Name	last	input	“Doe”	–
14	2019-03-03T19:08:16	Select Field (web)	https://www.unimelb.au	Birth Date	date	input	“”	–
15	2019-03-03T19:08:20	Edit Field (web)	https://www.unimelb.au	Birth Date	date	input	“18-11-1992”	–
16	2019-03-03T19:08:24	Select Field (web)	https://www.unimelb.au	Country of residence	country	input	“”	–
17	2019-03-03T19:08:27	Edit Field (web)	https://www.unimelb.au	Country of residence	country	input	“Australia”	–
18	2019-03-03T19:08:31	Click Button (web)	https://www.unimelb.au	Submit	submit	submit	–	–
19	2019-03-03T19:08:35	Click Button (web)	https://www.unimelb.au	New record	newRecord	Button	–	–
20	2019-03-03T19:08:38	Select Cell (Excel)	StudentRecords	Sheet1	A	3	“Albert”	–
21	2019-03-03T19:08:40	Copy Cell (Excel)	StudentRecords	Sheet1	A	3	“Albert”	“Albert”
22	2019-03-03T19:08:42	Select Field (web)	https://www.unimelb.au	First Name	first	input	“”	–
23	2019-03-03T19:08:43	Paste (web)	https://www.unimelb.au	First Name	first	input	“”	“Albert”
24	2019-03-03T19:08:44	Edit Field (web)	https://www.unimelb.au	First Name	first	input	“Albert”	–
25	2019-03-03T19:08:47	Select Cell (Excel)	StudentRecords	Sheet1	B	3	“Rauf”	–
26	2019-03-03T19:08:49	Copy Cell (Excel)	StudentRecords	Sheet1	B	3	“Rauf”	“Rauf”
27	2019-03-03T19:08:52	Select Field (web)	https://www.unimelb.au	Last Name	last	input	“”	–
28	2019-03-03T19:08:53	Paste (web)	https://www.unimelb.au	Last Name	last	input	“”	“Rauf”
29	2019-03-03T19:08:54	Edit Field (web)	https://www.unimelb.au	Last Name	last	input	“Rauf”	–
30	2019-03-03T19:08:59	Select Cell (Excel)	StudentRecords	Sheet1	C	3	“08/09/1989”	–
31	2019-03-03T19:09:02	Copy Cell (Excel)	StudentRecords	Sheet1	C	3	“08/09/1989”	“08/09/1989”
32	2019-03-03T19:09:07	Select Field (web)	https://www.unimelb.au	Birth Date	date	input	“”	–
33	2019-03-03T19:09:10	Paste (web)	https://www.unimelb.au	Birth Date	date	input	“”	“08/09/1989”
34	2019-03-03T19:09:12	Edit Field (web)	https://www.unimelb.au	Birth Date	date	input	“08-09-1989”	–
35	2019-03-03T19:09:17	Select Cell (Excel)	StudentReords	Sheet1	D	3	“Germany”	–
36	2019-03-03T19:09:21	Copy Cell (Excel)	StudentRecords	Sheet1	D	3	“Germany”	“Germany”
37	2019-03-03T19:09:26	Select Field (web)	https://www.unimelb.au	Country of residence	country	input	“”	–
38	2019-03-03T19:09:32	Paste (web)	https://www.unimelb.au	Country of residence	country	input	“”	“Germany”
39	2019-03-03T19:09:35	Edit Field (web)	https://www.unimelb.au	Country of residence	country	input	“Germany”	–
40	2019-03-03T19:09:48	Edit Field (web)	https://www.unimelb.au	International Student	international	checkbox	TRUE	–
41	2019-03-03T19:09:54	Click Button (web)	https://www.unimelb.au	Submit	submit	submit	–	–
...	...	...	...	...	...	...		

Table 1. Fragment of a user interaction log

payload of a UI is not standardized and depends on the UI type and application. Consequently, the UIs recorded in the same log may have different payloads. For example, the payload of UIs performed within a spreadsheet contains information regarding the spreadsheet name and the location of the target cell (e.g. the cell's row and column). In contrast, the payload of the UIs performed in a web browser contains information regarding the webpage URL, the name and identifier of the UI's target HTML element, and its value (if any).

An UI log consists of interactions of different types. To illustrate the types of interactions that may be exploited in the context of robotic process mining, Table 2 provides the concrete list of UI types (and associated parameters) supported by the Action Logger tool [30]. Action Logger is an open-source UI recording tool designed to record events generated by browsers and desktop applications, in a way that enables the discovery of automatable routines.

Note that in Table 2, the UI types are grouped into three groups: navigation, read, and write UIs. Navigation UIs correspond to actions that affect the state of the user interface, but without reading or writing any data. This includes, for

example, moving from one tab to another in a broader, or selecting a cell in an Excel spreadsheet. Read actions are those where some data item is accessed, for example in order to copy it into the clipboard. Meantime, "write" actions are those where data is written into an element of the UI, for example, pasting the contents of the clipboard into the currently selected cell of an Excel spreadsheet.

UI Group	UI Type	Parameter Names					
		P1	P2	P3	P4	P5	P6
Navigate	Create New Tab (web)	URL	ID	Title			
	Select Tab (web)	URL	ID	Title			
	Close Tab (web)	URL	ID	Title			
	Navigate To (web)	URL	Tab ID	Tab title			
	Add Worksheet (Excel)	Workbook	Worksheet				
	Select Worksheet (Excel)	Workbook	Worksheet				
	Select Cell (Excel)	Workbook	Worksheet	Cell column	Cell row	Value	
	Select Range (Excel)	Workbook	Worksheet	Range columns	Range rows	Value	
	Select Field (web)	URL	Name	ID	Type	Value	
Read	Copy (web)	URL	Name	ID	Value	Copied content	
	Copy Cell (Excel)	Workbook	Worksheet	Cell column	Cell row	Value	Copied content
	Copy Range (Excel)	Workbook	Worksheet	Range columns	Range rows	Value	Copied content
Write	Paste Into Cell (Excel)	Workbook	Worksheet	Cell column	Cell row	Value	Pasted content
	Paste Into Range (Excel)	Workbook	Worksheet	Range columns	Range rows	Value	Pasted content
	Paste (web)	URL	Name	ID	Value	Pasted content	
	Click Button (web)	URL	Name	ID	Type		
	Click Link (web)	URL	Inner text	Href			
	Edit Field (web)	URL	Name	ID	Type	Value	
	Edit Cell (Excel)	Workbook	Worksheet	Cell column	Cell row	Value	
	Edit Range (Excel)	Workbook	Worksheet	Range columns	Range rows	Value	

Table 2. User interaction types and their parameters

To obtain an UI log suitable for RPM, all UIs 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 this task one after the other. During such a session, a worker is expected to perform a number of tasks of the same or different types. The UI log shown in the example above 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, date of birth, and country of residence. If the country of residence is not Australia, the worker needs to perform one more step, indicating that the student will be registered as an international student.

Each execution of a task (herein also called a *task instance*) is represented by a *task trace*. In our running example, there are two traces belonging to a "new record creation" task. From the log, we can see that the worker performed this task in two different ways. In the first case, she manually filled in the form (UIs 1 to 18), while in the second case, she copied the data from a worksheet and pasted it into the corresponding fields (UIs 19 to 41).

Given a collection of task traces, the goal of RPM is to identify a repetitive sequence of UIs that can be observed in multiple task traces, herein called a *routine*, and to identify routines amenable for automation. For each such routine,

RPM then aims at discovering an executable specification (herein called a *routine specification*). This routine specification may be initially captured in a platform-independent manner and then compiled into a platform-dependent *RPA script* to be executed in a given RPA tool.

3.2 RPM Phases

We distinguish three main phases in RPM: (1) collecting and pre-processing UI logs corresponding to the executions of one or more tasks; (2) discovering candidate routines for RPA; and (3) discovering executable RPA routines.[15]

Collecting and pre-processing UI logs. We decompose the first phase into the recording step itself and two preprocessing steps, namely the segmentation of the log into task traces and the simplification of the resulting task traces. We map the second phase into a single step. Then, we decompose the third phase into three steps: the discovery of platform-independent routine specifications, the aggregation of routines with the same effects, and the compilation of the discovered specifications into platform-specific executable scripts. This decomposition of the three phases into steps is summarized in the RPM pipeline depicted in Fig. 3. Below we discuss each step of this pipeline.

The recording of an UI log involves capturing low-level UIs, such as 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 web browsers) used by the workers via plug-in or extension mechanisms. Logs collected by such plug-ins or extensions may be merged 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 be preprocessed to be suitable for RPM.

The main challenge in this step is to identify what UIs must be recorded. The same UI (e.g. mouse click) can either be important or irrelevant in a given context. For example, a mouse click on a button is an important UI, but a mouse click on a web page's background is an irrelevant UI. Also, when a worker selects a web form, we need to record UIs 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 and window sizes. Existing UIs recording tools,

[15] Once an RPA routine has been automated via an RPA bot, a fourth phase is to monitor this bot to detect anomalies or performance degradation events that may signal that the bot may need to be adjusted and 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 [17].

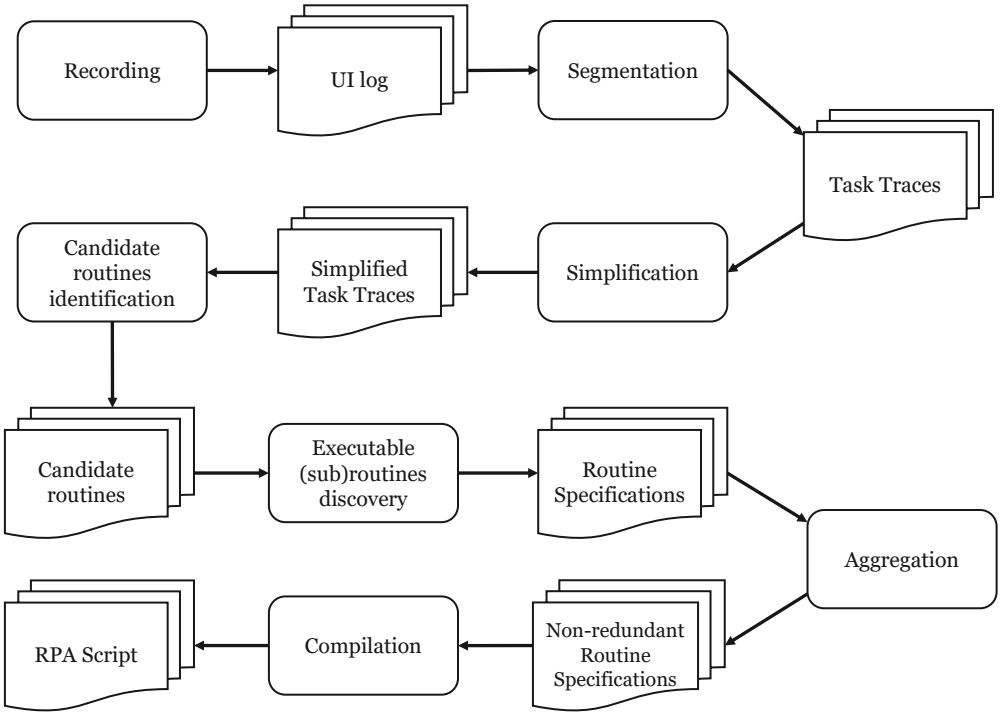

Fig. 3. RPM pipeline

such as JitBit Macro Recorder[16], TinyTask[17], and WinParrot[18], save all the UIs performed by the user at a too low level of granularity, with reference to pixel coordinates (e.g. click the mouse at coordinates 748,365). As a result, the UI logs generated by these tools are not suitable for extracting useful routines. The RPA tools mentioned in Section 2.1 (e.g. UiPath and Automation Anywhere) provide recording functionality. However, this functionality is intended to record RPA scripts. These tools do not capture details about different fields' values, as these values are not relevant for RPA script generation. For example, an RPA script must know which cell in a spreadsheet has to be copied, and it is agnostic to the value stored in that cell. Hence, a new family of recording tools is needed to record UI logs required for RPM.

In [30], we 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

[16] `https://www.jitbit.com/macro-recorder/`
[17] `https://www.tinytask.net/`
[18] `http://www.winparrot.com/`

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 tool, 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 achieve a more general solution, it may be necessary to combine this latter approach with OCR technology in order to detect UI events and associated data from application screenshots, as outlined in [35, 32].

In its raw form, an UI log consists of one single sequence of UIs recorded during a session. During this session, a user may have performed several executions of one or multiple tasks, that may be mixed up in the log. Moreover, in case of multi-tasking, UIs of multiple concurrent task executions may be mixed together. Before identifying candidate routines for automation, an UI log has to be segmented into task traces, such that each trace corresponds to the execution of one task instance. This involves the identification of the boundaries of the tasks and the assignment of UIs to specific task traces. Given the fragment of the UI log demonstrated in the running example, we can extract two segments, each corresponding to the processing of a specific entry in the spreadsheet containing students' data (UIs 1 to 18 and 19 to 41 in Table 1).

The problem of extracting segments from an UI log corresponding to task instances is similar to that of web session reconstruction [37], where the goal is to identify the beginning and the end of web navigation sessions in server log data (e.g. streams of clicks and web page navigation) [37]. Methods for session reconstruction are usually based on heuristics that rely on the structural organization of websites or time intervals between events. The former approach covers only the cases where all the user interactions are performed in the web applications. In contrast, the latter approach assumes that users make breaks in-between two consecutive segments – in our case, two routine instances.

The problem of segmentation is also related to that of preprocessing so-called *uncorrelated event logs* in process mining. As discussed in Chapters 1 and 2, each event in a log should include, as a minimum, a case identifier, a timestamp, and an activity label. When the events of an event log do not have a case identifier, the log is said to be uncorrelated. Various methods have been proposed to extract correlated (i.e. regular) event logs from uncorrelated ones. However, existing methods in this field address the problem in restrictive settings. Specifically, some approaches [14] assume that the underlying process is acyclic, while others [8, 7] assume that an explicit process model is given as input (in addition to the uncorrelated event log). These assumptions do not hold in the context of RPM, where no explicit process model is available, and a routine may contain repetitions. Also, the above approaches sometimes produce inaccurate results, whereas in the context of RPM, we need to identify routines with high levels of confidence (preferably 100% confidence), since an inaccurate replication of a routine by an unattended RPA bot may lead to costly errors.

In some scenarios, segmentation may be accomplished by combining transactional data recorded by enterprise information systems and user interactions logs, as proposed in [32]. However, a shortcoming of this approach is that such

transactional data often provides only limited information about the process context, which is not enough to identify the boundaries of tasks captured in the user interactions logs.

Recent work on UI log segmentation [3, 5] proposes to use trace alignment between the logs and the corresponding interaction models to identify the segments. In practice, however, such interaction models are not available beforehand.

Another related work [27] proposes to discover segments in the log by identifying cycles in the graph constructed from this log. These cycles represent repetitive behavior in the log and thus potentially correspond to task instances recorded in the log. However, this approach assumes that the task instances recorded in the log do not overlap and occur consequently one after the other.

In the context of desktop assistants, research proposals such as TaskTracer and TaskPredictor have tackled the problem of analyzing UI logs generated by desktop applications to identify the current task performed by a user and to detect switches between one task and another [36, 12]. These approaches can potentially be used to split the UI logs into segments corresponding to different tasks. However, such approaches are not able to distinguish different instances of the same task.

Ideally, UIs recorded in a log should only relate to the execution of the task(s) of interest. However, in practice, a log often also contains UIs that do not contribute to completing the recorded task(s). We can consider such UIs to be *noise*. Examples of noise UIs include a worker browsing the web (e.g. social networking) while executing a task that does not require doing that, or a worker committing mistakes (e.g. filling a text field with an incorrect value or copying a wrong cell of a spreadsheet). UIs 6, 7, 8, 9, 10, and 11 are noise in our running example. During the creation of the student record, the worker decided to make a small pause, switched to a new tab in the web browser (6-7), and navigated to Facebook (8), where she spent almost 4 minutes browsing the news feed, before going back to the tab with the active student form (9). All these UIs do not have any relation to the task being recorded; thus, they constitute noise. When performing the task, the worker selected a surname field in the form (10) and made a mistake by accidentally misspelling the surname of the student (11). She then had to select the same field again (12) and fill it in with the correct value (13). Although the UIs 10 and 11 belong to the performed task, their effects are overwritten by successive UIs (e.g. UI 11 is overwritten by UI 13) and, therefore, they do not affect the outcome of the routine and are considered to be noise. The presence of the noise may negatively affect the subsequent steps of the RPM pipeline (e.g. the discovery of the candidate routines). Accordingly, the next step in the RPM pipeline is *simplification*, which aims at noise identification and removal. The UIs in the log are removed so that the resulting log captures the same effects as the original one while being simpler (i.e. having fewer UIs).

One of the challenges that arises during the pre-processing step of the RPM pipeline is to separate irrelevant UIs (i.e. noise) from those UIs that do contribute to the completion of a task. A possible approach is to assume that noise takes the form of chaotic events that may happen anywhere during process execution.

One technique for filtering out such chaotic events is described in [39]. 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 not filter it out. Moreover, some UIs 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 UIs. Thus, it is important to consider the data perspective, i.e. values of data objects that are manipulated by the UIs. In this way, one can identify the UIs that share the same parameter values (e.g. copying a value from a worksheet and then pasting it in a web form), or have the same source/origin (e.g. all the UIs are performed on the same website). The UIs that do not share any data parameters and/or values or originate from different sources most likely constitute noise.

Discovering candidate routines for automation. Given a set of simplified task traces, the next phase is to identify candidate routines for automation. This phase aims at extracting repetitive sequences of UIs that occur across multiple task traces, a.k.a. routines, and to identify which of those routines are amenable for automation. The output of this step is a set of candidate routines for automation.

Even though an automated RPM tool can considerably reduce the effort required to automate routine, there is still a lot of development, quality assurance, and maintenance effort required to automate a routine in a real-life setting. Also, the automation of a routine may require re-training and re-allocation of human workers involved in the process. And if the routine is only partially automated (as opposed to fully automated), some handoffs will have to be put in place between the manual and the automated parts of a routine. As a result, the costs of automating a routine may sometime (or even often) outweigh the benefits. Thus, the cost-benefit analysis of routine automation is an important step in an end-to-end RPM method. To perform this analysis, a first step is to assess is a routine is suitable for automation.

Mindful of this requirement, Lacity and Willcocks [24] propose high-level guidelines for determining if a task is a candidate for automation in the context of a case study at Telefonica. The guidelines, however, do not provide a formal and precise definition of what makes a routine suitable for automation.

In a recent systematic review of the RPA literature, Syed et al. [38] conclude that "there is a need for formal, systematic and evidence-based techniques to determine the suitability of tasks for RPA.". In other words, a major challenge in the field of RPM is how to formally characterize what makes a routine amenable for automation via RPA or other automation technologies.

Two necessary criteria for a routine to be amenable for automation are:

1. *Frequency* [17] The more frequently a routine is performed, the more its automation is likely to lead to significant reductions in processing times, waiting times, and defects (due to human mistakes).
2. *Determinism* [31, 9]. A candidate routine for automation should be such that a software bot is always able to determine the next step it should

perform next in order to complete an execution of the routine. In other words, a routine can be automated only if: (1) every UI in the routine is deterministically activated, meaning that we know when to execute it (e.g. the box *International* is ticked whenever the student's country of residence is not Australia); and (2) every UI in the routine relies only on data produced by previous UIs (e.g. one of the UIs in the routine consists in entering the country of birth of a student into a field of a web form, and this data item has been previously copied from a cell of a spreadsheet in a previous UI).

Considering the running example provided in Table 1 and assuming that the identified task traces frequently occur in the log, we would discover two candidate routines, handling the domestic and international students, respectively. Note that the routine in the first task trace is only partially automatable. The worker manually filled in the form by looking at the corresponding entry values in the spreadsheet. Since she did not read the data values explicitly (e.g. by copying the values to the clipboard), these values are unknown for the recording tool. Hence, it is not possible to understand how the values used for editing the form's fields were obtained. On the other hand, the routine from the second task trace is fully automatable, as it is clear how to compute the values for the fields of the web form in the target application (i.e. by copying them from the spreadsheet).

Several techniques proposed in the field of UI log mining address the problem of identifying routines that fulfill the "frequency" criterion. Dev and Liu [11] have noted that the problem of frequent routine identification from (segmented) UI logs can be mapped to that of frequent pattern mining, a well-known problem in the field of data mining [19]. In the literature, several algorithms are available to mine frequent patterns from sequences of symbols. Depending on their output, we can distinguish two types of frequent pattern mining algorithms: those that discover only exact patterns [25, 34] (hence vulnerable to noise), and those that allow frequent patterns to have gaps within the sequence of symbols [42, 15] (hence noise-resilient).

Bosco et al. [9] address the problem of discovering routines that fulfill the "determinism" requirement. Specifically, this technique discovers sequences of actions such that the input(s) of each action in the sequence (except the first one) can be derived from the data observed in previous actions. However, this technique can only discover perfectly sequential routines and is hence not resilient to noise and variability in the order of the actions.

Leno et al. [26, 28] combine techniques for discovering frequent routines, with techniques for discovering deterministic routines, thus addressing both of the above requirements. This latter proposal also addresses the problem of synthesizing an executable routine specification and that of detecting semantically equivalent routines, as discussed later in this chapter.

The discovery of automatable routines from sequences of actions is related to the problem of automated process discovery, discussed in Chapters 2 and 3 of this handbook. This relation is explored by Jiménez-Ramírez et al. [21], who apply process discovery techniques to extract process models from segmented UI logs. Importantly though, while it is possible to use automated process discovery

algorithms to extract process models from segmented UI logs, the resulting process models cannot readily be used for automation (via RPA or other automation technology) for two reasons.

First, the process models discovered by process discovery techniques, such as those presented in Chapters 2 and 3, are control-flow models. They capture the occurrence and order of steps (tasks) in a process, but not the data taken as input and produced as output by each step in the process. Yet, in order to automate a routine, we need to know which data is used by each step in the routine and where these data comes from. We note that a subset of process discovery approaches can discover process models with data-driven branching conditions [10], or process models where some control-flow relations only hold under certain data-driven conditions [33], but they do not discover process models with data manipulation logic.

Second, the process models produced by automated process discovery techniques, typically contain traces that have not been observed (cf. the *generalization* property discussed in Chapter 2). However, when the purpose of a model is to serve as a blueprint for RPA, the generalization property is not desirable. Indeed, if a software bot executes such a model, it will sometimes produce sequences of action that might not correspond to a sequence of actions that a human worker would have performed. This, in turns, may lead to errors and these errors may later require time-consuming and costly corrective actions. Instead, routines for RPA must be 100% precise (cf. the definition of precision in Chapter 2), as a lack of precision may lead to potential errors when the routines are executed by an unattended RPA bot.

Discovering executable routine specifications. Having identified a set of candidate routines for automation, the next step is that of *executable (sub-)routine discovery*. For each candidate routine, this step identifies the *activation condition* (UIs 2 and 19 in Table 1), which indicates when an instance of the routine should be triggered, and the *routine specification*, which specifies what UIs should be performed within that routine, what data is used by each UI in the routine, and how these data should be obtained.

The discovery of a routine specification involves identifying and synthesizing the transformation functions that have to be applied to the input data to convert it to the required format in the target application. In the running example, we can see that the web form requires a different date format than the one used in the spreadsheet (UIs 29 to 34). Hence, transferring the date of birth via simple copy and paste operations is insufficient, and the transformation function must be applied to achieve the desired result.

The problem of discovering executable routine specifications has been widely studied in the context of table auto-completion and data wrangling. For example, the Excel's Flash Fill feature detects string patterns in the values of the cells in a spreadsheet and uses these patterns for auto-completion [18]. Similarly, the authors in [6] propose an approach to extract structured relational data from semi-structured spreadsheets. However, such approaches can discover only the

executable routines performed in one application and have a limited area of usage. In practice, the RPA routines often involve many of these applications.

Bosco et al. [9] suggest that the discovery of executable routine specifications can be tackled by applying methods for automated discovery of data transformations from examples [2, 22]. However, these methods suffer from scalability issues when applied naively. Leno et al. [26] explore this approach and propose a series of optimizations to improve performance of the data transformation discovery techniques in the context of synthesis of routine specifications for RPA. This approach is further elaborated by the same authors in [28].

Gao et al. [16] extract rules from segmented UI logs to automatically fill in (web) forms. However, this approach only discovers branching conditions that specify whether a given activity has to be performed or not (e.g. check a box in a form) and only focuses on copy-paste operations without identifying more complex manipulations.

Agostinelly et al. [4] present an approach to discover routines from segmented UI logs and automate these routines via scripts. This approach, however, assumes that all the actions within a routine are automatable. In practice, it is possible that some actions have to be performed manually, and they can not be automated.

The output of the *executable (sub)routine discovery* step is a set of executable routine specifications of each automatable candidate routine. However, some of these specifications may produce identical effects, as they describe different variants of the same routine (e.g. filling in a web form in different orders). These variants are considered as duplicates and should be ignored, as their automation will not bring any benefits to the organization. Therefore, the next step in the RPM pipeline is *aggregation*. During this step, the discovered routine specifications leading to the same effects are replaced with one specification that captures the optimal way of performing the underlying routine. Several routine specifications may also be combined into a more complex specification that contains instructions on how to deal with different cases.

Once the script has been generated, it may be manually refined by an RPA developer, tested, and deployed into a production environment. The bot can 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 and can be paused during execution, so that the user can make small corrections if needed and then resume the work. In unattended settings, the bot works independently without human involvement.

4 Outlook

There are a number of research challenges that need to be overcome to realize the vision of RPM, particularly in the areas of candidate routine discovery, extraction of automatable routines, and aggregation of equivalent routines (cf. Figure 3).

In the area of candidate routine identification (and the related area of UI log segmentation), existing techniques assume that the routine instances are strictly separated in the UI log, i.e. there is no interleaving of user interactions belonging to one instance of one routine, and user interactions belonging to another instance of the same or of another routine. In practice, such interleaving may occur, for example, when a user is multi-tasking and thus alternating their attention between multiple routines.

In the area of automatable routine discovery, existing techniques are based on data transformation discovery, and as such they are limited to data transfer routines, where the goal is to take data from one system and transfer them to another system. Furthermore, these techniques are limited in scope to discovering routines where one record in one application, e.g. one row of a spreadsheet, is copied into one or more fields of another application (e.g. a web form). In reality, a single routine may involve complex iterations, for example, a routine may involve copying an invoice containing multiple invoice line-items from one application to another. In this case, the top-level routine (copying an invoice) contains a nested iterated sub-routine (copying multiple line items). These kind of structures cannot be discovered via existing data transformation discovery techniques. These latter techniques can discover that there is a routine consisting in copying an invoice line item, but they cannot reason holistically about the higher-level routine where the entire invoice is copied.

The area of routine aggregation is still a green field of research. A fundamental open problem in this space is the definition of notions of routine equivalence that would allow us to detect, for example, that a routine performed by one worker is the same as the one performed by another worker, even though these two workers perform the steps in their respective routines in completely different ways.

The RPM techniques discussed in this chapter focus 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, which goes beyond the scope of 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, a bot may detect that this order comes from China and then it may stop because it does not know how to handle such orders. Or, in a similar vein, a bot may find that a PO number is missing (the corresponding cell is empty), and hence it cannot proceed. Discovering conditions under which a routine cannot be deterministically continued (or started) is an open challenge in the field of RPM. Yet, this capability is a precondition to ensure that bots synthesized via RPM techniques can gracefully degrade and stop in order to hand off to human operators.

Finally, the vision of RPM exposed in this chapter, focuses on the problem of discovering automatable routines. Besides this problem, we envision that the field of RPM will encompass complementary problems and questions such as performance mining of RPA bots. This includes answering questions such as: "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?", and "Are there cases where the effects of a bot's actions are abnormal and warrant manual inspection?" In other words, over time, we envision that the scope of RPM will expand to cover the entire RPA lifecycle (cf. Figure 1), rather than being purely focused on the development of RPA bots.

Acknowledgments. Work supported by the European Research Council (PIX project) and by the Australian Research Council (DP180102839).

References

1. Wil M. P. van der Aalst, Martin Bichler, and Armin Heinzl. Robotic process automation. *BISE*, 60(4), 2018.
2. Ziawasch Abedjan, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. Dataxformer: A robust transformation discovery system. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 1134–1145. IEEE Computer Society, 2016.
3. Simone Agostinelli. Automated segmentation of user interface logs using trace alignment techniques (extended abstract). In Claudio Di Ciccio, Benoît Depaire, Jochen De Weerdt, Chiara Di Francescomarino, and Jorge Munoz-Gama, editors, *Proceedings of the ICPM Doctoral Consortium and Tool Demonstration Track 2020*, volume 2703 of *CEUR Workshop Proceedings*, pages 13–14. CEUR-WS.org, 2020.
4. 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*, volume 393 of *Lecture Notes in Business Information Processing*, pages 116–131. Springer, 2020.
5. Simone Agostinelli, Andrea Marrella, and Massimo Mecella. Automated segmentation of user interface logs. In Christian Czarnecki and Peter Fettke, editors, *Robotic Process Automation*. De Gruyter, 2021.
6. Daniel W. Barowy, Sumit Gulwani, Ted Hart, and Benjamin G. Zorn. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation 2015*, pages 218–228, 2015.
7. Dina Bayomie, Ahmed Awad, and Ehab Ezat. Correlating unlabeled events from cyclic business processes execution. In *Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 274–289. Springer, 2016.
8. Dina Bayomie, Claudio Di Ciccio, Marcello La Rosa, and Jan Mendling. A probabilistic approach to event-case correlation for process mining. In *Proceedings of the Int. Conference on Conceptual Modeling (ER2019)*, Lecture Notes in Computer Science. Springer, 2019.

9. Antonio Bosco, Adriano Augusto, Marlon Dumas, Marcello La Rosa, and Giancarlo Fortino. Discovering automatable routines from user interaction logs. In *Proceedings of the Business Process Management Forum (BPM Forum)*. Springer, 2019.
10. Massimiliano de Leoni, Marlon Dumas, and Luciano García-Bañuelos. Discovering branching conditions from business process execution logs. In *Proceedings of FASE*. Springer, 2013.
11. Himel Dev and Zhicheng Liu. Identifying frequent user tasks from application logs. In *Proceedings of IUI 2017*, pages 263–273. Springer, 2017.
12. A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. R. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI*. ACM, 2005.
13. José Gonzalez Enríquez, Andres Jimenez Ramirez, Francisco José Domínguez Mayo, and J. A. García-García. Robotic process automation: A scientific and industrial systematic mapping study. *IEEE Access*, 8:39113–39129, 2020.
14. D. R. Ferreira and D. Gillblad. Discovering process models from unlabelled event logs. In *BPM*. Springer, 2009.
15. Fabio Fumarola, Pasqua Fabiana Lanotte, Michelangelo Ceci, and Donato Malerba. Clofast: closed sequential pattern mining using sparse and vertical id-lists. *Knowledge and Information Systems*, 48(2):429–463, 2016.
16. Junxiong Gao, Sebastiaan J van Zelst, Xixi Lu, and Wil MP van der Aalst. Automated robotic process automation: A self-learning approach. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 95–112. Springer, 2019.
17. Jerome Geyer-Klingeberg, Janina Nakladal, Fabian Baldauf, and Fabian Veit. Process mining and robotic process automation: A perfect match. In *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018*, pages 124–131. CEUR-WS.org, 2018.
18. Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*, pages 317–330, 2011.
19. Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data mining and knowledge discovery*, 15(1):55–86, 2007.
20. Intellipaat. RPA Lifecycle. `https://intellipaat.com/blog/tutorial/rpa-tutorial/rpa-lifecycle/`. Accessed on 12 September 2021.
21. 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 *International Conference on Advanced Information Systems Engineering*, pages 446–461. Springer, 2019.
22. Z. Jin, M. R. Anderson, M. J. Cafarella, and H. V. Jagadish. Foofah: Transforming data by example. In *SIGMOD*. ACM, 2017.
23. Marc Kerremans and Tushar Srivastava. Discover the differences and use cases of process mining versus task mining. Research Note G00723821, Gartner, April 2020.
24. Mary Lacity and Leslie P. Willcocks. Robotic process automation at telefónica O2. *MIS Quarterly Executive*, 15(1), 2016.
25. Sau Dan Lee and Luc De Raedt. An efficient algorithm for mining string databases under constraints. In *International Workshop on Knowledge Discovery in Inductive Databases*, pages 108–129. Springer, 2004.

26. V. Leno, M. Dumas, M. La Rosa, F. M. Maggi, and A. Polyvyanyy. Automated discovery of data transformations for robotic process automation. *ArXiv*, abs/2001.01007, 2020.
27. 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 2020, Padua, Italy, October 4-9, 2020*, pages 153–160. IEEE, 2020.
28. Volodymyr Leno, Adriano Augusto, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Artem Polyvyanyy. Discovering data transfer routines from user interaction logs. *Inf. Syst.*, 107:101916, 2022.
29. Volodymyr Leno, Artem Polyvyanyy, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. Robotic process mining: Vision and challenges. *Bus. Inf. Syst. Eng.*, 63(3):301–314, 2021.
30. Volodymyr Leno, Artem Polyvyanyy, Marcello La Rosa, Marlon Dumas, and Fabrizio Maria Maggi. Action logger: Enabling process mining for robotic process automation. In *Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019*, volume 2420 of *CEUR Workshop Proceedings*, pages 124–128. CEUR-WS.org, 2019.
31. Henrik Leopold, Han van der Aa, and Hajo A. Reijers. Identifying candidate tasks for robotic process automation in textual process descriptions. In *Proceedings of BPMDS and EMMSAD*, pages 67–81. Springer, 2018.
32. Christian Linn, Phileas Zimmermann, and Dirk Werth. Desktop activity mining - A new level of detail in mining business processes. In *Workshops der INFORMATIK 2018 - Architekturen, Prozesse, Sicherheit und Nachhaltigkeit*, pages 245–258, 2018.
33. Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M. P. van der Aalst. Data-driven process discovery – revealing conditional infrequent behavior from event logs. In *Proceedings of the 29th International Conference on Advanced Information Systems Engineering*, pages 545–560. Springer, 2017.
34. Enno Ohlebusch and Timo Beller. Alphabet-independent algorithms for finding context-sensitive repeats in linear time. *Journal of Discrete Algorithms*, 34:23–36, 2015.
35. 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 *Proceedings of the 31st International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 446–461. Springer, 2019.
36. J. Shen, L. Li, and T. G. Dietterich. Real-time detection of task switches of desktop users. In *IJCAI*, 2007.
37. Myra Spiliopoulou, Bamshad Mobasher, Bettina Berendt, and Miki Nakagawa. A framework for the evaluation of session reconstruction heuristics in web-usage analysis. *Informs journal on computing*, 15(2):171–190, 2003.
38. Rehan Syed, Suriadi Suriadi, Michael Adams, Wasana Bandara, Sander J. J. Leemans, Chun Ouyang, Arthur H. M. ter Hofstede, Inge van de Weerd, Moe Thandar Wynn, and Hajo A. Reijers. Robotic process automation: Contemporary themes and challenges. *Comput. Ind.*, 115:103162, 2020.
39. Niek Tax, Natalia Sidorova, and Wil M. P. van der Aalst. Discovering more precise process models from event logs by filtering out chaotic activities. *J. Intell. Inf. Syst.*, 52(1):107–139, 2019.
40. C Tornbohm. Gartner market guide for robotic process automation software. Report G00319864, Gartner, 2017.

41. Sebastiaan J. van Zelst, Felix Mannhardt, Massimiliano de Leoni, and Agnes Koschmider. Event abstraction in process mining: literature review and taxonomy. *Granular Computing*, 6:719–736, 2021.
42. Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In *Proceedings of the 20th international conference on data engineering*, pages 79–90. IEEE, 2004.