DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
33

VOLODYMYR LENO

Robotic Process Mining: Accelerating the Adoption of Robotic Process Automation

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

33

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

33

VOLODYMYR LENO

Robotic Process Mining: Accelerating the Adoption of Robotic Process Automation

UNIVERSITY OF TARTU
Press

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in Computer Science on December 02, 2021 by the Council of the Institute of Computer Science, University of Tartu.

Supervisor

Prof.	Marlon Dumas
	University of Tartu, Estonia
Prof.	Fabrizio Maria Maggi
	Free university of Bozen-Bolzano, Italy
Prof.	Marcello La Rosa
	University of Melbourne, Australia
Prof.	Artem Polyvyanyy
	University of Melbourne, Australia

Opponents

Assoc. Prof.	Paolo Ceravolo
	University of Milan, Italy
Assoc. Prof.	Carmelo del Valle
	University of Seville, Spain

The public defense will take place on January 17, 2022 at 09:00, Online.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

ISSN 2613-5906
ISBN 978-9949-03-789-6 (print)
ISBN 978-9949-03-790-2 (PDF)

University of Tartu Press
`http://www.tyk.ee/`

ABSTRACT

Nowadays, a vast majority of business processes involve many tedious and repetitive tasks, for example transferring data across multiple systems or transforming data from one format to another. Automating these tasks can bring many benefits such as error-rate reduction, performance improvements, workflow standardization and transparency.

Automating repetitive tasks has been made more convenient by recent technological advances, particularly the emergence of Robotic Process Automation. Robotic Process Automation (or RPA for short) is a technology that allows organizations to automate routine digital tasks by executing software scripts that encode sequences of fine-grained interactions with Web and desktop applications.

Although RPA tools allow automating a wide range of routines, identifying and scoping routines that can be automated is time-consuming. Manual identification of candidate routines via interviews, walkthroughs, or job shadowing allows analysts to identify only the most visible routines. These methods are not suitable for identifying routines for automation in large-scale settings.

The thesis studies the problem of discovering routines that can be automated using RPA tools from logs of user interactions with IT systems. This problem is decomposed into a sequence of subproblems such as logs collection and preprocessing, identifying candidate routines, evaluating their automatability, and synthesizing executable specifications that implement them. The thesis provides an overview and analysis of the state-of-the-art for each identified subproblem and identifies gaps to be addressed.

Based on this analysis, the thesis makes four contributions. The first contribution is a format to store user interactions logs and a tool to record them. The analysis of existing solutions shows that they are not capable of recording user interactions logs that can be used to discover useful routines. Specifically, they record user interactions at a low level of granularity (keystrokes and clickstreams) and do not capture the data values used during such interactions. The format proposed in the thesis specifies what information is required to be captured by a logging tool to analyze, improve and automate the underlying user interactions. The devised recording tool records user interactions performed in Chrome and MS Excel applications. It records user interactions at a level of logical input elements of a target application. Moreover, it captures all the data values used during these interactions so that the corresponding routines can be analyzed with respect to their automatability and implemented in the form of executable scripts.

The second contribution is an end-to-end pipeline to automate routine tasks. Given a user interactions log, the proposed pipeline aims 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 are defined in a platform-independent language that

can be compiled into a script and executed in a target RPA tool.

The final two contributions instantiate the proposed pipeline. Specifically, the devised approaches focus on discovering routines for RPA. The first approach aims at discovering candidate routines for RPA from user interactions logs. It applies graph theory to split a user interaction log given as input into a set of segments, each representing an instance of task execution, and then identifies candidate routines by mining frequent sequential patterns from these segments. The second approach focuses on determining whether candidate routines are automatable and discovering their executable specifications. This approach exploits state-of-the-art data transformation discovery techniques and introduces several optimizations to improve their performance and discovery quality. The thesis also describes a method for detecting and removing semantically equivalent routines that lead to the same final effects.

Altogether, these approaches give rise to a new family of techniques, called Robotic Process Mining (RPM). RPM aims at helping RPA developers and analysts in the early stages of the RPA lifecycle. Specifically, it assists the analysts in drawing a systematic inventory of the tasks that can be automated with RPA and synthesizes executable specifications of such tasks that can be used as a starting point for their automation.

All the artifacts designed and developed for this thesis are publicly available as standalone open-source command-line applications. They are also combined into the pipeline in the form of a software-as-a-service tool called Robidium. The efficiency and potential usefulness of the proposed techniques have been evaluated using synthetic and real-life user interactions logs. The evaluation results show that these approaches can discover automatable routines that present in a user interaction log and identify automatable routines that users recognize as such in real-life logs.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Robotic Process Automation	RPA
Robotic Process Mining	RPM
User Interaction	UI
Control-flow Graph	CFG
Strongly Connected Component	SCC
Structured Query Language	SQL
Information System	IS
Automated Process Discovery	APD
Business Process Management	BPM
Business Process Model and Notation	BPMN
Business Process Management System	BPMS
Multi-perspective	MP
Information Technology	IT
JavaScript Object Notation	JSON
Application Programming Interface	API
Document Object Model	DOM
Routine Automatability Index	RAI
Levenshtein Edit Distance	LED
Coloured Petri Net	CPN

1. INTRODUCTION

1.1. Research Problem

Automation is an integral part of process improvement. Throughout history, manual work had been gradually handed over to machines that could replicate human work on a larger scale and a higher speed, significantly increasing efficiency. This allowed people to move from heavy and sometimes dangerous manual labor to more sophisticated activities.

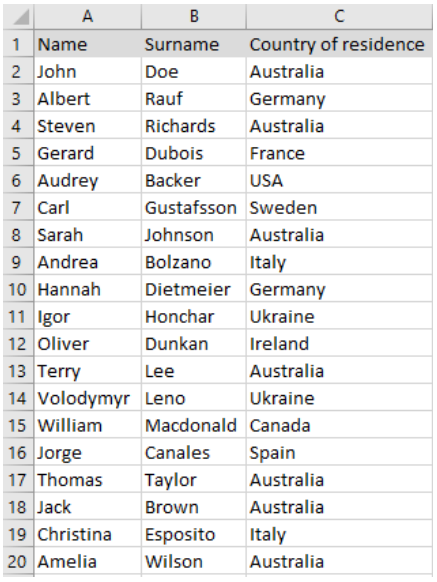
Nowadays, a vast number of processes are performed with the help of IT systems. For example, when a bank client applies for a loan, the employee enters all the necessary information into the bank system. This information is then analyzed, and the client is notified about the outcome. This process involves many tedious and routine activities, for example, filling in a web form of the banking system and checking the client's previous records. Such activities are time-consuming and do not bring much value. Nevertheless, they have to be performed daily.

Robotic Process Automation (RPA) [2] is an emerging technology that allows organizations to automate tedious and error-prone routine tasks in business processes by executing software scripts that encode sequences of fine-grained interactions with Web and desktop applications [3]. Using this technology, it is possible to automate data entry, data transfer, and verification tasks, particularly when they involve multiple applications.

Figure 1 shows an example of a task that can be automated via RPA. In this example, the spreadsheet with student records must be transferred row by row into a Web-based study information system. For each row in the spreadsheet, this task involves 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 the RPA tool's runtime environment, also known as *RPA software robot* (*RPA bot* or *bot* for short).

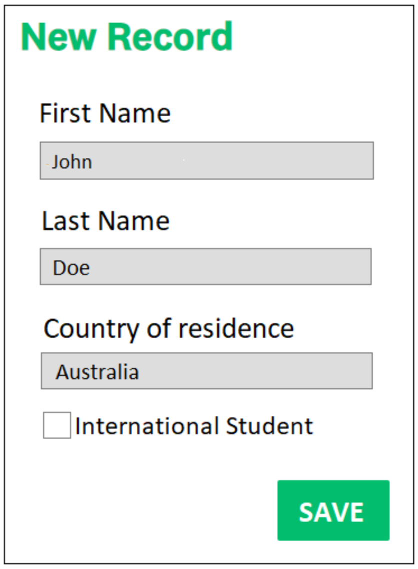
Several case studies have shown that RPA technology can improve efficiency and data quality in business processes involving clerical work [4, 5]. It leads to error-rate reduction and can free up workers from unrewarding activities to be reallocated to other more involving and stimulating tasks. RPA does not require substantial changes to the existing processes and IT infrastructure, operating instead on the user interface level by mimicking the human activities, e.g., by performing mouse clicks or keyboard inputs. This enables organizations to automate tasks (or chains thereof) with reasonably low costs and effort.

While RPA has already been successfully applied to various organizations [4, 6–10], to date, a great deal of time is still required to identify the routines for automation and manually program the RPA bots. Although RPA tools are able to automate a wide range of routines, they cannot determine which routines should be automated in the first place. The current practice for identifying candidate rou-



	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

(a) Student records spreadsheet



New Record

First Name: John

Last Name: Doe

Country of residence: Australia

☐ International Student

(b) New Record creation form

Figure 1: Extract of a spreadsheet with student data that needs to be transferred to a Web form

tines for RPA is through interviews, walk-throughs, and detailed observation of workers conducting their daily work, either in situ or using video-recordings [11]. 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 and are not cost-efficient in organizations where the number of routines is very high and where the routines are scattered around the process landscape. Besides, the information about routines obtained in such a way can often be incomplete or outdated. The negative consequences of mistakes introduced at this stage are magnified by the large number of bots typically deployed in an organization that adopts RPA. Hence, in practice, considerable time is invested in quality-testing the bots before deployment [2].

The goal of our research project is to minimize the amount of manual work required for automation by designing a family of approaches, namely Robotic Process Mining, to automatically identify routine tasks from the logs of user interactions with IT applications that can be automated via RPA, and discover executable specifications of such routines that the RPA bots can directly use.

In line with our goal, we formulate the following research questions:

- **RQ1. Given a user interaction log, how to identify the routines that can be potentially automated via an RPA tool?**
- **RQ2. Given a routine, how to discover its executable specification that can be executed via an RPA tool?**

We start with the analysis of the problem and design the architecture of the solution. For each component of the proposed architecture, we identify the research challenges, analyze how they are addressed by the existing research, and identify gaps. Accordingly, we implement a framework by devising a series of approaches that benefit from existing techniques while overcoming their limitations.

1.2. Requirements to solution

Routines for RPA automation can automatically be identified from the logs of users' interactions with information systems, or *UI logs*, for short, capturing the activities performed by one or more workers during the execution of their daily tasks. Table 1 shows a UI log that captures the execution of the task presented in Figure 1.

Table 1: Example of a user interaction log

	Timestamp	**UI Type**	...
1	2019-01-08 T 12:10:05	Copy Cell A2	...
2	2019-01-08 T 12:10:26	Paste in First Name	...
3	2019-01-08 T 12:10:51	Copy Cell B2	...
4	2019-01-08 T 12:11:04	Paste in Last Name	...
5	2019-01-08 T 12:11:36	Copy Cell C2	...
6	2019-01-08 T 12:11:49	Paste in Country of residence	...
7	2019-01-08 T 12:11:58	Click Button Save	...
...	...	...	...

Each row in this table corresponds to one user interaction (UI), e.g., copying a cell, editing a field, or clicking a button. A sequence of UIs frequently observed in a log represents the routine that can potentially be automated. The identification of such routines is not a trivial task.

While capturing multiple instances of workers' tasks, UI logs do not assign the recorded UIs to specific task instances (i.e., UI logs are unsegmented). Therefore, it is not clear where one instance of a task ends and another starts. A typical UI log is a large stream of UIs representing a long-running session of work. The identification of routines from such a log is computationally expensive. It becomes even more challenging due to the fact that UI logs often contain extraneous UIs that are not a part of the routine or that do not have any effect on its outcome. These UIs constitute *noise* and have to be ignored. 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 in the field with an incorrect value or copying a wrong cell in a spreadsheet), which need to be remedied afterwards.

UI logs may contain multiple routine variations if there are no restrictions on how the routine should be performed (e.g., in a fixed order of UIs). For example, if a routine involves editing a "First name" field and a "Last name" field, it generally does not matter in which order the corresponding edits are performed. Although being represented by different sequences of UIs, these variations may lead to the same effect(s) and, therefore, should be treated as one routine.

Another issue that needs to be addressed is that discovered routines can not be directly automated as they do not contain the information about the data used during their execution. Therefore, they are only considered to be candidates for automation. To automate a candidate routine, we have to discover its *executable specification*. An executable routine specification is a specification that contains all the information required to fully reproduce the effects of the routine with any inputs. In other words, given the same inputs observed in the log, it should be possible to use the executable routine specification to produce the same outputs/effects. An executable routine specification must specify all the UIs within the routine and how they are performed. It has to capture the required set of inputs and how these inputs are transformed into outputs (i.e., how data read during the routine execution is used to write data during the routine execution). In addition, an executable routine specification should include a specification of the conditions under which the routine should be activated. Finally, discovering the routine specification also involves the identification of non-automatable UIs, as not all UIs within a routine are necessarily automatable.

Accordingly, we define the following requirements for the desired solution:

- **REQ1.** The desired solution must discover routines that may be automated using the RPA technology from unsegmented logs.
- **REQ2.** The desired solution must be resilient to noise present in the logs.
- **REQ3.** The desired solution must be tolerant to routine variations.
- **REQ4.** The desired solution must correctly identify automatable and non-automatable UIs within a routine.
- **REQ5.** The desired solution must discover an executable specification for each automatable routine.
- **REQ6.** The desired solution must be scalable, i.e., have acceptable execution times on commodity hardware, when given as input a UI log corresponding to multiple hours of UI recordings.

1.3. Research Method

In this project, we follow the Design Science in Information Systems research method proposed by Hevner [12]. This method outlines the following seven guidelines to address a research problem.

- **GL1.** *Design as an artifact.* The result of the research project is a purposeful IT artifact that addresses an important problem.
- **GL2.** *Problem relevance.* The research project aims at acquiring knowledge and understanding that enable the development and implementation of technology-based solutions to a heretofore unsolved and important problem.

- **GL3.** *Design evaluation.* The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
- **GL4.** *Research rigor.* The construction and evaluation of the designed artifacts require the application of rigorous methods.
- **GL5.** *Research contributions.* The produced artifacts must provide a novel solution for the addressed research problem.
- **GL6.** *Design as a search process.* The produced artifacts are the result of an iterative search process aiming at finding the optimal solution.
- **GL7.** *Communication of research.* The research and its results must be presented to both academics and practitioners.

In this thesis, we implemented the above guidelines as follows. (GL2) The problem of discovering routines for automation via RPA is largely unexplored and only recently started to receive attention from academics. Although it resembles similar problems in other fields, for example, Process mining or Web usage mining, existing approaches only partially address the problem and fail to satisfy all the needs in the context of RPA. (GL5) This is the first attempt to design an approach to discover fully executable specifications of automatable routines.

(GL1) During the project, we designed and implemented multiple artifacts that solve the presented problem. We proposed a framework to discover executable specifications of automatable routines from user interactions logs. We designed a format for storing user interactions logs and implemented a logging tool capable of capturing the user interactions with Web and Excel applications. We also devised an approach to discover candidate routines from the recorded logs. Finally, we defined the requirements for a routine to be automatable and designed an approach that identifies automatable routines from a list of discovered candidates.

(GL4) In this thesis, we formally describe all the produced artifacts. We explain how they work by providing their pseudocode and by discussing it line-by-line. All our artifacts are supported by instructions on how to use them. (GL3) We evaluate our framework on a range of synthetic and real-life logs of varied characteristics (e.g., size, complexity) based on scalability and accuracy criteria. We explain in detail how the evaluation was performed so that it can be reproduced. (GL6) All the presented artifacts are the results of a long iterative design process, where we considered and tested multiple various approaches before selecting the optimal one. (GL7) We published six conference papers and three journal articles during our research project. The rank of the conferences and journals that published our research work ensures that it will reach both the research community and practitioners in the field of Business Process Management and Robotic Process Automation.

1.4. Contributions to the Research Area

This thesis presents four main contributions to the research area. The main idea of our research is that the routines for automation can be automatically discovered from the logs that capture user interactions with IT systems (e.g., selecting field or cells, copying and pasting, editing fields) during the execution of a worker's daily tasks. Therefore, our first contribution is the design of the format to store such interactions and the tool that automatically captures and records them. We analyzed the existing solutions to capture user interactions, identified their limitations and designed a list of requirements that our solution must satisfy. We identified the data that must be captured by the logging tool to automate user interactions and applied the developed logging tool in practice to record the user interactions logs that we used throughout our project.

The next major contribution is a "Robotic Process Mining" framework to identify automatable routine tasks from the logs of user interactions with Web and desktop applications and produce their executable specifications that can be used as a starting point for the automation effort. Given a user interactions log, the proposed framework aims 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 are defined in a platform-independent language that can be compiled into a script and executed in a target RPA tool.

The final two contributions instantiate the envisioned framework. Specifically, we devise and implement the artifacts needed to discover routines for RPA. The first artifact aims at discovering candidate routines for RPA from UI logs, while the second artifact focuses on the discovery of executable routines. The former artifact also implements a segmentation technique that extracts a set of task instances from a log. The latter artifact provides requirements for a routine to be automatable, assesses the level of routine's automatability, discovers the executable specification for a fully automatable routine, and filters out redundant specifications.

1.5. Thesis Structure

The rest of this thesis is organized as follows. Chapter 2 provides the necessary background to understand the scope of the problem addressed in the thesis. Chapter 3 presents the Robotic Process Mining framework. Chapter 4 reviews the state-of-the-art. Chapter 5 describes our approach to identify candidate routines from user interactions logs. Chapter 6 focuses on the discovery of automatable routines and their specifications. Chapter 7 presents the software that was produced during this thesis. Chapter 8 concludes the thesis by summarizing the contributions and outlining directions for future work.

2. BACKGROUND

This chapter provides all the prerequisites to understand the context and the scope of the thesis. First, in Section 2.1, we provide an overview of Business Process Management and how business processes can be automated in this field. Next, in Section 2.2, we focus on a specific automation technology, called Robotic Process Automation, which aims at automating the tasks within the processes via executable software scripts. Then, in Section 2.3 and Section 2.4, we present two families of approaches designed to extract valuable insights about the processes from the process execution data. Section 2.3 provides the basic concepts of Process Mining, while Section 2.4 introduces Task Mining.

2.1. Business Process Management

The increasing interest of business organizations in understanding and improving their processes gave rise to dedicated methods in process analysis, assessment, and refinement. Altogether, these methods can be framed under the *Business Process Management* discipline. Business Process Management, or BPM, in short, aims at optimizing business processes via their analysis and redesign. It allows business organizations to gain an operational advantage, reduce the costs and execution time of the processes, as well as related risks and errors [1].

BPM can be envisioned as a continuous cycle consisting of the following phases (see Figure 2):

- ***Process identification.*** This phase starts with the formulation of the business problem and the identification of the processes within the organization that are relevant to the problem. Their interrelations are analyzed, resulting in a process architecture used to select the process(es) for the subsequent phases of the cycle.
- ***Process discovery.*** This phase aims at discovering the workflow model of the selected process. Since the model corresponds to the current way the process is executed, it is called the *as-is* process model. For future analysis, the model is usually annotated with performance data (e.g., duration time of activities, waiting time between activities).
- ***Process analysis.*** During this phase, the constructed as-is process model is analyzed. During the analysis, all the issues associated with the model are identified, documented, quantified (if possible), and ranked with respect to their importance and potential impact.
- ***Process redesign.*** During this phase, the changes in the process required to address the documented issues are identified and evaluated. These changes are then applied to the as-is process model resulting in a new *to-be* process model that depicts how the process is expected to be executed.

- ◦ ***Process implementation.*** During this phase, the changes required to implement the to-be model are applied to the process. This may require structural changes (e.g., splitting, merging, or removing process activities, introducing parallelism) or process automation (e.g., developing and deploying IT systems).
- ◦ ***Process monitoring.*** During this phase, the newly implemented process is monitored and analyzed to verify whether it fully conforms to the intended execution. The presence of any deviations or errors may require another execution of the lifecycle.[1]

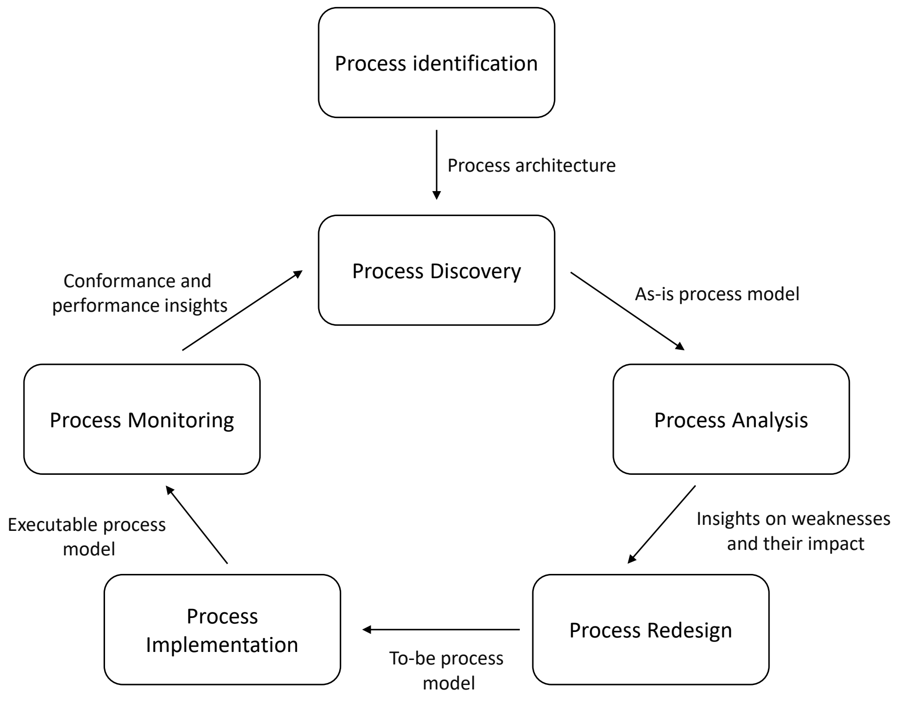


Figure 2: BPM lifecycle [1]

The process automation in BPM is achieved by developing and configuring IT systems that execute and coordinate the tasks within the process. Such systems aim at supporting the process participants (e.g., workers, or applications) in the execution of the process. This includes assigning tasks to process participants, providing them with the information required to perform the tasks, and monitoring the execution of the tasks [1].

A typical example of such a system is a Business Process Management System (BPMS). A BPMS is a system that supports design, analysis, execution, and monitoring of business processes based on their process models. The purpose of a BPMS is to coordinate an automated business process by assigning the process

[1]In some cases, the lifecycle is required to be repeated due to some external factors, e.g., the introduction of new policies, market situation changes, advancements in the technology.

tasks to responsible resources at the required points in time.

The process model constructed at the process redesign phase of the BPM lifecycle can be used as a basis for the BPMS to coordinate the underlying process execution. However, this model is not executable as it depicts the process at a conceptual level. The information required to execute such a model is missing as it is not relevant for process analysis. In addition, the model does not include the instructions on how to react in the case of unexpected issues and exceptions that may be observed during the execution of the process, capturing only the "happy path".

Accordingly, to be deployed into a BPMS, the conceptual process model must be extended and transformed into an executable model. This involves the identification of the automation boundaries, the detection of the automatable and manual tasks within the process, and the refinement of the process model by introducing exception handlings, by specifying the data objects required in each step of the process, by defining the decision-making logic within the process, by decomposing and aggregating some of the process tasks, and by specifying their execution properties (e.g., data types, participant assignment rules).

The process automation with BPMS improves the quality and the efficiency of the business processes. A BPMS is able to reduce the workload by automatically assigning the tasks to the process participants and gathering all the relevant information needed for their execution. It enables organizations to become more flexible in managing and updating their business processes and applications and allows for the creation of a unified IT infrastructure by integrating all the applications within the organizations. Another advantage of automation with BPMS is the execution transparency, which leads to improvements in organizational control, as the organizations get better knowledge about their processes and resources. Finally, a BPMS ensures that the processes are executed as designed. It assures that the processes are performed in the best way possible and guarantees their compliance with the established policies and norms.

The main disadvantage of this type of automation is that it usually demands massive changes in the organizations. The existing IT systems have to be entirely redesigned or reconfigured to be integrated with a BPMS, as they are usually not designed to be process-aware. Therefore, the costs and benefits of automating certain processes must be thoroughly analyzed before applying such heavyweight automation.

Alternatively, the processes can be automated via Robotic Process Automation (RPA). RPA works at the level of user interfaces and does not require a lot of integration effort, thus allowing automation at a relatively low cost. In the next section, we overview RPA and show how it is different from traditional business process automation. Then, we will talk about different RPA types, present the RPA lifecycle and discuss what tasks are the best candidates for automation via RPA.

2.2. Robotic Process Automation

Several partially overlapping definitions of RPA can be found in the research and industry literature. For example, [5] defines RPA as a category of software tools designed "to automate rules-based business processes that involve routine tasks, structured data, and deterministic outcomes". Meanwhile, [3] defines 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 [13] 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 in the RPA tool language for the software robot to follow, 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, driven by rules (e.g., if-then-else rules) and have "deterministic outcomes". Second, RPA tools can execute tasks that involve user interactions and other operations accessible via APIs (in any case, automated actions). And, third, RPA tools allow one 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* [13]. In attended use cases, the bot is triggered by a user. During its execution, an attended bot may provide and 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 script's execution. Attended bots are suitable for routines where dynamic inputs (i.e., inputs gathered during a routine execution) 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. It is important to detect such 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 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[2].

[2] In this thesis, we focus on discovering unattended RPA bots

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 BPMS, document workflow systems, and other types of workflow automation tools [1]. A key difference between RPA on the one hand and BPMSs, 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 user interface 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 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, e.g., a BPMS may trigger an RPA tool to perform a given step in a process.

Figure 3 shows the RPA lifecycle, which consists of the following four phases:

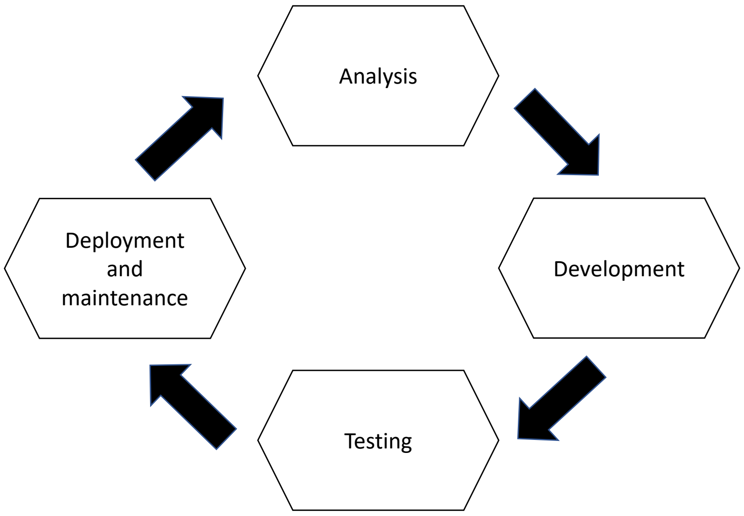


Figure 3: RPA lifecycle

- ◦ ***Analysis.*** During 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 en-

vironment 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[3] or Automation Anywhere[4] 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 large amount of manual, error-prone work, a code review and script evaluation are required.

- ***Testing.*** During 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 that identify and fix the errors that caused the failure.
- ***Deployment and maintenance.*** After successful testing, the bot can be deployed in the production environment and is ready to be used by the customers. However, in case any other issues arise, it is sent back to testing and development. Otherwise, it becomes available for the public. Starting from this point, continuous support and maintenance are provided to the customers if they find any bot defects.

The candidates for automation are selected based on a set of criteria. Good candidates for automation are characterized by a large number of executions and a high level of standardization. They should have a well-defined structure, follow rule-based logic, and should not require a lot of exception handling. The tasks to be automated should have low complexity and work with structured data available in a digital form. Additionally, the tasks are good candidates for automation if they are time-consuming and require much error-prone manual labor.

2.3. Process Mining

Process Mining [14] is a family of approaches that extract information about business processes from process execution data, a.k.a. event logs, recorded by standard enterprise systems available in organizations (e.g., CRM, ERP systems).

Each event in a log refers to an activity (i.e., a well-defined step in a business process) and is related to a particular case (i.e., a process instance). Events that belong to a case are ordered and constitute a single “run” of the process (often referred to as a trace of events). Event logs may store additional information about events such as resources (i.e., people and devices) executing or initiating the activities, timestamps indicating when the events occur, and data elements associated with the events. Data elements stored in the log can be either event attributes, i.e., data produced by the activities of a business process, or case attributes, namely

[3] `www.uipath.com`

[4] `www.automationanywhere.com`

data associated with the whole process instance. Table 2 shows an example of an event log.

Table 2: Example of an event log

Case ID	**Activity**	**Timestamp**	...
1	Create Fine	14/02/2021 14:00:10	...
2	Create Fine	14/02/2021 15:00:26	...
1	Send Bill	14/02/2021 15:05:31	...
2	Send Bill	14/02/2021 15:07:45	...
1	Process Payment	15/02/2021 14:30:15	...
1	Close Case	15/02/2021 14:32:24	...
2	Send Reminder	18/02/2021 18:56:00	...
2	Send Reminder	20/02/2021 19:01:04	...
2	Process Payment	21/02/2021 14:32:00	...
2	Close Case	21/02/2021 14:35:53	...
...	...	...	...

There are similarities between UI logs and event logs used in process mining. Specifically, both types of logs consists 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. Some commercial process mining vendors have exploited the similarities between UI logs and business process event logs to offer RPA-related features.

Depending on the input data type, we distinguish two classes of process mining techniques: *tactical* and *operational*. Tactical techniques work with historical data; their goal is to extract the information about process execution, such as how the process is performed or how much time it takes to complete the process [1, Chapter 11]. In contrast, operational techniques are applied to real-time data, and their goal is to provide insights into ongoing processes. A typical example of such insights is the information about the remaining execution time of the process [15] and its outcome [16]. Below, we describe the tactical process mining, as the problem addressed in the thesis is tactical in nature.

In general, tactical process mining techniques can be classified into four categories:

- ***Process discovery.*** Given an event log as input, process discovery aims is to construct a model (e.g., a flowchart) that depicts the behavior captured in the log.
- ***Conformance checking.*** Conformance checking aims at comparing existing process models against event logs that capture the same process, verify whether the model conforms to the real execution of the process, identify mismatches, and highlight the differences in behavior.
- ***Performance mining.*** Performance mining analyzes the performance efficiency of the processes using the information from the event logs. The output of performance mining are performance statistics (e.g., throughput

time, average activity execution time, waiting time), that can be used for process improvement, e.g., to identify the bottlenecks within the process.

◦ ***Variant analysis.*** The goal of variant analysis is to compare the executions of multiple variants of the same business process to identify the differences and what may cause them.

Discovering RPA routines is closely related to process discovery. Process discovery allows one to analyze the processes by extracting insights about their execution in the form of a model that describes how the process is conducted. Such a model captures the activities performed during the process and their control-flow relationships. A process model can be discovered in different formats, e.g., BPMN, EPC, YAWL models, Petri nets.

The simplest representation of a process is a process map. A process map is a directed weighted graph, where each vertex represents an activity of the process, and each edge corresponds to a directly-follows relation between two activities. Every edge and vertex is annotated with its frequency (number of times the activity/relation has been observed in the log). Figure 4 shows an example of a process map constructed from the event log presented in Table 2.

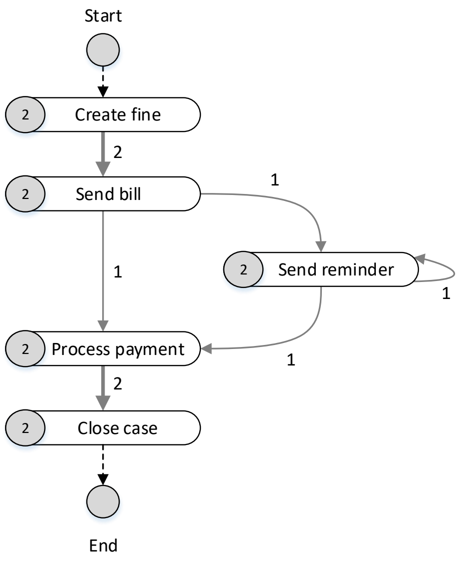


Figure 4: Example of a process map

Process maps capture only the sequential order of the activities within the process and do not identify more complex relations, for example, parallelism or exclusive choice. They are used as a basis by many process discovery techniques to construct more advanced models, e.g., BPMN models.

The quality of the discovered model can be assessed with respect to four criteria [14]:

◦ ***Fitness.*** The discovered model should describe the behavior observed in the event log, i.e., it should be able to replay each trace in the log.

- ◦ ***Precision.*** The discovered model should not allow the behavior that is not present in the event log.
- ◦ ***Generalization.*** The discovered model should generalize the behavior observed in the event log, i.e., it should support traces that are likely to be generated by the underlying process even if they are not present in the log.
- ◦ ***Simplicity.*** The discovered model should be as simple as possible.

As can be seen clearly, some of these criteria are contradicting. For example, to achieve high precision, one has to sacrifice generalization, and vice versa. Balancing these quality metrics is the main challenge of process discovery.

The problem of automated process discovery (APD) has been studied intensively in the past two decades, and a wide range of APD techniques has been proposed [17]. However, most of these techniques focus on discovering the control-flow models that represent a high-level abstraction of the underlying processes.

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 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.

Third, a record in an event log often does not contain all input or output data used or produced during the corresponding task's execution. 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., the address). Meanwhile, 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 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 and 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 always recorded when the patient arrives, but it is only 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 record it. Meanwhile, a UI log is explicitly recorded for analysis purposes, which allows all relevant events to be collected subject to the UI recording tool's capabilities.

2.4. Task Mining

Every business process can be represented as a sequence of tasks (e.g., process payment, register application) performed within an organization to achieve a business goal. At a lower level, each task is composed of steps that specify how to execute the task in detail. For example, registering the application may require the worker to create an instance of a new form in the Web-based information system, fill in its fields (e.g., name, address, date of birth), and click a button to save it in the system. Figure 5 shows an example of the fines handling process and the steps required to complete the first task in the process, namely *Create Fine*.

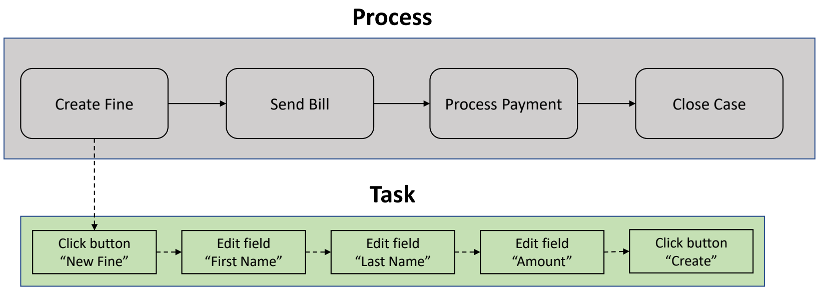


Figure 5: Process vs. task

Task mining [18] is a technology that extracts valuable insights about the tasks within a process from low-level event data available in UI logs. In contrast to process mining which is focused on end-to-end processes, task mining works at a lower level of abstraction. In particular, it concentrates on the discovery and analysis of the tasks within the processes. While process mining aims at discovering in what order the tasks are performed, task mining also discovers how they are performed, i.e., what actions have to be executed to complete the tasks. In addition to discovering the control-flow model of the underlying tasks, it also identifies what data is used by each of the steps of the task and how it is transformed.

Depending on the goal of analysis, we can distinguish three main use cases of task mining:

- ***Task optimization.*** In this use case, the main goal is to discover how a task is performed, to detect inefficiencies and deviations with respect to work policies and instructions, and to identify the opportunities for its improvement.
- ***Resource optimization.*** In this use case, the main goal is to analyze the resource productivity, i.e., what applications are frequently used, how pro-

ductive the workers are, and how much time do they spend on different applications. This can be used to identify and improve the applications that workers struggle to use efficiently.

- ***Task automation.*** In this use case, the main goal is to identify opportunities for task-level automation, i.e. to identify what tasks can be potentially automated and how much value could this automation bring to the company. The automation of a task can be achieved using a variety of technologies. For example, one could develop a middleware software to connect the applications involved in a task via their APIs to replace a manual flow with an automated (programmatic) flow. An alternative approach is to implement an RPA bot that would automate a task by replicating user actions on the target applications. Task automation, and synthesis of automatable specifications that can be used for RPA developments, in particular, are the main focus of the thesis.

Both, task mining and process mining belong to a wider family of process analytics tools (see Figure 6). They analyze business processes from different perspectives, thus, complementing each other. The combination of task mining and process mining enables a deeper analysis of the business processes of an organization and allows organizations to obtain a full picture of their processes. Given that task mining and process mining operate on different levels of abstraction, they also take different inputs. While process mining works with event logs that capture high-level information about the process execution, task mining operates over UI logs consisting of fine-grained interactions with desktop applications.

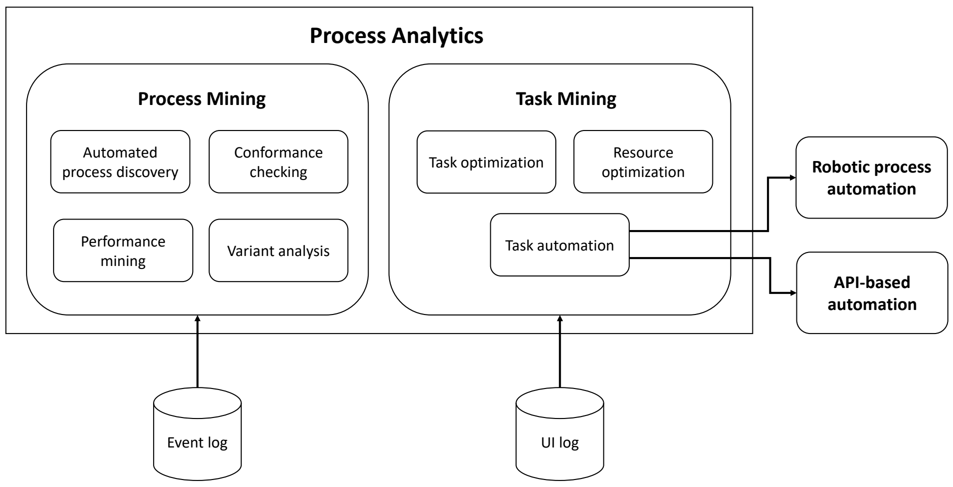


Figure 6: Process analytics overview

3. ROBOTIC PROCESS MINING: AREA, DEFINITION, ARCHITECTURE

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?* [19] We envision a new class of tools, namely Robotic Process Mining (RPM) tools, that address this question. Specifically, 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 the automation. Thus, RPM focuses on the first two phases of the RPA lifecycle: analysis and development.

In line with the above definition, we distinguish three main phases in RPM: (1) collecting and pre-processing UI logs corresponding to the executions of one or more tasks; (2) identifying candidate routines for RPA; and (3) discovering executable RPA routines.[1] In this chapter[2], we analyze the concepts involved across these three phases (Section 3.1), and we refine these phases into a tool pipeline (Section 3.2).

3.1. Concepts

The main input for RPM is a *UI log*, which has to be recorded beforehand. A 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 a UI log, which we use herein as a running example, is given in Table 3.

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), sufficient enough to reconstruct the performed activity. The payload of a UI is not standardized and depends on the UI type and application. Consequently,

[1]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 [20].

[2]Corresponding to [21]

Table 3: Fragment of a user interaction log

	UI	UI	Payload					
Row	**Timestamp**	**Type**	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 in google	–	–	–
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-page/	219	New Tab	–	–	–
7	2019-03-03T19:03:56	Select Tab (Web)	https://chrome/new-tab-page/	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 creation	–	–	–
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	–	–
…	…	…	…	…	…	…		

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., cell 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). Table 4 shows UIs and their associated payloads recorded by the Action Logger tool [22]. The UIs are logically grouped, based on their type, into three groups: navigation, read, and write UIs. Every UI in a log is an instantiation of one of the UI types from Table 4, with every parameter assigned with a specific value.

Table 4: User interaction types and their parameters

UI	**UI**	**Parameter Names**					
Group	**Type**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
	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			
Navigation	Add Worksheet (Excel)	Workbook name	Worksheet name				
	Select Worksheet (Excel)	Workbook name	Worksheet name				
	Select Cell (Excel)	Workbook name	Worksheet name	Cell column	Cell row	Value	
	Select Range (Excel)	Workbook name	Worksheet name	Range columns	Range rows	Value	
	Select Field (Web)	URL	Name	ID	Type	Value	
	Copy (Web)	URL	Name	ID	Value	Copied content	
Read	Copy Cell (Excel)	Workbook name	Worksheet name	Cell column	Cell row	Value	Copied content
	Copy Range (Excel)	Workbook name	Worksheet name	Range columns	Range rows	Value	Copied content
	Paste Into Cell (Excel)	Workbook name	Worksheet name	Cell column	Cell row	Value	Pasted content
	Paste Into Range (Excel)	Workbook name	Worksheet name	Range columns	Range rows	Value	Pasted content
	Paste (Web)	URL	Name	ID	Value	Pasted content	
	Click Button (Web)	URL	Name	ID	Type		
Write	Click Link (Web)	URL	Inner text	Href			
	Edit Field (Web)	URL	Name	ID	Type	Value	
	Edit Cell (Excel)	Workbook name	Worksheet name	Cell column	Cell row	Value	
	Edit Range (Excel)	Workbook name	Worksheet name	Range columns	Range rows	Value	

To obtain a UI log, 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 the 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 identify routines amenable for automation. For each such routine, RPM then aims at extracting an executable specification (herein called a *routine specification*). 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, Figure 7 presents a class diagram capturing the above concepts and their relations.

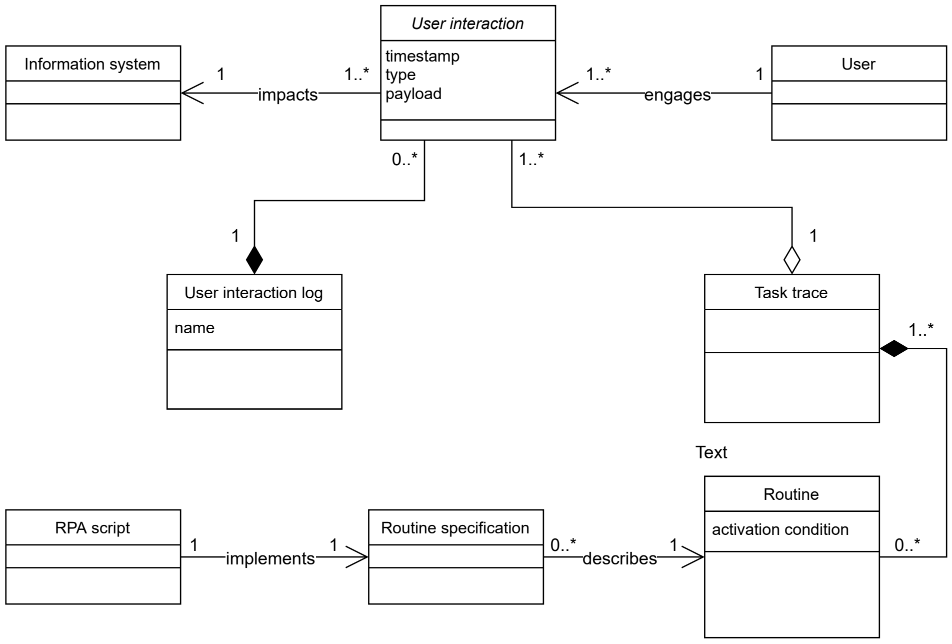


Figure 7: Class diagram of RPM concepts

3.2. Architecture

As mentioned earlier, the three main phases of RPM are (1) UI log collection and preprocessing, (2) candidate routine identification, and (3) executable routine discovery. 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 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 Figure 8. Below we discuss each step of this pipeline.

Recording. The recording of a UI log involves capturing low-level UIs, such as selecting a field in a form, editing a field, opening a desktop application, or

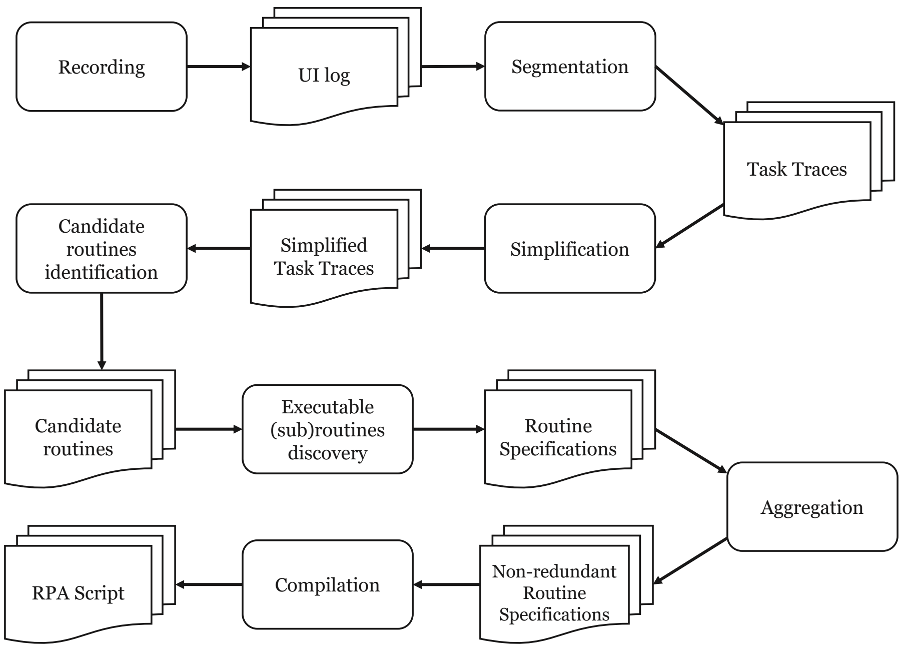


Figure 8: RPM pipeline

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, such as JitBit Macro Recorder[3], TinyTask[4], and WinParrot[5], 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. RPA tools (e.g., UiPath

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

[4]`https://www.tinytask.net/`

[5]`http://www.winparrot.com/`

Enterprise RPA Platform[6], and Automation Anywhere Enterprise RPA[7]) 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. To this end, a proposal has been made [23]. However, at the moment, the paper only presents a conceptual design of a tool.

Segmentation. In its raw form, a 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, we, therefore, need to segment a UI log into 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 3).

Simplification. 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 further steps of the pipeline (e.g., the discovery of the candidate routines). Accordingly, the next step is *simplification*, which aims at noise identification and removal. The UIs in the log

[6]`https://www.uipath.com/`

[7]`https://www.automationanywhere.com/`

are removed so that the resulting log captures the same effects as the original one while being simpler (i.e., having fewer UIs). We also remove the UIs that are not important for the automation (e.g., selecting a field or a cell).

A vast majority of noise UIs can be discovered even before the segmentation. Removal of noise at this stage can help in identifying task traces. However, some noise can be identified only after the segmentation, when the information about UIs in task traces is available. A typical example of such noise is a UI that corresponds to overwriting the value of a field. In the unsegmented log, removing the previous UI of type edit may be an error if the second UI of type edit belongs to a successive task execution. Hence, we apply simplification two times: before and after segmentation, where we remove different types of noise.

Candidate routines identification. 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 UIs that occur across multiple task traces, a.k.a. routines, and identify which such routines are amenable for automation. This step's output is a set of automatable or semi-automatable routines, ranked according to their automation potential (e.g., based on their execution frequency and length)[8].

Lacity and Willcocks [4] 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 allow for the automatic identification of automatable routines. In a recent systematic review of the RPA literature, Syed et al. [19] conclude that "there is a need for formal, systematic and evidence-based techniques to determine the suitability of tasks for RPA.". In particular, 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 candidates for RPA from large volumes of UI logs. In this thesis, we use the notion of *determinism* to assess the routine's amenability for automation. A routine can be automated if every UI belonging to the routine is deterministically activated and uses the data produced from the previous UIs (e.g., manual input into a text field is an example of a non-deterministic UI). We say that a UI is deterministically activated if we know when to execute it (e.g., tick the box *International* if the student's country of residence is not Australia).

Considering the running example provided in Table 3 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

[8]We also observe that not every routine is worth being automated. The automation of one routine can bring much more benefits than the automation of another. Thus, the cost-benefit analysis of routine automation is an important task [4]. However, this is a separate topic and is out of the scope of this thesis.

spreadsheet. Since she did not read the data values explicitly (e.g., by copying the values into 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).

Executable routines discovery. After the candidate routines for automation are identified, the next step is the *executable (sub)routines discovery*. For each candidate routine, this step identifies the *activation condition* (UIs 2 and 19 in Table 3), 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 it can be obtained. The discovery of the 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.

Aggregation. The result of the *executable (sub)routine discovery* step is a set of executable routine specifications for all the automatable candidate routines. 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. Figure 9 shows an example of a specification that can be discovered from the running example shown in Table 3 (assuming that the full-length UI log contains traces capturing a fully automatable routine that handles domestic students). The output of the aggregation step is a set of non-redundant routine specifications.

Compilation. The *executable (sub)routine discovery* and the *aggregation* steps lead 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 UIs from the routine specification into commands in the target RPA tool's scripting language. This step requires the correct identification of the application elements involved during the routine execution (e.g., button or text field on the Web form). For example, when converting a UI of clicking a button on a Web page into an executable command, we need to identify the HTML element representing this button and extract its

DOM position. Such information can be captured by a logging tool during the *recording* step[9].

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.

[9]The information about the exact location of an element involved in a UI sometimes may be missing or not available at all (e.g., when working with custom applications). In such a case, 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 UIs recorded and capture all the data involved during their execution.

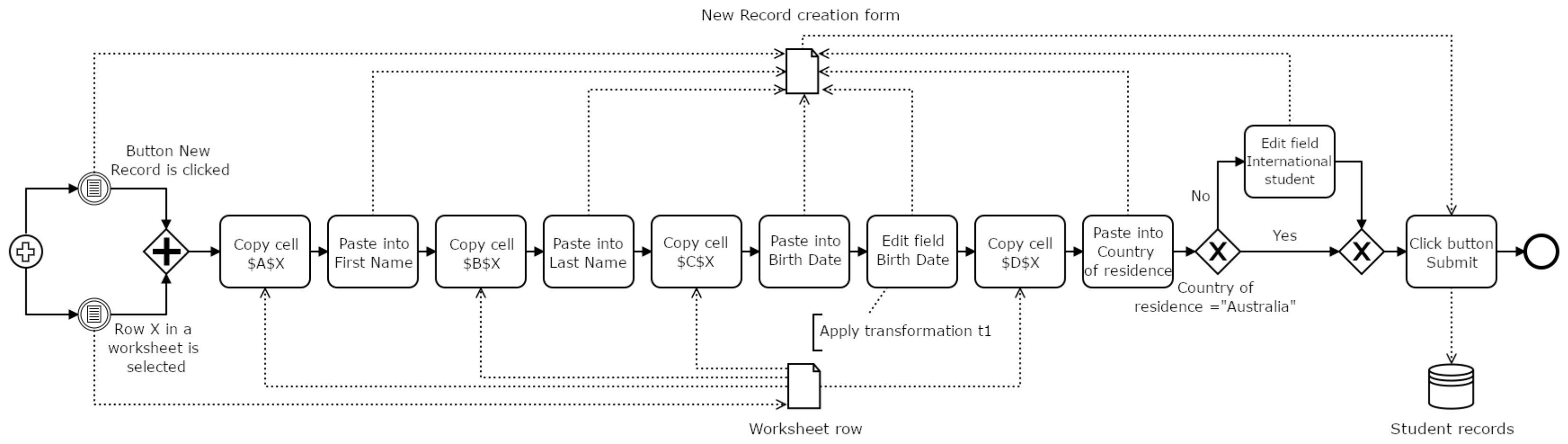


Figure 9: Example of a routine specification presented in BPMN format

3.3. Summary

This chapter presented Robotic Process Mining, a new family of techniques to automate routine tasks within business organizations' processes. We aggregated these techniques into a pipeline to analyze logs of fine-grained user interactions with IT systems to identify routines amenable for automation using RPA tools, and synthesize executable specifications of such routines that RPA developers can use as a basis for implementing further automation. In the following chapters, we will review the state-of-the-art techniques that can be used to perform the steps in this pipeline and present our own approaches to implement them that address the limitations of the existing works.

4. STATE OF THE ART

In the previous chapter, we presented RPM and the pipeline to discover executable specifications of routines for RPA from UI logs. The underlying pipeline can be divided into three main phases: 1) recording and preprocessing of the UI logs; 2) discovery of candidate routines for automation; 3) discovery of executable routines. In this chapter, we overview the works focused on realizing these phases, and identify their limitations. Section 4.1 focuses on the segmentation of the UI logs, which is the core step of preprocessing. Section 4.2 presents state-of-the-art on discovering candidate routines for automation. Finally, Section 4.3 reviews the works on discovering executable routines, and Section 4.4 concludes the chapter.

4.1. Segmentation

Given a UI log (i.e., a sequence of UIs), the main goal of segmentation is to break it into non-overlapping subsequences of UIs, namely *segments*, where each segment represents the execution of a task performed by an employee from start to end. In other words, the purpose of segmentation is to identify the boundaries of tasks captured in a log, i.e., to find the points in a log where one task execution ends and another starts.

The problem of segmentation is similar to the problem of web session reconstruction [24], whose 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) [24]. Methods for session reconstruction are usually based on heuristics that rely on the structural organization of web sites 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.

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 [25, 26]. 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.

Segmentation can also be seen as a problem of discovering periodic patterns in time series. While several studies addressed the latter problem over the past decades [27, 28], most of them require information regarding the length of the pattern to discover or assume a natural period to be available (e.g., hour, day, week). This makes the adaptation of such techniques to solve the problem of segmentation challenging unless periodicity and pattern length are known a priori.

Segmentation also relates to the problem of correlating uncorrelated event logs in process mining. In such logs, each event should normally include an identifier of a process instance (case identifier), a timestamp, an activity label, and possibly other attributes. When the events in an event log do not contain explicit case identifiers, this log is said to be uncorrelated. Various methods have been proposed to extract correlated event logs from uncorrelated ones [29–31]. However, existing methods in this field address the problem in restrictive settings. Ferreira et al. [29] assume that the underlying process is acyclic, whereas Bayomie et al. [30, 31] assume that a process model is given as input, which means that the model of the routine is known. Both these assumptions are unrealistic in our setting: a process model is not available since we are precisely trying to identify the routines in the log, and a routine may contain repetitions. Also, these approaches were shown to produce rather inaccurate results. In contrast, RPM seeks to identify routines with high confidence levels, given that replicating a routine inaccurately can lead to costly errors, especially in unattended bot contexts.[1]

The segmentation problem resembles the event abstraction problem in Process Mining [32], where low-level events are grouped into high-level business activities. However, most of the existing techniques that address this problem are supervised, i.e., they require a reference model [33, 34], mapping examples [35] or time intervals [36]. This information is not usually available in advance. Although these techniques can identify semantic groups of events, they are not able to discover entire tasks consisting of events performed in various orders often interspersed with noise.

In some scenarios, segmentation may be accomplished by combining transactional data recorded by enterprise information systems and user interactions logs, as proposed in [37]. However, the problem of this approach is that such transactional data often provides only a limited information about the process context, which is not enough to identify the boundaries of tasks captured in the user interactions logs.

In [38], an approach for real-time task recognition based on a sequence of events using supervised machine learning is proposed. While this approach is capable of identifying the switches between tasks, it does not discover the actual task boundaries, and, hence, it is unable to achieve the main goal of segmentation.

Recent work on UI log segmentation [39] 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.

4.2. Candidate routines identification

Dev and Liu [40] have noted that the problem of 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 [41]. Indeed, routine identification aims at

[1]In the case of attended RPA, it is not necessary to discover routines with 100% confidence given that a user verifies the outcome of the routine and may correct errors.

identifying repetitive (frequent) sequences of interactions, which can be represented as symbols. 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 [42,43] (hence vulnerable to noise), and those that allow frequent patterns to have gaps within the sequence of symbols [44,45] (hence noise-resilient).

Depending on their input, we can distinguish between algorithms that operate on a collection of sequences of symbols and those that discover frequent patterns from a single long sequence of symbols [43]. The former algorithms can be applied to segmented UI logs, while the latter can be applied directly to unsegmented ones although they only scale up when identifying exact patterns. While such approaches discover the frequently repeated routines, they do not analyze whether they are automatable. In other words, these approaches focus on the discovery of the control-flow models and not on the discovery of executable specifications.

The identification of frequent routines from sequences of actions is related to the problem of Automated Process Discovery (APD) [17], which has been studied in the field of process mining. Recent works [20,46,47] show that RPA can benefit from process mining. In particular, [46] and [47] propose to apply traditional APD techniques to discover process models of routines captured in UI logs. However, traditional APD approaches can not be used to discover routines that can be automated via RPA for several reasons.

First, they focus on discovering models that capture the behavior observed in a log from a control-flow perspective. They do not consider the data 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. A subset of APD approaches targets the problem of discovering process models with data-driven branching conditions [48] as well as control-flow relations that only hold under certain conditions [49]. These approaches can be used as a starting point for developing techniques for discovering RPA routines. Indeed, to discover RPA routines, we need to discover the activation conditions that trigger a routine and possibly other conditions within the routine.

Second, APD techniques generalize, i.e., they produce models with traces that have not been observed. However, in the context of RPA, we seek to discover only routines that have been previously observed (many times) in order to reproduce these routines. Automating a routine that has never been observed is risky because the probability of letting a bot do something that should not be done is high. If these errors go undetected, they can later lead to costly mistakes and time-consuming corrective actions. APD approaches are also approximate, meaning that they can produce models that do not fit 100% with the log. Instead, routines for RPA must be precise, as the lack of precision can lead to potential errors when executing a bot.

Also, process discovery techniques operate over event logs consisting of coarsegrained events, such as events indicating the start and completion of a task, as opposed to low-level actions such as clicks and keystrokes. Commercial process mining vendors address this granularity difference by providing features to discover two-level process models. For example, the Minit[2] process mining tool provides a multi-level process discovery feature to support RPM tasks. Specifically, given an event log recording task executions and a UI log, Minit can 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 with relatively simple internal structures and could, therefore, be potentially automated. However, it cannot determine if a task contains fully automatable (sub-)routines. Also, the tool assumes that there is a clear relationship between the events in the UI log and those in the high-level log.

Another commercial tool, namely Kryon Process Discovery,[3] 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, not all routines identified as candidates for automation by this tool can be automated. Suppose the data values that are entered in a particular step cannot be determined from the previously observed values. In that case, 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.

Another technique for routine identification [50] attempts to identify tasks amenable for automation from textual documents. 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 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. This approach is suitable for earlier stages of routine identification and could be used to determine which processes or tasks could be recorded and analyzed to identify routines.

Recent work [51] proposes to leverage data distribution in recorded logs to identify automation opportunities. Specifically, it shows how the Pareto principle can be used to select routines that maximize the expected benefit.

The discovery of data transfer routines that are amenable to RPA automation has been addressed in [52]. This paper proposes a technique to discover sequences of actions such that the inputs 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

[2]`https://www.minit.io/`

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

to noise and variability in the order of the actions.

[53] presents a vision for Intelligent Process Automation, a new class of tools that combine machine learning and artificial intelligence to identify automation opportunities and achieve more complex automation than the one provided by RPA. Specifically, it focuses on the automation of complex tasks involving decision making, coordination and collaboration of multiple RPA solutions. The paper reports artificial intelligence challenges that have to be overcome to realize this vision.

4.3. Executable routines discovery

The discovery of executable routines 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 [54]. Similarly, the authors in [55] 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.

A subset of process discovery approaches focuses on discovering simulation models [56] that can be given as input to business process simulators, which execute them in a stochastic sense. However, 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 automatically discovers the data transformations (i.e., the mappings between inputs and outputs) in the discovered processed models. Nevertheless, these data transformations are essential for process models that have to be executed by a process execution engine or by an RPA tool.

Recent work [52] suggests that this step in the discovery of executable routines can be implemented using existing methods for automated discovery of data transformations "by example" [57, 58]. However, these methods suffer from scalability issues. The approach presented in [59] to extract rules from segmented UI logs that can automatically fill in forms. However, this approach only discovers branching conditions that specify whether a certain 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. In [60] and [61], the authors present an approach to automatically discover routines from segmented UI logs and automate them in the form of 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.

Year	Paper title	Ref	Segmentation	Candidates discovery	Automatable routines discovery
2003	A framework for the evaluation of session reconstruction heuristics in web usage analysis	[24]	✓		
2005	Tasktracer: a desktop environment to support multi-tasking knowledge workers	[26]	✓		
2007	Discovery of periodic patterns in spatiotemporal sequences	[27]	✓		
2007	Real-time detection of task switches of desktop users	[25]	✓		
2009	Discovering process models from unlabelled event logs	[29]	✓		
2011	Automating string processing in spreadsheets using input-output examples	[54]			✓
2015	Flashrelate: extracting relational data from semi-structured	[55]			✓
2016	Correlating unlabeled events from cyclic business processes execution	[31]	✓		
2016	The use of process mining in business process simulation model construction - structuring the field	[56]			✓
2017	Time series chains: A new primitive for time series data mining	[28]	✓		
2017	Identifying frequent user tasks from application logs	[40]		✓	
2018	Process mining and robotic process automation: A perfect match	[20]		✓	
2018	Efficiently interpreting traces of low level events in business process logs	[33]	✓		
2018	Guided process discovery - a pattern-based approach	[34]	✓		
2018	Desktop activity mining - a new level of detail in mining business processes	[37]	✓	✓	
2018	Identifying candidate tasks for robotic process automation in textual process descriptions	[50]	✓		
2019	A probabilistic approach to event-case correlation for process mining	[30]	✓		
2019	Machine learning-based framework for log-lifting in business process mining applications	[35]	✓		
2019	A method to improve the early stages of the robotic process automation lifecycle	[46]	✓	✓	
2019	Discovering automatable routines from user interaction logs	[52]			✓
2019	Automated robotic process automation: A self-learning approach	[59]		✓	✓
2020	Event-log abstraction using batch session identification and clustering	[36]	✓		
2020	Iot-based activity recognition for process assistance in human-robot disaster response	[38]	✓		
2020	Automated segmentation of user interface logs using trace alignment techniques	[39]	✓		
2020	On the Pareto principle in process mining, task mining, and robotic process automation	[51]		✓	
2020	From robotic process automation to intelligent process automation	[53]		✓	
2020	Automated generation of executable RPA scripts from user interface log	[60]		✓	✓
2021	Event abstraction in process mining: literature review and taxonomy	[32]	✓		
2021	Candidate digital tasks selection methodology for automation with robotic process automation	[47]		✓	
2021	SmartRPA: A tool to reactively synthesize software robots from user interface logs	[61]		✓	✓

Table 5: Summary of prior work related to the three phases of RPM

4.4. Summary

This chapter gave an overview of the state-of-the-art techniques that aim at addressing the problems envisioned in the RPM pipeline. Specifically, it reported techniques focused on the three main phases of the pipeline: segmentation, discovering candidate routines for automation, and discovering executable routines. The overview of the related literature is provided in Table 5. The review highlights that existing approaches do not solve the problem posed in the thesis. Existing approaches for routine identification assume that the input UI log is already segmented, while in practice, all UI logs originally are represented by a single sequence of UIs. Moreover, they have many limitations, e.g., focusing on the control-flow and ignoring the data-flow of routines, discovering imprecise routines, and intolerance to noise. Also, currently available approaches for segmentation are too restrictive. They make assumptions that are not realistic in the RPM context. Finally, although there are some approaches to discover executable routines, they only allow for partial automation and limited usage can not discover executable specifications that can be used for RPA. Therefore, discovering routines that can be fully automated using RPA and generating the corresponding executable specifications (scripts) is still an open challenge.

5. DISCOVERY OF CANDIDATE ROUTINES

In this chapter[1], we address **RQ1**: *"Given a user interaction log, how to identify the routines that can be potentially automated via an RPA tool?"*. The review of approaches for discovering candidate routines for automation (see Chapter 4) highlights that there is no efficient technique to identify candidate routines from unsegmented UI logs. Moreover, these techniques focus only on the control-flow perspective and are not robust to noise. Therefore, there is the need to develop a new approach to identify candidate routines for automation. In this chapter, we present a noise-resilient approach to discover candidate routines from unsegmented UI logs. This chapter is structured as follows. Section 5.1 describes the approach, Section 5.2 reports its empirical evaluation and Section 5.3 summarizes the chapter.

5.1. Approach

In this section, we describe our approach for identifying candidate routines in UI logs. As input, the approach takes a UI log and outputs a set of candidate routines. The approach follows the RPM pipeline demonstrated in Chapter 3. It consists of a preprocessing/normalization phase followed by two macro steps in which the UI log is decomposed into segments and, then, candidate routines are identified by mining frequent sequential patterns from the segments. The approach is summarized in Figure 10. Next, we describe the steps in detail, including the required UI log preprocessing and normalization.

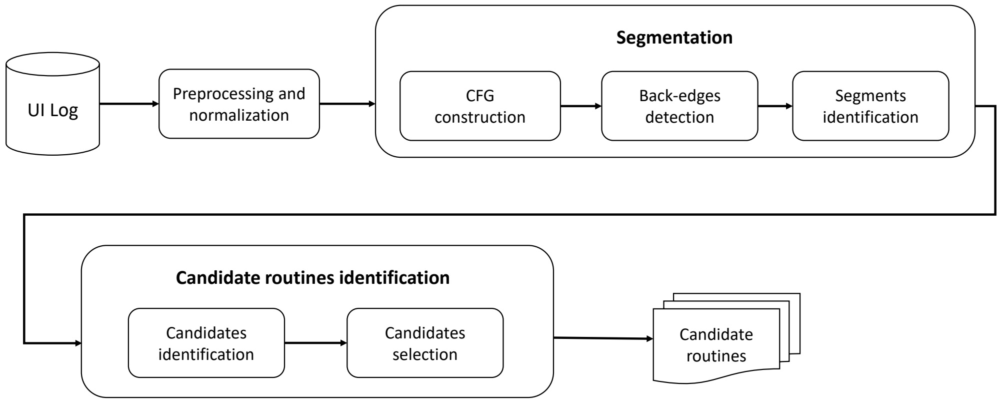


Figure 10: Outline of the approach for discovering candidate routines

5.1.1. UI Log Preprocessing and Normalization

Before we proceed, we give the formal definitions necessary to support the subsequent discussions.

[1]Corresponding to [62] and partially [63]

Definition 5.1.1 (**User Interaction (UI)).** *A* user interaction *(UI) is a tuple $u = (t, \tau, P_\tau, Z, \phi)$, where: t is a UI timestamp; τ is a UI type (e.g., click button, copy cell); P_τ is a set of UI parameters (e.g. button name, worksheet name, URL, etc.); Z is a set of UI parameters values; and $\phi : P_\tau \to Z$ is a function that maps UI parameters onto values.*

Definition 5.1.2 (**UI Log).** *A UI Log Σ is a sequence of user interactions $\Sigma = \langle u_1, u_2, \ldots, u_n \rangle$, ordered by timestamp, i.e. $u_{i|t} < u_{j|t}$ (where $u_{i|t}$ is the timestamp of u_i) for any $i, j \mid 1 \leq i < j \leq n$. In the remainder of this thesis, we refer to UI log also as a log.*

Table 6 shows an example of a UI log, which we will use herein as running example.

Table 6: Running example for the candidate routines discovery approach

	UI	**UI**	**Payload**					
Row	**Timestamp**	**Type**	P_1	P_2	P_3	P_4	P_5	P_6
1	2019-03-03T19:02:18	Click button (Web)	https://unimelb.edu.au	New Record	newRecord	button	–	–
2	2019-03-03T19:02:20	Select cell (Excel)	StudentRecords	Sheet1	A	2	Albert Rauf	–
3	2019-03-03T19:02:23	Copy cell (Excel)	StudentRecords	Sheet1	A	2	Albert Rauf	Albert Rauf
4	2019-03-03T19:02:25	Select field (Web)	https://unimelb.edu.au	Full Name	name	“”	–	–
5	2019-03-03T19:02:26	Paste (Web)	https://unimelb.edu.au	Full Name	name	“”	Albert Rauf	–
6	2019-03-03T19:02:28	Edit field (Web)	https://unimelb.edu.au	Full Name	name	text	Albert Rauf	–
7	2019-03-03T19:02:30	Select cell (Excel)	StudentRecords	Sheet1	B	2	11/04/1986	–
8	2019-03-03T19:02:31	Copy cell (Excel)	StudentRecords	Sheet1	B	2	11/04/1986	11/04/1986
9	2019-03-03T19:02:34	Select field (Web)	https://unimelb.edu.au	Date	date	“”	–	–
10	2019-03-03T19:02:37	Paste (Web)	https://unimelb.edu.au	Date	date	“”	11/04/1986	–
11	2019-03-03T19:02:40	Edit field (Web)	https://unimelb.edu.au	Date	date	text	11-04-1986	–
12	2019-03-03T19:07:30	Select cell (Excel)	StudentRecords	Sheet1	C	2	+61 043 512 4834	–
13	2019-03-03T19:07:33	Copy cell (Excel)	StudentRecords	Sheet1	C	2	+61 043 512 4834	+61 043 512 4834
14	2019-03-03T19:07:40	Select field (Web)	https://unimelb.edu.au	Phone	phone	“”	–	–
15	2019-03-03T19:07:46	Paste (Web)	https://unimelb.edu.au	Phone	phone	“”	+61 043 512 4834	–
16	2019-03-03T19:07:48	Edit field (Web)	https://unimelb.edu.au	Phone	phone	text	043-512-4834	–
17	2019-03-03T19:07:50	Select cell (Excel)	StudentRecords	Sheet1	D	2	Germany	–
18	2019-03-03T19:07:52	Copy cell (Excel)	StudentRecords	Sheet1	D	2	Germany	Germany
19	2019-03-03T19:07:55	Select field (Web)	https://unimelb.edu.au	Country of residence	country	“”	–	–
20	2019-03-03T19:07:57	Paste (Web)	https://unimelb.edu.au	Country of residence	country	“”	Germany	–
21	2019-03-03T19:07:59	Edit field (Web)	https://unimelb.edu.au	Country of residence	country	text	Germany	–
22	2019-03-03T19:08:02	Edit field (Web)	https://unimelb.edu.au	Student status	status	select	Domestic	–
23	2019-03-03T19:08:05	Edit field (Web)	https://unimelb.edu.au	Student status	status	select	International	–
24	2019-03-03T19:08:08	Click button (Web)	https://unimelb.edu.au	Submit	submit	submit	–	–
25	2019-03-03T19:08:12	Click button (Web)	https://unimelb.edu.au	New Record	newRecord	button	–	–
26	2019-03-03T19:08:15	Select cell (Excel)	StudentRecords	Sheet1	B	3	20/06/1987	–
27	2019-03-03T19:08:18	Copy cell (Excel)	StudentRecords	Sheet1	B	3	20/06/1987	20/06/1987
28	2019-03-03T19:08:21	Select field (Web)	https://unimelb.edu.au	Date	date	“”	–	–
29	2019-03-03T19:08:26	Paste (Web)	https://unimelb.edu.au	Date	date	“”	20/06/1987	–
30	2019-03-03T19:08:28	Edit field (Web)	https://unimelb.edu.au	Date	date	text	20-06-1987	–
31	2019-03-03T19:08:32	Select cell (Excel)	StudentRecords	Sheet1	C	3	+61 519 790 1066	–
32	2019-03-03T19:08:34	Copy cell (Excel)	StudentRecords	Sheet1	C	3	+61 519 790 1066	+61 519 790 1066
33	2019-03-03T19:08:36	Select field (Web)	https://unimelb.edu.au	Phone	phone	“”	–	–
34	2019-03-03T19:08:38	Paste (Web)	https://unimelb.edu.au	Phone	phone	“”	+61 519 790 1066	–
35	2019-03-03T19:08:39	Edit field (Web)	https://unimelb.edu.au	Phone	phone	text	519-790-1066	–
36	2019-03-03T19:08:40	Select cell (Excel)	StudentRecords	Sheet1	A	3	Audrey Backer	–
37	2019-03-03T19:08:41	Copy cell (Excel)	StudentRecords	Sheet1	A	3	Audrey Backer	Audrey Backer
38	2019-03-03T19:08:42	Select field (Web)	https://unimelb.edu.au	Full Name	name	“”	–	–
39	2019-03-03T19:08:44	Paste (Web)	https://unimelb.edu.au	Full Name	name	“”	Audrey Backer	–
40	2019-03-03T19:08:46	Edit field (Web)	https://unimelb.edu.au	Full Name	name	text	Audrey Backer	–
41	2019-03-03T19:08:50	Select cell (Excel)	StudentRecords	Sheet1	D	2	Germany	–
42	2019-03-03T19:08:52	Copy cell (Excel)	StudentRecords	Sheet1	D	2	Germany	Germany
43	2019-03-03T19:08:58	Select cell (Excel)	StudentRecords	Sheet1	D	3	Australia	–
44	2019-03-03T19:09:01	Copy cell (Excel)	StudentRecords	Sheet1	D	3	Australia	Australia
45	2019-03-03T19:09:05	Select field (Web)	https://unimelb.edu.au	Country of residence	country	“”	–	–
46	2019-03-03T19:09:08	Paste (Web)	https://unimelb.edu.au	Country of residence	country	“”	Australia	–
47	2019-03-03T19:09:10	Edit field (Web)	https://unimelb.edu.au	Country of residence	country	text	Australia	–
48	2019-03-03T19:09:14	Edit field (Web)	https://unimelb.edu.au	Student status	status	select	Domestic	–
49	2019-03-03T19:09:20	Click button (Web)	https://unimelb.edu.au	Submit	submit	submit	–	–
…	…	…	…	…	…	…	…	…

An UI log often may contain UIs that do not contribute to the routine(s) we intend to discover. On the one hand, some UIs are completely independent of the operations in a routine. For example, a worker might perform web browsing operations (e.g., visiting a social networking site) while executing a routine. On the other hand, some UIs correspond to operations that are normally part of a routine, but that do not change the effect of the routine (i.e. they are redundant). For example, copying twice consecutively from the same cell in a spreadsheet has the same effect as copying a single time from that cell (i.e., copying from a cell is an idempotent operation, unless another operation changes the contents of the cell in-between the two copy operations). Such UIs are considered to be noise and have to be removed in order to obtain correct routines. The first type of noise (i.e., UIs that are not related to routine execution) is expected to be irregular and infrequent. Such noise will be handled during the discovery of candidate routines. To identify and remove the second type of noise (i.e., redundant UIs), we rely on three search-and-replace rules defined as regular expressions that operate as follows.

1. Remove UIs of type *copy* that are not eventually followed by UI of type *paste* before another UI of type *copy* occurs (e.g., Table 6, row 42). This rule captures the fact that multiple subsequent copy operations have the same effect as the last copy operation, unless in the middle, the contents that are copied into the clipboard are used via a paste operation.
2. Remove UIs of type type *edit* (or *paste*) that are followed by another UI of the same type that targets the same object and overwrites its content before the current value was copied (e.g., Table 6, row 22). This rule captures the overriding effect of paste and edit operations (i.e., the last paste edit operation on a field of cell overrides previous paste/edits on that cell).
3. Remove UIs of type *select cell* and *select field* (e.g., Table 6, rows 2, 4, 7). The rationale for this rule is that, in the representation of UI logs we have adopted, the cell (or field) upon which an operation is applied is recorded as part of the operation, and therefore the position of this cell (or field) does not need to be inferred from the navigation operations in the UI log.

Note that, given an unsegmented log, it is impossible to apply the third rule straightforward, as removing the first UI of type *edit* (considered redundant) may be an error if the second UI of type *edit* belongs to successive task execution. Therefore, we postpone the application of the third rule after the segmentation step. The filtering rules are applied recursively on the log until no more UIs are removed, and the log is assumed to be free of *detectable* noise.

The above rules capture basic properties of copy, paste and edit operations. We do not claim that these rules capture every type of redundancy in a UI log. Devising and applying more sophisticated filtering algorithms is outside the scope of this thesis, and we leave it as an avenue for future work.

After filtering the log, the vast majority of the UIs are unique because they differ by their unique payload. Even the UIs capturing the same action within the same task execution (or different task executions) would appear different. To discover each task execution recorded in the log, we need to detect all the UIs that even having different payloads correspond to the same action within the same or different task execution(s).

Given a UI, its payload can be divided into *data parameters* and *context parameters*. The former store the data values used during the execution of the tasks, e.g., the value of text fields or copied content. Consequently, *data parameters* usually have different values in different task executions. In contrast, the latter capture the context in which the UIs were performed, e.g., the application and the location within the application. Therefore, *context parameters* of the same UI within a task are likely to have the same values across different task executions. For example, the payload of a UI of type *copy cell* has the following parameters: *workbook name* (the Excel file name); *worksheet name* (within the Excel file); *cell column* (i.e., the column of the cell in the worksheet that was selected for the UI); *cell row* (i.e., the row of the cell in the worksheet that was selected for the UI); *value* (i.e., the current value of the cell selected for the UI); *copied content* (the content copied as the result of the UI). Here, *workbook name*, *worksheet name*, *cell column/row* are *context parameters*, while *copied content* and *value* are *data parameters*. Different context parameters characterize different UI types. For example, a UI of type *click button* performed in a web browser has only these context parameters: *URL*; *name* (i.e., the label of the button); *ID* (of the button, as an element in the HTML page); and *type*. Often, context parameters are determined by the type of UI, however, to reduce the chance of possible automated misinterpretations, we allow the user to configure the context parameters of various UI types manually.

To segment an input UI log, we rely on the context parameters of the UIs. We call a UI whose payload has been reduced to its context parameters a *normalized UI*.

Definition 5.1.3 (Normalized UI). *Given a UI $u = (t, \tau, P_\tau, Z, \phi)$, the UI $\bar{u} = (t, \tau, \bar{P_\tau}, \bar{Z}, \phi)$ is its normalized version, where $\bar{Z}$ contains only the values of the parameters in $\bar{P_\tau}$, where $\bar{P_\tau}$ is a set of context parameters.*

Two normalized UIs $u_1 = (t_1, \tau, \bar{P_\tau}, \bar{Z_1}, \phi_1)$ and $u_2 = (t_2, \tau, \bar{P_\tau}, \bar{Z_2}, \phi_2)$ are *equivalent*, denoted by $u_1 = u_2$ iff $\forall p \in \bar{P_\tau} \Rightarrow \phi_1(p) = \phi_2(p)$.

A log in which all the UIs have been normalized is a *normalized log*, and we refer to it with the notation $\bar{\Sigma} = \langle \bar{u_1}, \bar{u_2}, \ldots, \bar{u_n} \rangle$. Table 6 and Table 7 show, respectively, a fragment of a log and its normalized version.

Intuitively, in a normalized log, the chances that two executions of the same task have the same sequence (or set) of normalized UIs are high because they have only context parameters. We leverage such a characteristic of a normalized log to identify its segments (i.e., start and end of each executed task) and then the routine(s) within the segments.

Table 7: Normalized running example after preprocessing

	UI	UI	Payload			
Row	Timestamp	Type	P_1	P_2	P_3	P_4
1	2019-03-03T19:02:18	Click button (Web)	http://www.unimelb.edu.au	New Record	newRecord	button
2	2019-03-03T19:02:23	Copy cell (Excel)	StudentRecords	Sheet1	A	–
3	2019-03-03T19:02:26	Paste (Web)	http://www.unimelb.edu.au	Full Name	name	–
4	2019-03-03T19:02:28	Edit field (Web)	http://www.unimelb.edu.au	Full Name	name	text
5	2019-03-03T19:02:31	Copy cell (Excel)	StudentRecords	Sheet1	B	–
6	2019-03-03T19:02:37	Paste (Web)	http://www.unimelb.edu.au	Date	date	–
7	2019-03-03T19:02:40	Edit field (Web)	http://www.unimelb.edu.au	Date	date	text
8	2019-03-03T19:07:33	Copy cell (Excel)	StudentRecords	Sheet1	C	–
9	2019-03-03T19:07:40	Paste (Web)	http://www.unimelb.edu.au	Phone	phone	–
10	2019-03-03T19:07:48	Edit field (Web)	http://www.unimelb.edu.au	Phone	phone	text
11	2019-03-03T19:07:50	Copy cell (Excel)	StudentRecords	Sheet1	D	–
12	2019-03-03T19:07:55	Paste (Web)	http://www.unimelb.edu.au	Country of residence	country	–
13	2019-03-03T19:08:02	Edit field (Web)	http://www.unimelb.edu.au	Country of residence	country	text
14	2019-03-03T19:08:05	Edit field (Web)	http://www.unimelb.edu.au	Student status	status	select
15	2019-03-03T19:08:08	Click button (Web)	http://www.unimelb.edu.au	Submit	submit	submit
16	2019-03-03T19:08:12	Click button (Web)	http://www.unimelb.edu.au	New Record	newRecord	button
17	2019-03-03T19:08:17	Copy cell (Excel)	StudentRecords	Sheet1	B	–
18	2019-03-03T19:08:21	Paste (Web)	http://www.unimelb.edu.au	Date	date	–
19	2019-03-03T19:08:28	Edit field (Web)	http://www.unimelb.edu.au	Date	date	text
20	2019-03-03T19:08:35	Copy cell (Excel)	StudentRecords	Sheet1	C	–
21	2019-03-03T19:08:38	Paste (Web)	http://www.unimelb.edu.au	Phone	phone	–
22	2019-03-03T19:08:39	Edit field (Web)	http://www.unimelb.edu.au	Phone	phone	text
23	2019-03-03T19:08:40	Copy cell (Excel)	StudentRecords	Sheet1	A	–
24	2019-03-03T19:08:42	Paste (Web)	http://www.unimelb.edu.au	Full Name	name	–
25	2019-03-03T19:08:43	Edit field (Web)	http://www.unimelb.edu.au	Full Name	name	text
26	2019-03-03T19:08:45	Copy cell (Excel)	StudentRecords	Sheet1	D	–
27	2019-03-03T19:08:47	Paste (Web)	http://www.unimelb.edu.au	Country of residence	country	–
28	2019-03-03T19:08:49	Edit field (Web)	http://www.unimelb.edu.au	Country of residence	country	text
29	2019-03-03T19:08:52	Edit field (Web)	http://www.unimelb.edu.au	Student status	status	select
30	2019-03-03T19:08:53	Click button (Web)	http://www.unimelb.edu.au	Submit	submit	submit
…	…	…	…	…	…	…

5.1.2. Segmentation

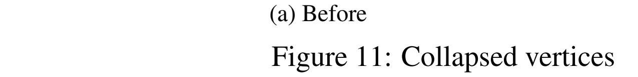
A log may capture long working sessions, where a worker performs multiple instances of one or more tasks. The next step of our approach decomposes the log into *segments* that identify the start and the end of each recorded task in the log. Given a normalized log, we generate its control-flow graph (CFG). A CFG is a graph where each vertex represents a different normalized UI, and each edge captures a directly-follows relation between the two normalized UIs represented by the source and the target vertices of the edge. A CFG has an explicit source vertex representing the first normalized UI recorded in the log.

Given a log, the directly follows relation on UI is defined as follows.

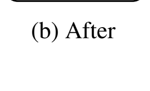
Definition 5.1.4 (Directly-follows relation). *Let $\bar{\Sigma} = \langle \bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n \rangle$ be a normalized log. Given two UIs, $\bar{u}_x, \bar{u}_y \in \bar{\Sigma}$, we say that $\bar{u}_y$ directly-follows $\bar{u}_x$, i.e., $\bar{u}_x \leadsto \bar{u}_y$, iff $\bar{u}_{x|t} < \bar{u}_{y|t} \wedge \nexists \bar{u}_z \in \bar{\Sigma} \mid \bar{u}_{x|t} \leq \bar{u}_{z|t} \leq \bar{u}_{y|t}$.*

Definition 5.1.5 (Control-Flow Graph (CFG)). *Given a normalized log, $\bar{\Sigma} = \langle \bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n \rangle$, let $\bar{A}$ be the set of all the normalized UIs in $\bar{\Sigma}$. A Control-Flow Graph (CFG) is a tuple $G = (V, E, \hat{v}, \hat{e})$, where: V is the set of vertices of the graph, each vertex maps one UI in $\bar{A}$; $E \subseteq V \times V$ is the set of edges of the graph, and each $(v_i, v_j) \in E$ represents a directly-follows relation between the UIs mapped by v_i and v_j; $\hat{v}$ is the graph* entry vertex*, such that $\forall v \in V \nexists (v, \hat{v}) \in E \wedge \nexists (\hat{v}, v) \in E$; while $\hat{e} = (\hat{v}, v_0)$ is the graph* entry edge*, such that v_0 maps $\bar{u}_1$. We note that $\hat{v} \notin V$, and $\hat{e} \notin E$, since they are artificial elements of the graph.*

A CFG is likely cyclic since a loop represents the start of a new execution of the task recorded in the log. Indeed, in an ideal scenario, once a task execution ends with a certain UI (a vertex in the CFG), the next UI (i.e., the first UI of the next task execution) should have already been mapped to a vertex of the CFG, and a loop will be generated. In such a case, all the vertices in the loop represent the UIs performed during the execution of the task. If several different tasks are recorded in sequence in the same log, we would observe several disjoint loops in the CFG, while if a task has repetitive subtasks, we would observe nested loops in the CFG. Figure 12 shows the CFG generated from the log captured in Table 7, we note that for simplicity we collapsed some vertices as shown in Figure 11.



(a) Before



(b) After

Figure 11: Collapsed vertices

Once the CFG is generated, we turn our attention to identifying its back-edges (i.e., its loops). By identifying the CFG back-edges and their UIs, we extract the start and end UIs of the repeated task. These UIs are used to mark the boundaries between task executions. The back-edges of a CFG can be identified by analyzing

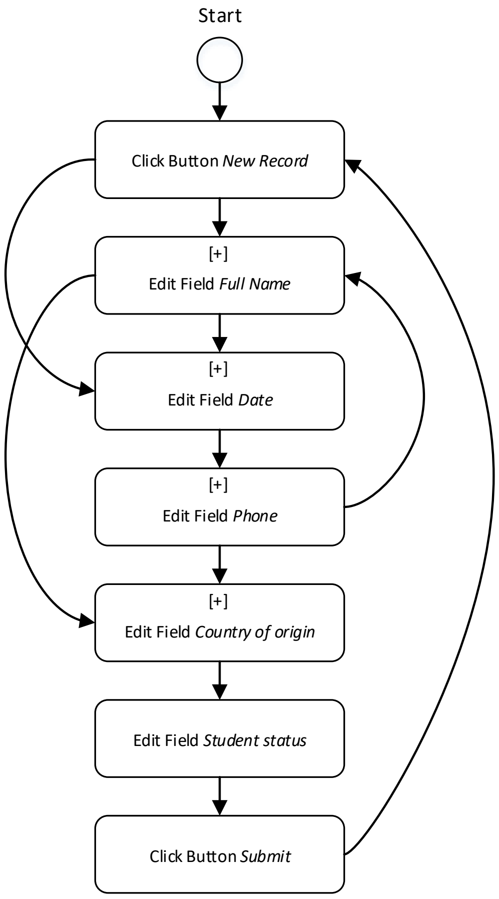


Figure 12: Control-flow graph for the running example

the CFG Strongly Connected Components (SCCs). Given a graph, an SCC is a subgraph where for all its pairs of vertices, there exist a set of edges connecting the pair of vertices such that all the sources and targets of these edges belong to the subgraph.

Definition 5.1.6 (CFG Path). *Given a CFG $G = (V, E, \hat{v}, \hat{e})$, a CFG path is a sequence of vertices $p_{v_1,v_k} = \langle v_1, \dots, v_k \rangle$ such that for each $i \in [1, k-1] \Rightarrow v_i \in V \cup \{\hat{v}\} \wedge \exists (v_i, v_{i+1}) \in E \cup \{\hat{e}\}$.*

Definition 5.1.7 (Strongly Connected Component (SCC)). *Given a graph $G = (V, E, \hat{v}, \hat{e})$, a strongly connected component (SCC) of G is a pair $\delta = (\bar{V}, \bar{E})$, where $\bar{V} = \{v_1, v_2, \dots, v_m\} \subseteq V$ and $\bar{E} = \{e_1, e_2, \dots, e_k\} \subseteq E$ such that $\forall v_i, v_j \in \bar{V} \exists p_{v_i,v_j} \mid \forall v \in p \Rightarrow v \in \bar{V}$. Given an SCC $\delta = (\bar{V}, \bar{E})$, we say that δ is* non-trivial *iff $|\bar{V}| > 1$. Given a graph G, Δ_G denotes the set of all the non-trivial SCCs in G.*

Algorithm 1 and Algorithm 2 describe how we identify the SCCs of the CFG.

Given a CFG $G = (V, E, \hat{v}, \hat{e})$, we first build its dominator tree Θ (Algorithm 1, line 2), which captures domination relations between the vertices of the CFG.

Algorithm 1: Back-edges detection

```
input  : CFG $G$
output : Back-edges Set $B$
1  $B \leftarrow \varnothing$;
2  Dominator Tree $\Theta \leftarrow$ computeDominatorTree($G$);
3  Set $\Delta_G \leftarrow$ findSCCs($G$);
4  foreach $\delta \in \Delta_G$ do AnalyseSCC($\delta$, $\Theta$, $B$) ;
5  return $B$;
```

Algorithm 2: Analyse SCC

```
input  : SCC $\delta = (\bar{V}, \bar{E})$, Dominator Tree $\Theta$, Back-edges Set $B$
1  Header $\hat{h} \leftarrow$ findHeader($\delta$, $\Theta$);
2  if $\hat{h} \neq null$ then
3      Set $I \leftarrow$ getIncomingEdges($\delta$, $\hat{h}$);
4      $B \leftarrow B \cup I$;
5      $\bar{E} \leftarrow \bar{E} \setminus I$;
6  else
7      Set $L \leftarrow$ findLoopEdges($\delta$);
8      Edge $e \leftarrow$ getTheDeepestEdge($\delta$, $L$);
9      remove $e$ from $\bar{E}$;
10 Set $\Delta_\delta \leftarrow$ findSCCs($\delta$);
11 foreach $\gamma \in \Delta_\delta$ do AnalyseSCC($\gamma$, $\Theta$, $B$) ;
```

Figure 13 shows the dominator tree of the CFG in Figure 12.

Then, we discover the set of all non-trivial SCCs (Δ_G) by applying the Kosaraju's algorithm [64] and removing the trivial SCCs (Algorithm 1, line 3).

The main idea behind Kosaraju's algorithm is that if vertex u can be visited from vertex v and, at the same time, vertex v can be visited from vertex u then these two vertices are strongly connected and belong to the same strongly connected component. Accordingly, the algorithm requires two Depth-first search (DFS) traversals [65] over the graph to find all strongly connected components. First, a DFS is performed on the original graph, and a post-order of the explored search tree is saved. Then, starting from the last vertex in the post-order, a DFS on the transposed graph is performed. Transposed graph $G\prime$ of a directed graph G is a graph with the same set of vertices as G and all edges of G with reversed orientation, such that if G contains an edge (u, v), then $G\prime$ contains an edge (v, u) and vice versa. As a result, we will obtain a forest consisting of trees that contain all the nodes that are reachable from the corresponding root. If a pair of vertices is present in the same tree of the original and the transposed traversal, these vertices are placed in the same strongly connected component. If a vertex in the transposed graph is not connected to any other vertex, it is placed in a separate strongly connected component, and the next vertex in the reversed post order is taken as the start vertex for the DFS. The algorithm stops when all the vertices are assigned to strongly connected components.

For each $\delta = (\bar{V}, \bar{E}) \in \Delta_G$, we discover its *header* using the dominator tree (Algorithm 2, line 1). The header of a dominator tree δ is a special vertex $\hat{h} \in \bar{V}$, such that $\forall p_{\hat{v},v} \mid v \in \bar{V} \Rightarrow \hat{h} \in p_{\hat{v},v}$, i.e., the *header* $\hat{h}$ (a.k.a. the SCC entry) is

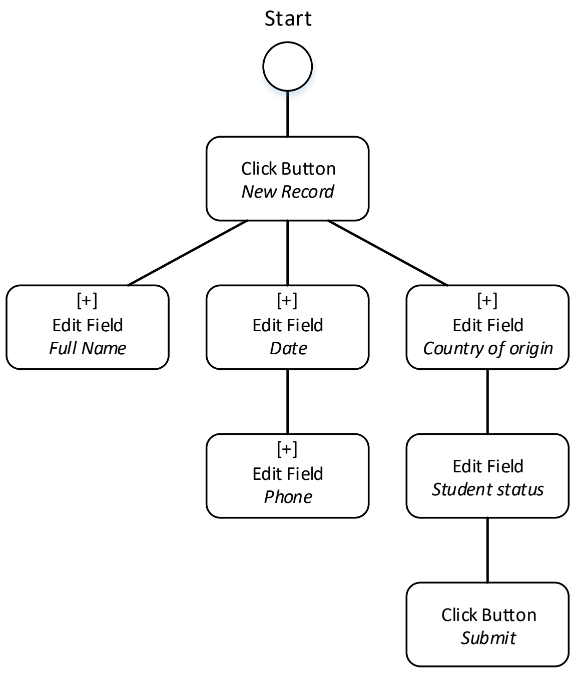


Figure 13: Dominator tree for the running example

the SCC vertex that dominates all the other SCC vertices. Once we have $\hat{h}$, we can identify the back-edges as $(v, \hat{h})$ with $v \in \bar{V}$ (line 3). Finally, the identified back-edges are stored and removed (lines 4 and 5) in order to look for nested SCCs and their back-edges by recursively executing Algorithm 2 (line 11), until no more SCCs and back-edges are found. However, if we detect an SCC that does not have a header vertex (formally, the SCC is irreducible), we cannot identify the SCC back-edges. In such a case, we collect via a depth-first search of the CFG the edges $(v_x, v_y) \in \bar{E}$ such that v_y is topologically deeper than v_x - we call these edges *loop-edges* of the SCC (line 7). Then, out of all the loop-edges, we store (and remove from the SCC) the one having target and source connected by the longest *simple path* entirely contained within the SCC (lines 8 to 9).

Given the CFG presented in Figure 12 and its corresponding dominator tree (see Figure 13), we identify the SCC that consists of all the vertices except the *entry vertex*. Then, by applying Algorithm 2, we identify: the SCC header – *Click Button [New Record]*; and the only back-edge – (*Click Button [Submit]*, *Click Button [New Record]*), which we save and remove from the SCC. After the removal of this back-edge, we identify the nested SCC that contains edits of the *Full Name*, *Date*, and *Phone* fields. Note that this second SCC does not have a header because it is irreducible, due to its multiple entries (*Edit Field [Full Name]* and *Edit Field [Date]*). However, by applying the depth-first search, we identify as candidate loop-edge for removal: (*Edit Field [Phone]*, *Edit Field [Full Name]*). After we remove this edge from the CFG, no SCCs are left, so Algorithm 2 terminates.

At this point, we collected all the back-edges of the CFG. Next, we use them to segment the log. We do so by applying Algorithm 3.

Algorithm 3: Segmentation

```
input  : Normalized UI log $\bar{\Sigma}$, Back-edges Set $B$
output : Segments List $\Psi$
1  Set $\Psi \leftarrow \varnothing$;
2  Set $T \leftarrow$ getTargets($B$);
3  Set $S \leftarrow$ getSources($B$);
4  Boolean WithinSegment $\leftarrow$ false;
5  Normalized UI $u_0 \leftarrow$ null;
6  Queue $s \leftarrow \varnothing$;
7  for $i \leftarrow 1$ to $size(\bar{\Sigma})$ do
8      Normalized UI $\bar{u} \leftarrow$ getUI($\bar{\Sigma}$, $i$);
9      if $\bar{u} \in T$ then
10         if $WithinSegment = false$ then
11             $s \leftarrow \varnothing$;
12             append $\bar{u}$ to $s$;
13             $u_0 \leftarrow \bar{u}$;
14             WithinSegment $\leftarrow$ true;
15         else
16             append $\bar{u}$ to $s$;
17     else
18         if $WithinSegment = true$ then
19             append $\bar{u}$ to $s$;
20             if $\bar{u} \in S \wedge (\bar{u}, u_0) \in B$ then
21                 $\Psi \leftarrow \Psi \cup \{s\}$;
22                 WithinSegment $\leftarrow false$;
23 return $\Psi$;
```

First, we retrieve all the targets and sources of all the back-edges in the CFG and collect their corresponding UIs (lines 2 and 3). Each UI mapped onto a back-edge target is an eligible segment starting point (from now on, *segment-start UI*). A back-edge conceptually captures the end of a task execution, while its target represents the first UI of the next task execution. By applying the same reasoning, each UI mapped onto the source of a back-edge is an eligible segment ending point (from now on, *segment-end UI*). Then, we sequentially scan all the UIs in the log (line 7). When we encounter a segment-start UI (line 9), and we are not already within a segment (see line 10), we create a new segment (s, a list of UIs), we append the segment-start UI ($\bar{u}$), and we store it in order to match it with the correct segment-end UI (line 11 to 14). The following underlying assumption drives our strategy to detect segments in the log: a specific segment-end UI will be followed by the same segment-start UI to match segment-end and segment-start UIs exploiting back-edge's sources and targets (respectively). If the UI is not a segment-start (line 17), we check if we are within a segment (line 18) and, if not, we discard the UI, assuming it is noise since it fell between the previous segment-end UI and the next segment-start UI. Otherwise, we append the UI to the current segment, and we check if this UI is a segment-end matching the current segment-start UI (line 20). If that is the case, we reached the end of the segment, and we add it to the set of segments (line 21); otherwise, we continue reading the segment.

Table 8 shows the segment-start and the segment-end UIs (highlighted in green and red, respectively), delimiting two segments within the normalized UI log of our running example (see Table 7).

Table 8: Segments identification

Row	UI Timestamp	UI Type	Payload P_1	P_2	P_3	P_4
1	2019-03-03T19:02:18	Click button (Web)	http://www.unimelb.edu.au	New Record	newRecord	button
2	2019-03-03T19:02:23	Copy cell (Excel)	StudentRecords	Sheet1	A	–
3	2019-03-03T19:02:26	Paste (Web)	http://www.unimelb.edu.au	Full Name	name	–
4	2019-03-03T19:02:28	Edit field (Web)	http://www.unimelb.edu.au	Full Name	name	text
5	2019-03-03T19:02:31	Copy cell (Excel)	StudentRecords	Sheet1	B	–
6	2019-03-03T19:02:37	Paste (Web)	http://www.unimelb.edu.au	Date	date	–
7	2019-03-03T19:02:40	Edit field (Web)	http://www.unimelb.edu.au	Date	date	text
8	2019-03-03T19:07:33	Copy cell (Excel)	StudentRecords	Sheet1	C	–
9	2019-03-03T19:07:40	Paste (Web)	http://www.unimelb.edu.au	Phone	phone	–
10	2019-03-03T19:07:48	Edit field (Web)	http://www.unimelb.edu.au	Phone	phone	text
11	2019-03-03T19:07:50	Copy cell (Excel)	StudentRecords	Sheet1	D	–
12	2019-03-03T19:07:55	Paste (Web)	http://www.unimelb.edu.au	Country of residence	country	–
13	2019-03-03T19:08:02	Edit field (Web)	http://www.unimelb.edu.au	Country of residence	country	text
14	2019-03-03T19:08:05	Edit field (Web)	http://www.unimelb.edu.au	Student status	status	select
15	2019-03-03T19:08:08	Click button (Web)	http://www.unimelb.edu.au	Submit	submit	submit
16	2019-03-03T19:08:12	Click button (Web)	http://www.unimelb.edu.au	New Record	newRecord	button
17	2019-03-03T19:08:17	Copy cell (Excel)	StudentRecords	Sheet1	B	–
18	2019-03-03T19:08:21	Paste (Web)	http://www.unimelb.edu.au	Date	date	–
19	2019-03-03T19:08:28	Edit field (Web)	http://www.unimelb.edu.au	Date	date	text
20	2019-03-03T19:08:35	Copy cell (Excel)	StudentRecords	Sheet1	C	–
21	2019-03-03T19:08:38	Paste (Web)	http://www.unimelb.edu.au	Phone	phone	–
22	2019-03-03T19:08:39	Edit field (Web)	http://www.unimelb.edu.au	Phone	phone	text
23	2019-03-03T19:08:40	Copy cell (Excel)	StudentRecords	Sheet1	A	–
24	2019-03-03T19:08:42	Paste (Web)	http://www.unimelb.edu.au	Full Name	name	–
25	2019-03-03T19:08:43	Edit field (Web)	http://www.unimelb.edu.au	Full Name	name	text
26	2019-03-03T19:08:45	Copy cell (Excel)	StudentRecords	Sheet1	D	–
27	2019-03-03T19:08:47	Paste (Web)	http://www.unimelb.edu.au	Country of residence	country	–
28	2019-03-03T19:08:49	Edit field (Web)	http://www.unimelb.edu.au	Country of residence	country	text
29	2019-03-03T19:08:52	Edit field (Web)	http://www.unimelb.edu.au	Student status	status	select
30	2019-03-03T19:08:53	Click button (Web)	http://www.unimelb.edu.au	Submit	submit	submit
…	…	…	…	…	…	…

5.1.3. Candidate routines identification

Once the log has been segmented, we move to the identification of the candidate routines. The identification step is based on the CloFast sequence mining algorithm [45]. CloFast discovers closed frequent sequences of itemsets by exploiting sparse and vertical id-lists. Specifically, it uses sparse and vertical id-lists to fast count the support of patterns, check their closure, and prune the search space with the minimal number of database scans. It does not need to maintain already mined closed sequences for pruning and checking whether the newly discovered patterns are closed. Thus, it outperforms state-of-the-art algorithms both in time and memory consumption, especially when mining long closed sequences.

To integrate CloFast in our approach, we have to define the structure of the sequential patterns we want to identify. In the context of our problem, we define a *sequential pattern* within a UI log as a sequence of normalized UIs always occurring in the same order in different segments, yet allowing gaps between the UIs belonging to the pattern. For example, if we consider the following three segments: $\langle u_1, u_y, u_2, u_3\rangle$, $\langle u_1, u_2, u_x, u_3\rangle$, and $\langle u_1, u_x, u_2, u_3\rangle$; they all contain the same sequential pattern that is $\langle u_1, u_2, u_3\rangle$.

Furthermore, we define the *support* of a sequential pattern as the ratio of segments containing the pattern and the total number of segments. We refer to *closed* patterns and *frequent* patterns (relatively to an input threshold) as they are known in the literature. Specifically, a frequent pattern is a pattern that appears in at least a number of occurrences indicated by the threshold, while a closed pattern is a pattern that is not included in another pattern having exactly the same support. By applying CloFast to the log segments, we discover all the *frequent closed* sequential patterns.

Some of these patterns may be *overlapping*, which (in our context) means that they share some UIs. An example of overlapping patterns is the following, given three segments: $\langle u_1, u_y, u_2, u_3, u_x, u_4\rangle$, $\langle u_1, u_y, u_2, u_x, u_3, u_4\rangle$, and $\langle u_1, u_x, u_2, u_3, u_4\rangle$; $\langle u_1, u_2, u_3, u_4\rangle$ and $\langle u_1, u_x, u_4\rangle$ are sequential patterns, but they overlap due to the shared UIs: u_1 and u_4. In practice, each UI belongs to only one routine. Therefore, we are interested in discovering only non-overlapping patterns. For this purpose, we implemented an optimization that we use on top of CloFast. Given the set of patterns discovered by CloFast, we rank them by a pattern quality criterion, and we select the best pattern (i.e., the top one in the ranking). We integrated four pattern quality criteria to select the candidate routines: pattern frequency, pattern length, pattern coverage, and pattern cohesion score [40].

Definition 5.1.8 (Pattern length). *Given a pattern $P = \langle u1, \ldots, u_n\rangle$,* pattern length *is the length of sequence P.*

Definition 5.1.9 (Pattern frequency). *Given a pattern $P = \langle u_1, \ldots, u_n\rangle$, and a segmented log $\Sigma_{seg} = \{S_1, \ldots, S_m\}$, where $S_i, i \in [1, m]$ are segments,* pattern frequency *is the number of segments that contain P as a subsequence.*

Definition 5.1.10 (Pattern coverage). *Given a pattern $P = \langle u_1, \ldots, u_n\rangle$, and a segmented log $\Sigma_{seg} = \{S_1, \ldots, S_m\}$, where $S_i, i \in [1, m]$ are segments,* pattern coverage *is the ratio of UIs in the log belonging to P to the total size of the log $\sum_{i=1}^{m} |S_i|$, where $|S_i|$ is the length of S_i.*

In order to explain the pattern cohesion, we first need to define the notions of minimum occurrence window and outlier.

Definition 5.1.11 (Minimum occurrence window). *The* minimum occurrence window $W_{P,S}$ *of a pattern P in a segment S is the shortest subsequence of S that contains P.*

Definition 5.1.12 (Outlier). *Given a minimum occurrence window $W_{P,S} \in S$ for pattern P in segment S, an* outlier *is a UI $o \in W_{P,S}$ that does not belong to a pattern P or that appears at wrong position with respect to P.*

Definition 5.1.13 (Pattern cohesion score). *Given a pattern $P = \langle u_1, \ldots, u_n \rangle$ and a segmented log $\Sigma_{seg} = \{S_1, \ldots, S_m\}$, where $S_i, i \in [1,m]$ are segments,* pattern cohesion *is a signed difference between the pattern length $|P|$ and a median number of outliers in the minimum occurrence windows W_{P,S_i}, where $i \in [1,m]$.*

Cohesion prioritizes the patterns whose UIs appear consecutively without (or with few) gaps while also considering the pattern length. As noted by Dev and Liu [40], the cohesion metric captures the fact that the instances of a given pattern may be interspersed by infrequent UIs corresponding to noise.

Given a segmented log $\Sigma_{seg} = \{S_1 = \langle u_1, u_y, u_2, u_3, u_x, u_4 \rangle, S_2 = \langle u_1, u_y, u_2, u_x, u_3, u_4 \rangle, S_3 = \langle u_1, u_x, u_2, u_3, u_4 \rangle\}$ and a pattern $P_1 = \langle u_1, u_2, u_3, u_4 \rangle$, we can compute all its metrics as follows. $Length(P_1) = 4$; $Frequency(P_1) = 3$; $Coverage(P_1) = 12/17 = 0.706$; $Cohesion(P_1) = 4 - Median(2,2,1) = 4 - 2 = 2$.

Meanwhile, for a pattern $P_2 = \langle u_1, u_x, u_4 \rangle$ we will have the following values: $Length(P_2) = 3$; $Frequency(P_2) = 3$; $Coverage(P_2) = 9/17 = 0.529$; $Cohesion(P_2) = 3 - Median(3,3,2) = 3 - 3 = 0$.

Based on most of the metrics (except frequency) pattern P_1 is more preferable than P_2, and therefore, it will be selected.

For the candidate routine that we identified as the best pattern for a given quality criterion, we collect and remove all its occurrences from the log. An occurrence of a candidate routine is called a *routine instance*. Formally, a routine instance is a sequence of (non-normalized) UIs, e.g., $r = \langle u_1, u_2, u_3, u_4 \rangle$. After removing all the instances of the best candidate routine from the log, we repeat this identification step until no more candidate routines are identified. After this step, we obtain a set of candidate routines, referred to as $\mathscr{C}_\Sigma$, such that, for each candidate routine $c_i \in \mathscr{C}_\Sigma$, we can retrieve the set of its routine instances, referred to as $\mathscr{R}_{c_i}$.

Considering our running example, refer to Table 8, assuming that the two routine instances that we identified in the previous step (by detecting their segment-start and segment-end UIs) frequently occur in the original log (a snapshot of which is captured in Table 6), and choosing length as a selection criterion, at the end of this step, we would discover two candidate routines, each consisting of 15 normalized UIs (as shown in Table 8). An example of a routine instance for each of the two candidate routines can be easily observed in the original log, in Table 6 rows 1 to 24 and 25 to 49 (excluding the UIs filtered in the first step of our approach).

5.2. Evaluation

We implemented our approach as an open-source Java command-line application.[2] The goal of our evaluation is threefold. First, we assess to what extent our approach can rediscover routines that are known to be recorded in the input UI

[2] Available at https://github.com/volodymyrLeno/RPM_Segmentator

logs. Second, we analyze how the use of different candidate routine selection criteria such as frequency and cohesion impact the quality of the discovered routines. Last, we assess the efficiency and effectiveness of our approach when applied to real-life UI logs. Accordingly, we define the following research questions:

- ◦ **RQ1.** Does the approach rediscover routines that are known to exist in a UI log?
- ◦ **RQ2.** How do the candidate routine selection criteria affect the quality of the discovered routines?
- ◦ **RQ3.** Is the approach applicable in real-life settings in terms of both efficiency and effectiveness?

5.2.1. Datasets

To answer our research questions, we rely on a dataset of seventeen UI logs, which can be divided into three subgroups: artificial logs, real-life logs recorded in a supervised environment, and real-life logs recorded in an unsupervised environment.[3] Table 9 shows the characteristics of the logs.

Table 9: UI logs characteristics

UI Log	# Routine Variants	# Task Traces	# UIs	# UIs per trace (avg)
CPN1	1	100	1400	14.000
CPN2	3	1000	14804	14.804
CPN3	7	1000	14583	14.583
CPN4	4	100	1400	14.000
CPN5	36	1000	8775	8.775
CPN6	2	1000	9998	9.998
CPN7	14	1500	14950	9.967
CPN8	15	1500	17582	11.721
CPN9	38	2000	28358	14.179
Student Records (SR)	2	50	1536	30.720
Reimbursement (RT)	1	50	3108	62.160
SRRT$_{+}$	3	100	4643	46.430
RTSR$_{+}$	3	100	4643	46.430
SRRT$_{\|}$	3	100	4643	46.430
RTSR$_{\|}$	3	100	4643	46.430
Scholarships 1 (S1)		-	693	
Scholarships 2 (S2)		-	509	

The artificial logs (CPN1–CPN9) were generated from Colored Petri Nets (CPNs) [52]. The CPNs have increasing complexity, from low (CPN1) to high (CPN9). These logs are originally noise-free and segmented. We removed the segment identifiers to produce unsegmented logs.

The *Student Records* (SR) and *Reimbursement* (RT) logs record the simulation of real-life scenarios. The SR log simulates the task of transferring students' data from a spreadsheet to a Web form. The RT log simulates the task of filling reimbursement requests with data provided by a claimant. Each log contains fifty

[3]The real-life logs were recorded with the Action Logger tool [22]. All the logs are available at `https://doi.org/10.6084/m9.figshare.12543587`

recordings of the corresponding task executed by following strict guidelines on how to perform the task. These logs contain little noise, which only accounts for user mistakes, such as filling the form with an incorrect value and performing additional actions to fix the mistake. We know how the underlying task was executed for both logs, and we use such information as ground truth when evaluating our approach. Additionally, we created four more logs (SRRT_+, RTSR_+, $\text{SRRT}_{||}$, $\text{RTSR}_{||}$) by combining SR and RT. SRRT_+ and RTSR_+ capture the scenario where the user first completes all the instances of one task and then moves to the other task. These logs were generated by concatenating SR and RT. $\text{SRRT}_{||}$ and $\text{RTSR}_{||}$ capture the scenario where the user is working simultaneously on two tasks. To simulate such behavior, we interleaved the segments of SR with those of RT, i.e., the first segment of SR is followed by the first segment of RTand so on.

Finally, the *Scholarships* logs (S1 and S2) were recorded by two employees of the University of Melbourne who performed the same task. The logs record the task of processing scholarship applications for international and domestic students. The task mainly consists of students' data manipulation with transfers between spreadsheets and Web pages. Compared to the other logs, we have no a-priori knowledge of how to perform the task in the *Scholarships* logs (no ground truth). Also, when recording the UI logs, the University employees were not instructed to perform their task in a specific manner, i.e., they were left free to perform the task as they would normally do when unrecorded.

5.2.2. Setup

To answer RQ1 and RQ2, we analyzed the quality of the segmentation and that of the discovered routines, using the first 15 logs described above (CPN1 to CPN9, SR, RT, SRRT_+, RTSR_+, $\text{SRRT}_{||}$, $\text{RTSR}_{||}$), against the four candidate routine selection criteria in Section 5.1.3, i.e., frequency, length, coverage, and cohesion. We discuss the results of this part of our experiment from two perspectives: the quality of the segmentation, and the quality of the discovered routines. To assess the quality of the segmentation, we use the *normalized* Levenshtein Edit Distance (LED), where a segment and its normalized UIs represent the string and its characters, respectively. Precisely, for each discovered segment, we collect all the ground truth segments that have at least one shared UI with the discovered segment, calculate the LED between the discovered segment and the ground truth segments and assign the minimum LED to the discovered segment as its quality score. Finally, we assess the overall quality of the segmentation as the average of the LEDs of each discovered segment.

One of the underlying assumptions of our approach is to provide minimal to no input for the segmentation step. Given that all existing segmentation techniques are supervised, their comparison with our approach for segmentation is out of scope of this thesis.[4]

[4]Note, that in a semi-automated context, the use of such techniques is reasonable.

The quality of the discovered routines is measured with the Jaccard Coefficient (JC), which captures the level of similarity between discovered and ground truth routines in a less strict manner compared to LED. In fact, the JC does not penalize the order of the UIs in a routine. This follows from the assumption that a routine could be executed performing some actions in a different order, and the ordering should not be penalized. The JC between two routines is the ratio $\frac{n}{m}$, where n is the number of UIs contained in both routines, while m is the total number of UIs in each of the two routines (i.e., the sum of the lengths of the two routines).

Given the set of discovered routines and the set of ground truth routines, for each discovered routine, we compute its JC with all the ground truth routines and assign the maximum JC to the discovered routine as its quality score. Finally, we assess the overall quality of the discovered routines as the average of the JC of each discovered routine. As the ground truth, we used the segments of the artificial logs and the guidelines used to perform the tasks in SR and RT.

However, we cannot rely on the JC alone to assess the quality of the discovered routines, as this measure does not consider the routines we may have missed in the discovery. Thus, we also measure the total coverage to quantify how much log behavior is captured by the discovered routines. We would like to reach high coverage with as few routines as possible. Thus, we prioritize long routines over short ones by measuring the average routine length alongside coverage.

To answer RQ3, we tested our approach on the S1 and S2 logs. Given that the ground truth for these logs is not available and therefore, the quantitative metrics cannot be computed, we performed a qualitative assessment of the results with the help of the employees who performed the task. Specifically, we asked them to compare the rediscovered routines and the actions (i.e., UIs) they performed while recording.

All experiments were conducted on a Windows 10 laptop with an Intel Core i5-5200U CPU 2.20 GHz and 16GB RAM with the minimum support threshold set to 0.1 and the minimum coverage threshold equal to 0.05.

5.2.3. Results

Table 10 shows the results of the segmentation. As we can see, the LED for all the CPN logs is 0.0, highlighting that all the segments were discovered correctly. On the other hand, the segments discovered from the SR, RT, SRRT_+, RTSR_+, $\text{SRRT}_{||}$, and $\text{RTSR}_{||}$ logs slightly differ from the original ones. The main difference between the CPN logs and those recorded in a controlled environment is that the former contain routines always having the same starting UI, while the latter contain routines with several different starting UIs. As a result, we identified the correct number of segments from all the logs except $\text{SRRT}_{||}$ and $\text{RTSR}_{||}$, where we could not discern the ending UI of the routine belonging to one task and the starting UI of the routine belonging to the other task, consequently merging the two routines and discovering only half of the total number of segments (50 out of

100). From the table, we can also see that the time performance of our approach is reasonable, with a maximum execution time of 3.6 seconds.

Table 10: Segmentation results

UI Log	# UIs	# Original Segments	# Discovered Segments	LED (avg)	Execution Time (sec)
CPN1	1400	100	100	0.000	0.571
CPN2	14804	1000	1000	0.000	1.705
CPN3	14583	1000	1000	0.000	0.835
CPN4	1400	100	100	0.000	0.461
CPN5	8775	1000	1000	0.000	1.025
CPN6	9998	1000	1000	0.000	0.707
CPN7	14950	1500	1500	0.000	1.566
CPN8	17582	1500	1500	0.000	1.596
CPN9	28358	2000	2000	0.000	3.649
SR	1536	50	50	0.059	0.535
RT	3107	50	50	0.096	1.507
SRRT_{+}	4643	100	100	0.077	1.707
RTSR_{+}	4643	100	100	0.077	1.560
$\text{SRRT}_{\|\|}$	4643	100	50	0.331	2.166
$\text{RTSR}_{\|\|}$	4643	100	50	0.331	2.177

Table 11 shows the quality of the discovered routines for each selection criterion when setting 0.1 as the minimum support threshold of CloFast. Overall, the results highlight that the routines with the highest JC and the longest length are those discovered using cohesion as a selection criterion, followed closely by those discovered using length as the criterion. Even though we do not always achieve the highest coverage using these criteria, the coverage scores are very high, above 0.90 for all the logs except CPN5.

Following these results, we decided to use cohesion as the selection criterion to discover the routines from the *scholarships* logs. From the S1 log, we discovered five routine variants. The first routine variant consists in manually adding graduate research student applications to the student record in the university's information system. The application is then assessed, and the student is notified of the outcome. The second routine variant consists in lodging a ticket to verify possible duplicate applications. When a new application is entered into the information system and its data matches an existing application, the new application is temporarily put on hold. The employee fills in and lodges a ticket to investigate the duplicate. The remaining three routine variants represent exceptional cases, where the employee executed either the first or the second variant in a different manner (i.e., by altering the order of the actions or overlapping routines executions). To assess the results, we showed the discovered routine variant to the employee of the University of Melbourne who recorded the S1 log, and they confirmed that the discovered routines correctly capture their task executions. Also, they confirmed that the last three routine variants are alternative executions of the first routine variant.[5]

While the results from the S1 log were positive, our approach could not dis-

[5]Detailed results at `https://doi.org/10.6084/m9.figshare.12543587`

cover any correct routine from the S2 log. By analyzing the results, we found out that the employee worked with multiple worksheets at the same time, frequently switching between them for visualization purposes only. Such behavior recorded in the log negatively affects the construction of the CFG and its domination tree, ultimately leading to the discovery of incorrect segments and routines. This also impacted the execution time; indeed, while it took only 41.7 seconds to discover the routines from the S1 log, it took 426.3 seconds to discover the routines from the S2 log.

5.2.4. Threats to validity

The reported evaluation has a number of threats to validity. First, a potential threat to internal validity is the fact that the context parameters (i.e., the attributes in the log that capture the notion of "user interaction") were manually selected. These context parameters are required as one of the inputs of the proposed method (in addition to the UI log). To mitigate this threat, we selected the parameters independently and then cross-checked them to reach a mutual agreement. Another possible threat to internal validity is the limited use of parameter values to configure the approach at hand. To ensure we do not miss any significantly important behavior in the logs, we used very low support and coverage, equal to 0.1 and 0.05, respectively.

A potential threat to external validity is given by the use of a limited number of real-life logs (four). These logs focus on one type of task that can be automated via RPA, namely data transferring. These logs, however, exhibit different characteristics in terms of the complexity of the captured processes and log size. To mitigate this threat, we additionally performed a more extensive evaluation on a battery of artificial logs. For the two real-life logs, we had no information about the underlying processes. Therefore, we evaluated the results qualitatively with the workers responsible for their execution. To ensure the full reproducibility of the results, we have released all the logs, both real-life and artificial, used in our experiments. The only exceptions are S1 and S2 as they contain sensitive information.

5.3. Summary

This chapter addressed **RQ1** posed in the thesis[6], presenting an approach to discover candidate routines for automation from UI logs. The approach starts by decomposing the UI log into segments corresponding to paths within the Control-Flow Graph's connected components derived from the log. Once the log is segmented, a noise-resilient sequential pattern mining technique is used to extract

[6]*Given a user interaction log, how to identify the routines that can be potentially automated via an RPA tool?*

Table 11: Quality of the discovered routines

UI Logs	Selection Criterion	# Discovered Routines	Routine Length	Total Coverage	JC	Execution Time (sec)
CPN1	Frequency	1	14.00	1.00	1.000	2.643
	Length	1	14.00	1.00	1.000	1.553
	Coverage	1	14.00	1.00	1.000	3.702
	Cohesion	1	14.00	1.00	1.000	**1.530**
CPN2	Frequency	3	6.33	**0.99**	0.452	3.908
	Length	2	**14.50**	0.95	**1.000**	4.789
	Coverage	2	14.00	**0.99**	0.964	**3.166**
	Cohesion	2	**14.50**	0.95	**1.000**	3.730
CPN3	Frequency	4	5.75	0.95	0.511	4.682
	Length	3	**14.33**	0.93	**1.000**	4.324
	Coverage	3	9.67	**0.96**	0.833	**3.940**
	Cohesion	3	**14.33**	0.93	**1.000**	6.237
CPN4	Frequency	1	12.00	0.86	0.857	3.512
	Length	4	**14.00**	**1.00**	**1.000**	**3.118**
	Coverage	1	12.00	0.86	0.857	3.470
	Cohesion	4	**14.00**	**1.00**	**1.000**	3.799
CPN5	Frequency	6	1.67	**0.86**	0.206	**6.418**
	Length	7	7.29	0.83	0.849	9.715
	Coverage	4	3.75	0.80	0.462	6.587
	Cohesion	8	**7.5**	**0.86**	**0.910**	18.206
CPN6	Frequency	3	4.67	1.00	0.485	4.250
	Length	2	**10.00**	1.00	**1.000**	2.924
	Coverage	3	4.67	1.00	0.485	**2.483**
	Cohesion	2	**10.00**	1.00	**1.000**	4.678
CPN7	Frequency	7	2.43	0.91	0.257	10.118
	Length	7	**9.57**	0.88	**0.986**	8.957
	Coverage	6	3.67	0.91	0.385	**7.203**
	Cohesion	7	9.43	**0.93**	0.971	11.983
CPN8	Frequency	5	4.20	0.75	0.337	11.801
	Length	6	**10.67**	**0.91**	**0.967**	9.070
	Coverage	5	7.60	0.89	0.618	**7.354**
	Cohesion	6	**10.67**	**0.91**	**0.967**	11.250
CPN9	Frequency	5	5.20	0.82	0.401	13.784
	Length	6	**14.67**	**0.95**	**1.000**	**8.265**
	Coverage	5	6.60	0.88	0.511	8.603
	Cohesion	6	**14.67**	**0.95**	**1.000**	13.943
SR	Frequency	2	15.00	0.96	0.533	2.042
	Length	2	**30.00**	0.92	**0.967**	3.547
	Coverage	2	15.50	**0.98**	0.532	**2.001**
	Cohesion	2	**30.00**	0.92	**0.967**	3.621
RT	Frequency	3	19.00	**0.91**	0.295	3.565
	Length	2	**61.00**	0.90	**0.875**	4.225
	Coverage	1	58.00	0.86	**0.857**	**3.408**
	Cohesion	2	**61.00**	0.90	**0.875**	4.765
SRRT$_+$	Frequency	4	21.25	0.91	0.472	6.771
	Length	4	**45.25**	**0.93**	**0.929**	**5.201**
	Coverage	2	43.00	0.88	**0.929**	5.207
	Cohesion	4	**45.25**	**0.93**	**0.929**	6.885
RTSR$_+$	Frequency	4	21.25	0.91	0.472	7.193
	Length	4	**45.25**	**0.93**	**0.929**	5.581
	Coverage	2	42.50	0.88	**0.929**	**5.269**
	Cohesion	4	**45.25**	**0.93**	**0.929**	7.199
SRRT$_\parallel$	Frequency	3	28.33	0.91	0.228	**7.436**
	Length	4	**90.25**	**0.92**	**0.595**	14.595
	Coverage	1	86.00	0.85	0.593	7.818
	Cohesion	4	**90.25**	**0.92**	**0.595**	17.935
RTSR$_\parallel$	Frequency	3	28.33	0.91	0.228	7.211
	Length	4	**90.25**	**0.92**	**0.595**	14.194
	Coverage	1	86.00	0.85	0.593	**7.22**
	Cohesion	4	**90.25**	**0.92**	**0.595**	17.858

frequent routines. The routines are then ranked according to four quality criteria: frequency, length, coverage, and cohesion.

The approach has been implemented as an open-source tool and evaluated using synthetic and real-life logs. The evaluation shows that the approach can rediscover routines injected into a synthetic log and that it discovers relevant routines in real-life logs. The execution times range from seconds to a few dozen seconds, even for logs with tens of thousands of interactions.

The proposed approach makes a number of limiting assumptions. First, it relies on the information recorded in the log to identify segments and discover routines. Thus, its effectiveness is correlated with data quality. Since the UI log is fine-grained, deviations occurring during the routine execution affect the effectiveness of our approach. In our evaluation, we observed this phenomenon to varying degrees when dealing with real-life logs. In practice, the approach can identify correct routines only if they are observed frequently in the UI log. Recurring noise affects the accuracy of the results (see the S2 log).

Second, the approach is designed for logs that capture consecutive routine executions. In practice, routine instances may sometimes overlap (cf. the S2 real-life log in the evaluation). A possible avenue to address this limitation is to search for overlapping frequent patterns directly in the unsegmented log instead of first segmenting it and then finding patterns in the segmented log. This approach has been previously investigated in the context of so-called Local Process Mining (LPM), where the goal is to discover process models capturing frequently repeated (and possibly overlapping) behavior in an unsegmented sequence of events [66].

Finally, while our approach is robust against routine executions with multiple ends, it is sensitive to multiple starts. Ideally, all routine executions should start with the same UI unless different starts are recorded in batch (e.g., first only routines with a start, then routines with another start, etc.). In general, our approach can handle logs containing multiple different routines, provided that each routine does not share any UIs with other routines, except their start UIs.

At the current stage, we have no information on the automatability of identified candidate routines. In the next chapter, we present an approach to assess the candidates' amenability for automation and synthesize executable specifications for the automatable routines.

6. DISCOVERY OF EXECUTABLE ROUTINES

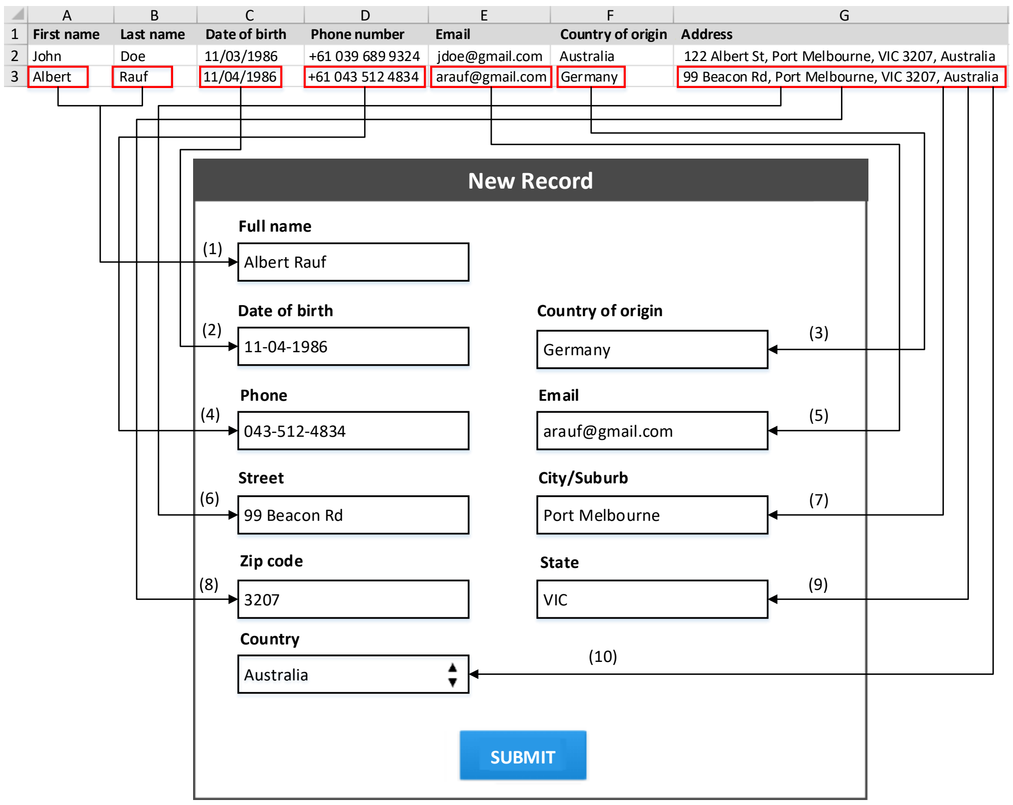
In the previous chapter, we presented an approach to identify candidate routines for automation. This chapter[1] focuses on the discovery of executable specifications of such routines. Specifically, we present two approaches to obtain such specifications. In Section 6.1, we present an approach that maps the problem of discovering executable routines to the problem of discovering data transformations and produces routine specifications in the form of data transformation programs. In Section 6.2, we adapt the approach from Section 6.1 to assess the degree of automatability of candidate routines and to discover more flexible executable specifications. Next, in Section 6.3, we present a method to identify semantically equivalent routines to produce a non-redundant set of automatable routines. Section 6.4 reports the empirical evaluation of the proposed approaches, and Section 6.5 concludes the chapter.

6.1. Global transformations approach

One of the recurrent use cases for RPA bots is to automate data transfers across multiple applications, especially when these applications do not provide suitable APIs to enable their programmatic integration. Figure 14 illustrates an example of a data transferring task, where student records (e.g., name, surname, phone, email, and address) from an Excel spreadsheet have to be transferred to the data-entry Web form of a students management system, as part of the student admission process at the university. For each row in the spreadsheet (representing a student), a student admission staff member manually copies every cell in that row and pastes that into the corresponding text field in the Web-form. Once the data transfer for a given student has been completed, the staff member presses the "Submit" button to submit the data into the students management system, and repeats the procedure for the next row. Such data transferring task often involves data transformations as source and target data can be in a different format (e.g., spreadsheet and web form have different formats for the date of birth). Therefore, she has to copy data from a certain cell, paste it into the corresponding field and do some manual manipulations to convert it into the required format.

The transferring of data from one application to another can be seen as a data transformation process, where the data from the source application has to be transformed into the data in the target application. Indeed, the final effect of a data transferring task is that one or more records (with an implicit schema) are mapped to one or more records with the same or a different schema. For each data element of the target in the example presented above, Figure 14 shows the mapping to the corresponding data elements from the source. Such manipulations can be automated via an RPA bot so long as they are described via a transformation pro-

[1] Corresponding to [67] and [63]



	A	B	C	D	E	F	G
1	First name	Last name	Date of birth	Phone number	Email	Country of origin	Address
2	John	Doe	11/03/1986	+61 039 689 9324	jdoe@gmail.com	Australia	122 Albert St, Port Melbourne, VIC 3207, Australia
3	Albert	Rauf	11/04/1986	+61 043 512 4834	arauf@gmail.com	Germany	99 Beacon Rd, Port Melbourne, VIC 3207, Australia

New Record

Full name	Albert Rauf
Date of birth	11-04-1986
Country of origin	Germany
Phone	043-512-4834
Email	arauf@gmail.com
Street	99 Beacon Rd
City/Suburb	Port Melbourne
Zip code	3207
State	VIC
Country	Australia

Figure 14: Routine as a transformation

gram. Hence, instead of discovering repetitive sequences of actions to transfer data across applications, we propose to discover the transformations that the users effectively perform when executing these sequences of actions.

Depending on the number of source and target elements involved in a transformation, transformations can be of four different types. *One-to-one* (1 - 1) transformations involve one source and one target element. Usually, they are represented by manipulations that correspond to characters replacement (#2 in Figure 14) or partial copying (#4). When the data is already in the target format, no manipulations are required and the user has to perform a simple copy-paste operation (#3, #5). The next transformation type is *one-to-many* (1 - N). This type usually describes manipulations that require a split operation. In the example depicted in Figure 14, the user has to split the full address from the spreadsheet into five different elements: street, city, ZIP code, region and country (#6-#10). Next, *many-to-one* (N - 1) transformations require a merge operation over several elements. In our example, cells from the spreadsheet that contain the student's first and last names have to be joined to fill in the text field that corresponds to the full name in the Web form (#1). Finally, *many-to-many* (N - N) transformations involve manipulations such as copying and pasting a range of cells, pivoting a row into a column in a spreadsheet, etc. These are not shown in Figure 14.

The underlying transformations can be learned from the examples present in UI logs. All the UIs representing a read operation (e.g., copy a cell) provide us with the information about the data stored in a source application, while all the edit UIs describe how it is saved in a target application. Given a set of such input and output examples, we can synthesize a data transformation program to convert raw input data into the format required by the target application. Figure 15 presents a general approach to discover executable routines in the form of data transformation programs from UI logs.

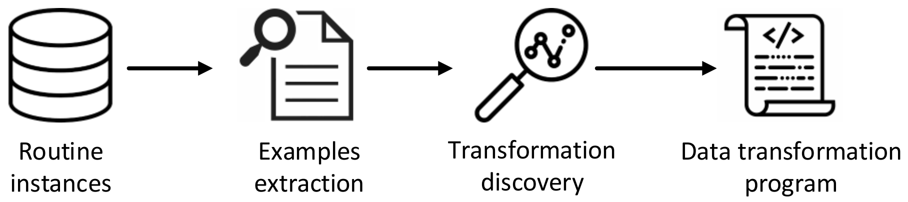


Figure 15: Baseline approach for discovering data transformations in UI logs

Given a collection of routine instances of a particular routine, the approach extracts a set of transformation examples that are then used for discovering data transformations. The approach is described in Section 6.1.1, and we refer to it as a baseline approach. In Sections 6.1.2 and 6.1.3, we will present two optimizations that can improve its efficiency and to address its limitations.

6.1.1. Baseline Approach

Given a collection of routine instances, the first step of our baseline approach for data transformation discovery is to extract *transformation examples*. A transformation example is a tuple (I, O, S, T), where I is the raw data from a source document S, and O is the data in a final format stored in a target document T. A document is an instance of an application used to accomplish a task (e.g., a spreadsheet or a Web form). For each routine instance of a given candidate routine, we extract all the values used in the source document S and the target document T. All these values are then used to build a transformation example. Considering the example presented in Figure 14, the collection of target values is:

O = [“Albert Rauf”,“11-04-1986”, “Germany”, “043-512-4834”, “arauf@gmail.com”, “99 Beacon Rd”, “Port Melbourne”, “VIC”, “3207”, “Australia”],

while the corresponding collection of source values is:

I = [“Albert”, “Rauf”, “11/04/1986”, “+61 043 512 4834”, “arauf@gmail.com”, “Germany”, “99 Beacon Rd, Port Melbourne, VIC 3207, Australia”].

After all transformation examples are extracted, they are provided as input to a transformation discovery technique. Specifically, we use a state-of-the-art data transformation-by-example discovery technique, called Foofah [58]. Given transformation examples summarized in the form of input and output tables, this technique aims at synthesizing an optimal transformation program that describes what manipulations have to be performed to convert raw input data into the required final format. Foofah describes program synthesis as a search problem in a state-space graph, where each edge represents a modification operation (e.g., split, removal, etc.), start and end nodes describe raw and final data, and all other nodes denote an intermediate value during the transformation process (see Figure 16). It then uses a heuristic search approach based on the A* algorithm to find an optimal path from the initial state to the goal state. The cost function is defined as the minimum number of manipulations required to transform one value into another. Given the two possible transformation functions to change the format of phone numbers depicted in Figure 16, the top transformation function will be selected and retrieved by Foofah because it is characterized by the lowest cost (i.e., requires only two manipulations over input data, in contrast, the bottom transformation function involves four manipulations). Foofah can handle both string and tabular manipulations, supporting all four different transformations presented earlier.[2]. It supports the Potter's Wheel operations [68] and is extensible. We assume that the output is noise- and error-free, meaning that the analyzed data transformations are correct.

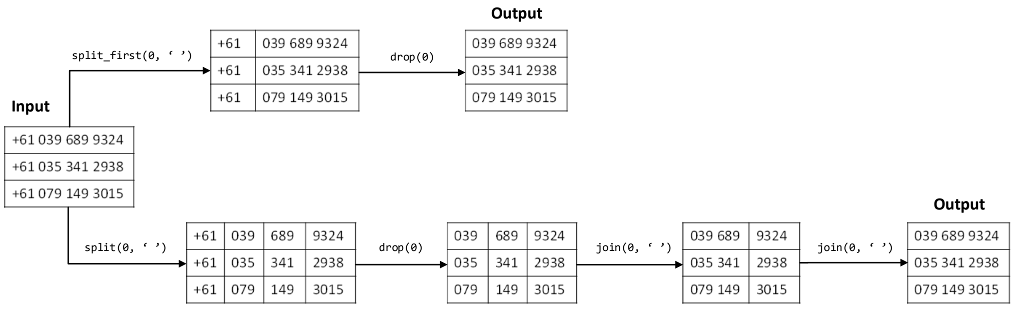


Figure 16: Example of Foofah execution

Figure 17 presents a sample transformation program that may be discovered by Foofah for the example depicted in Figure 14. The program takes table *I* as input and produces table *O* as output. The intermediate results of operations are stored in variable *t*. The sample transformation program in Figure 17 consists of join and split commands. For instance, the `f_join_char` command at line 1 of Figure 17 merges first name and last name into full name and places one space character (cf. the third parameter) between them. The second parameter equal to 0 tells that the element at the first position in table *I* (*"Albert"*) should be joined with its immediate successor (*"Rauf"*); note that we count positions from zero and join two subsequent elements. The `f_split` command at line 2 divides the date into

[2]For details on Foofah's implementation, we refer the reader to [58]

a day, month, and year components using the ‘/’ symbol; note that the command takes the result of line 1, i.e., table `t`, as input, which in the second position, referenced by the value of the second parameter, contains the string *“11/04/1986”*. The main commands in Foofah’s language are provided in Table 12.

```
1  t = f_join_char(I, 0, ‘ ’)      8  t = f_join_char(t, 2, ‘-’)
2  t = f_split(t, 1, ‘/’)          9  t = f_split(t, 5, ‘,’)
3  t = f_join_char(t, 2, ‘-’)     10  t = f_split_w(t, 7)
4  t = f_join_char(t, 1, ‘-’)     11  t = f_split_w(t, 9)
5  t = f_split(t, 2, ‘ ’)         12  t = f_split_w(t, 6)
6  t = f_drop(t, 2)               13  O = f_join_char(t, 6, ‘ ’)
7  t = f_join_char(t, 3, ‘-’)
```

Figure 17: A sample transformation program

Table 12: List of the most commonly used Foofah’s commands

No.	Command	Description
1	f_join_char(t, i, c)	Join columns *i* and *i*+1 of table *t* with character *c* in between
2	f_join(t, i)	Join columns *i* and *i*+1 of table *t* with no separator in between
3	f_split(t, i, c)	Split column *i* of table *t* by character *c*
4	f_split_first(t, i, c)	Split column *i* of table *t* by the first occurence of character *c*
5	f_split_w(t, i)	Split column *i* of table *t* into words (split by space character)
6	f_split_tab(t, i)	Split column *i* of table *t* by tabulation symbol
7	f_move(t, i, j)	Put column *i* before column *j* in table *t*
8	f_move_to_end(t, i)	Put column *i* at the end of table *t*
9	f_move_from_end(t, i)	Put the last column before column *i* in table *t*
10	f_wrap(t, i)	Concatenate all rows of column *i* into one row
11	f_wrap_every_k_rows(t)	Concatenate every *k* rows
12	f_wrap_one_row(t)	Concatenate all rows into one
13	f_fold(t, i)	Collapse all columns after column *i*
14	f_unfold(t)	Unfold all rows on last column
15	f_fill(t, i)	Fill empty cells in column *i* with the values from the upmost cell
16	f_delete(t, i)	Remove all empty cells in column *i*
17	f_delete_empty_cols(t)	Remove all empty columns in table *t*
18	f_transpose(t)	Transpose table *t*
19	f_extract(t, i, regex)	Extract first substring in column *i* that satisfies given *regex* pattern
20	f_drop(t, i)	Remove column *i* from table *t*

Nonetheless, Foofah suffers from scalability issues when the number of the transformation examples is large or when the target transformation is not trivial (i.e., cannot be described using a single join or split command). Furthermore, in some cases, Foofah may fail to discover a transformation program even if it exists, which is caused by its inability to deal with heterogeneous data and handle ambiguity.

The time complexity of Foofah technique is $O((kmn)^d)$, where m and n are, respectively, the number of cells in the input and output tables, k is the number of candidate data operations for each intermediate transformation result, and d is the number of components in the synthesized program [58]. The number of cells in the input and output tables can be calculated as the total number of columns and rows in such tables. In our context, columns of the table correspond to the fields of the document that were read or edited, while rows represent transformation examples given as input to Foofah. Next, we present two optimizations of the baseline approach that aim at reducing both the number of data fields (horizontal optimization) and transformation examples (vertical optimization) provided to Foofah for processing, thus improving the overall performance.

6.1.2. Optimization 1: Grouping Examples by Target

Document-to-document transformations usually involve complex manipulations. Thus, it takes a significant amount of time to synthesize a corresponding transformation program, and often Foofah fails to discover any transformation. We propose an optimization that aims at decomposing transformations from document-to-document to source-to-target level. This is done by projecting transformation examples onto target elements within a target document using the information about data elements involved in task executions available in the UI log. For instance, the transformation example with input table I and output table O from Section 6.1.1 projected onto the target text field *Full name* results in the transformation example with input I' = *["Albert", "Rauf"]* and output O' = *["Albert Rauf"]*. For the obtained transformation example, it is easy to find the corresponding transformation of join by space operation.

Algorithm 4 shows the procedure for extracting transformation examples considering this optimization. For each routine instance r, we iterate over its UIs backwards starting from the last UI. For each UI (u_1) of type *edit*, we extract the target element from its payload t_1 (line 10) that can be the ID of a web browser element or the location of a cell in a spreadsheet. We then check whether this target element had already been modified (line 11). If it was edited previously, we ignore the corresponding UI and keep iterating. Otherwise, we add the target element t_1 to a set of edited elements F (line 12), and read the payload of u_1 to retrieve the value of the target element after the editing (O, line 13). We initialize two queues, S (which stands for *sources*) and I (which stands for *inputs*). Queue S stores the ID or location of the (source) element(s) that produced the data used by the *edit* UI instance u_1; while queue I stores the data used by the *edit* UI instance u_1. After this initialization, we iterate over all the UIs preceding u_1 in r (lines 16 to 27). Such iteration goes backward from u_1 to the first UI in r. For each encountered u_2 of type *paste* (line 18), we check its target element and we compare it to t_1 (line 20). If they are the same, we again traverse backward the routine instance from the *paste* UI until we find a *copy* UI u_3 (lines 21 to 27).[3]. Then, we retrieve the target element of u_3, we add it to the queue S, and we add the copied value of u_3 to the queue I (lines 25 and 26). When we reach this point, we also stop searching for *copy* UIs for the current *paste* u_2, as we assume that a clipboard may contain only one value at a time. After we found all the UIs that contributed to the final value of u_1, we create a transformation example (I, O, S, t_1) and add it to set T (line 28). Next, we continue searching for UIs whose target elements are

[3]Our filtering approach, described in Section 5.1.2 guarantees that there exists a u_3 of type *copy* preceding the *paste* UI

Algorithm 4: Transformation examples extraction

```
input  : Routine Instances Set $\mathscr{R}_{c_i}$
output : Transformation Examples $T$
1  Set $C \leftarrow$ { “copy cell”, “copy range”, “copy field” };
2  Set $P \leftarrow$ { “paste into cell”, “paste into range”, “paste” };
3  Set $T \leftarrow \varnothing$;
4  foreach $r \in \mathscr{R}_{c_i}$ do
5      Set $F \leftarrow \varnothing$;
6      Integer $n \leftarrow$ getLength($r$);
7      for $i \leftarrow n$ to $1$ do
8          UI $u_1 \leftarrow$ get(r, i);
9          if $getType(u_1) \in E$ then
10             $t_1 \leftarrow$ getTargetElement($u_1$);
11             if $t_1 \notin F$ then
12                 $F \leftarrow F \cup \{t_1\}$;
13                 $O \leftarrow$ getParameterValue($u_1$, “Value”);
14                 Queue $S \leftarrow \varnothing$;
15                 Queue $I \leftarrow \varnothing$;
16                 for $j \leftarrow i-1$ to $1$ do
17                     $u_2 \leftarrow$ get(r, j);
18                     if $getType(u_2) \in P$ then
19                         $t_2 \leftarrow$ getTargetElement($u_2$);
20                         if $t_2 = t_1$ then
21                             for $k \leftarrow j-1$ to $1$ do
22                                 $u_3 \leftarrow$ get(r,k);
23                                 if $getType(u_3) \in C$ then
24                                     $s \leftarrow$ getTargetElement($u_3$);
25                                     push $s$ to $S$;
26                                     push getParameterValue($u_3$, “Value”) to $I$;
27                                     break
28                 $T \leftarrow T \cup \{(I,O,S,t_1)\}$;
29 return $T$
```

not yet marked as already modified and repeat the procedure described above.

After obtaining a set of transformation examples, the approach splits it into smaller groups based on targets. For each obtained group, we then discover a separate transformation program using Foofah. Figure 18 presents the transformations discovered using this approach.

As can be seen clearly, these transformations are concise and clear. They are also discovered much faster in comparison to the baseline approach.

6.1.3. Optimization 2: Grouping Examples by Input Structure

The first optimization seeks to reduce the number of input fields that Foofah needs to consider, but it does not reduce the number of transformation examples given as input. Thus, when a complex transformation occurs between one or multiple source elements and one target element, Foofah may still fail to synthesize a transformation program. Accordingly, the number of transformation examples used to synthesize the transformation program has to be reduced. On the other hand, to discover a correct transformation program many examples that capture

Columns A,B → Full_Name

```
O1 = f_join_char(I1, 0, ‘ ’)
```

Column D → Phone_Number

```
t = f_split(I2, 0, ‘ ’)
t = f_drop(t, 0)
t = f_join_char(t, 1, ‘-’)
O2 = f_join_char(t, 0, ‘-’)
```

Column G → Address_Street

```
t = f_split_first(I3, 0, ‘,’)
o3 = f_drop(t, 1)
```

Column G → Adress_ZipCode

```
t = f_split_first(I4, 0, ‘,’)
t = f_drop(t, 0)
t = f_extract(t, 0, ‘\d+’)
o4 = f_drop(t, 0)
```

Figure 18: Transformation programs synthesized by the approach that groups transformation examples by targets

different cases and behaviors are required. Therefore, it is crucial to preserve all the behavior during the reduction process.

There are further limitations of Foofah that are not addressed by the improvement presented in Section 6.1.2. For example, Foofah does not discover conditional transformations, where different manipulations are applied depending on the input. Consequently, it cannot deal with heterogeneous data. When the data comes from different sources, it can be stored in different formats, and Foofah will fail to discover a transformation program. Another limitation of Foofah is that it can discover transformations only if the output is not ambiguous, i.e., it is clear from which components it was obtained. Consider the example in Table 13, where the user needs to extract a ZIP code from a full address. While for Example 3 it is clear that ZIP code “3205” was obtained from Address 3 (“396 Clarendon St, South Melbourne, VIC 3205, Australia”), the origin of ZIP code “3207” is ambiguous as it can be obtained from Address 1 (“122 Albert St, Port Melbourne, VIC 3207, Australia”) or Address 2 (“99 Beacon Rd, Port Melbourne, VIC 3207, Australia”). In this case, Foofah will not discover any transformation.

Table 13: Transformation example with ambiguous output

No.	**Input**	**Output**
1	122 Albert St, Port Melbourne, VIC 3207, Australia	3207
2	99 Beacon Rd, Port Melbourne, VIC 3207, Australia	3207
3	396 Clarendon St, South Melbourne, VIC 3205, Australia	3205

To address these limitations, we group the data transformation examples into equivalence classes, where each class represents a different structural pattern of the input data. To create these equivalence classes, for each data sample in the input data series, we discover its symbolic representation describing its structural pattern by applying *tokenization*. The tokenization that we apply replaces each maximal chained subsequence of symbols of the same type (either digits or letters) with a special token character ($\langle d\rangle$+ or $\langle a\rangle$+, resp.), and leaves any other symbol unaltered. Below, we show how a raw address value from our running example is tokenized:

s = “99 Beacon Rd, Port Melbourne, VIC 3207, Australia”

1) Replace all alphabetic characters with tokens:

s = "99 $\langle a \rangle$+ $\langle a \rangle$+, $\langle a \rangle$+ $\langle a \rangle$+, $\langle a \rangle$+ 3207, $\langle a \rangle$+"

2) Replace all digits with tokens:

s = "$\langle d \rangle$+ $\langle a \rangle$+ $\langle a \rangle$+, $\langle a \rangle$+ $\langle a \rangle$+, $\langle a \rangle$+ $\langle d \rangle$+, $\langle a \rangle$+"

Hence, the resulting pattern is "$\langle d \rangle$+ $\langle a \rangle$+ $\langle a \rangle$+, $\langle a \rangle$+ $\langle a \rangle$+, $\langle a \rangle$+ $\langle d \rangle$+, $\langle a \rangle$+". Note that all the space characters as well as punctuation are preserved in the pattern.

We discover a data transformation function for each equivalence class by providing to Foofah one randomly selected data transformation example from the equivalence class. The use of equivalence classes allows us to remove the heterogeneity of the input data and to facilitate the application of Foofah, which will operate only on a single data transformation example.[4] Sometimes, several equivalence classes may be characterized by the same transformation program. If this occurs, the corresponding equivalence classes are merged. The transformation programs synthesized by Foofah for our running example using this optimization are shown in Figure 19.

6.2. Local transformations approach

Despite discovering executable specifications, the approach proposed in Section 6.1 does not provide any information about the non-automatable steps within a routine. Thus, it is difficult to identify the relative position of the automatable steps, and when should the discovered specification be executed. Moreover, it allows only partial automation as the activities unrelated to the editing of the fields are ignored.

In this section, we propose an approach to discover sequential routine specifications containing the information about all activities performed during a routine, specify what data they take as input and produce as output, and how to obtain it (e.g., by applying data transformation). Accordingly, the specification discovered by this approach is a pair (c, Λ), where c is a sequence of UIs, or a *candidate routine*, and Λ is a set of *data transformation steps*. Each *data transformation step* is a triplet that specifies: i) variables from which the data was read, ii) variables to which the data was written, and iii) a function capturing the data transformation (if any occurs). Such routine specifications can be compiled into software bots that can be deployed on a tool like UiPath, [5] which would be able to replicate the routine automatically. The approach takes as input a collection of candidate routines and produces a set of their executable specifications.

[4]It may happen that the synthesized transformation does not generalize to all examples in a group, e.g., due to ambiguity in the mapping from inputs to outputs. Hence, we check that the transformation discovered from one example fits all examples in the group. If it does not, we try to discover a transformation from all examples in the group. If the latter fails, the discovery procedure fails.

[5]A commercial tool available at www.uipath.com

Column G → Address_City

For pattern "<d>+ <a>+ <a>+, <a>+ <a>+, <a>+ <d>+, <a>+":

```
t = f_split_first(I_5, 0, ‘,’)
t = f_drop(t, 0)
t = f_split_first(t, 0, ‘,’)
t = f_split_w(t, 0)
t = f_join_char(t, 0, ‘ ’)
O_5 = f_drop(t, 1)
```

For pattern "<d>+ <a>+ <a>+ <a>+, <a>+ <a>+, <a>+ <d>+, <a>+":

```
t = f_split(I_6, 0, ‘,’)
t = f_drop(t, 0)
t = f_split_w(t, 0)
t = f_join_char(t, 0, ‘ ’)
t = f_join_char(t, 1, ‘ ’)
O_6 = f_drop(t, 1)
```

Otherwise:

```
t = f_split_first(I_7, 0, ‘,’)
t = f_drop(t, 0)
t = f_extract(t, 0, ‘[A-Za-z0-9\ ]+’)
t = f_drop(t, 0)
O_7 = f_split_w(t, 0)
```

Figure 19: Transformation discovered by combining both optimizations

The candidate routines (and their instances) identified by the approach in Chapter 5 represent behavior recorded in the log that frequently repeats itself, thus being a candidate for automation. However, the fact that a routine is frequently observed in a log is not sufficient to guarantee its automatability. Consider the following example; a worker fills in and submits 100 times the same web form, doing it always with the same sequence of actions but inputting manually-generated data (e.g., received over a phone call or copied from a hard-copy document). In such a scenario, although we would identify the filling and submission of the web form as a candidate routine, we would not be able to automate it because we cannot automatically generate the data in input to the web forms. On the other hand, if the data in the input to the web forms was copied from another digital document, for example, a spreadsheet, we could probably automate the routine.

Accordingly, for each candidate routine capturing the frequently observed sequence of UIs, we have to assess the degree of its automatability. To do so, we check whether all UIs of the routine are deterministic. We consider a UI to be deterministic if a software robot can replicate its execution. This is possible when: i) the input data of a UI can be determined automatically, or ii) the input data of a UI can be provided as input by the user when deploying the software robot. According to such constraints, we can provide the following rules to check whether a UI is deterministic or not.

1. UIs belonging to the *navigation* group (see Table 4) are always deterministic because they do not take in input any data; except the *select cell*, *select field*, and *select range* UIs which are removed during the filtering of the log (as described in Section 5.1.2);
2. UIs belonging to the *read* group are always deterministic because the only input they require is the source of the copied content (e.g., row and column of a cell), which is either constant or can be inputted by the user when deploying the software robot in an RPA tool;
3. UIs belonging to the *write* group that are of type *click* are always deterministic because they do not take in input any data, except the information regarding the element to be clicked which is always constant for a given candidate routine (by construction);
4. UIs belonging to the *write* group that are of type *paste* are always deterministic because they always retrieve data from the same source (i.e., the system clipboard).
5. UIs belonging to the *write* group that are of type *edit* are the only ones that are not always deterministic. In fact, these UIs are deterministic only if it is possible to determine the updated value of the edited elements (e.g., the value of a cell in a spreadsheet or of a text field in the web browser after the UI is executed). Furthermore, it has also to be possible to determine the target of the editing, although this is usually constant (if a web element) or can be inputted by the user when deploying the software robot in UiPath.

Algorithm 5 shows how we check these five rules given as input a candidate routine c_i and its routine instances $\mathscr{R}_{c_i}$, and how we compose the corresponding routine specification of the input c_i. The algorithm starts by initializing the set E as a collection of *edit* UI types (*edit cell*, *edit range*, *edit field*). Then, it iterates over all the normalized UIs in the input c_i by checking their types. If the type of a normalized UI $\bar{u}$ is not in E (line 6), i.e., one of the rules 1 to 4 applies, we add it to the queue D, which stores all the deterministic UIs we identified. Otherwise, rule 5 applies. While rules 1 to 4 are simple checks on the UI types, the complexity of rule 5 required us to operationalize it through a separate algorithm, i.e., Algorithm 6, which is called within Algorithm 5 (line 8). Algorithm 6 returns a pair (d, λ), where d is a *boolean* (true if the input normalized UI is deterministic), and λ is a *data transformation step* required to automate $\bar{u}$ and therefore available only if $\bar{u}$ is deterministic. Once all the normalized UIs in the input c_i have been checked, Algorithm 5 outputs the *routine specification* of c_i, as the pair (c_i, Λ), where Λ is the set of all the *data transformation steps* we collected by executing Algorithm 6 (line 8).

In the following, we describe how Algorithm 6 verifies whether an input (normalized) UI of type *edit* ($\bar{u}$) is deterministic. In essence, Algorithm 6 checks whether the value of the element edited by the execution of $\bar{u}$ can be deterministically computed from the UIs observed before $\bar{u}$ (in all the routine instances in

Algorithm 5: Routine automatability assessment

```
input   : Candidate Routine $c_i$, Routine Instances Set $\mathscr{R}_{c_i}$
output  : Routine Specification ($c_i$, $\Lambda$)
1  Set $\Lambda \leftarrow \varnothing$;
2  Set $E \leftarrow$ { “edit cell”, “edit range”, “edit field” };
3  Queue $D \leftarrow \varnothing$;
4  foreach Normalized UI $\bar{u} \in c_i$ do
5      if getType($\bar{u}$) $\notin E$ then
6          append $\bar{u}$ to $D$;
7      else
8          $k \leftarrow$ checkUIofTypeEdit($\bar{u}$, $c_i$, $\mathscr{R}_{c_i}$);
9          Boolean $d \leftarrow$ getDeterministic($k$);
10         if $d = true$ then
11             append $\bar{u}$ to $D$;
12             $\Lambda \leftarrow \Lambda \cup$ getTransformationStep($k$);
13 return ($c_i$, $\Lambda$)
```

$\mathscr{R}_{c_i}$). To do so, the algorithm looks for a possible data transformation function to compute the value of the edited element from the payloads of the UIs observed before $\bar{u}$. If such a data transformation function exists, $\bar{u}$ is considered deterministic and the algorithm returns the identified function in the form of a data transformation step (which also includes source(s) and target of the data transformation function).

The algorithm starts by assuming that the UI in input is non-deterministic, and it tries to prove the opposite. We initialize to false the boolean variable which we will output at the end of the algorithm (line 1), and we create the necessary data structures (line 2 to 7). Given the input candidate routine c_i and the normalized UI $\bar{u}$, we extract the index of $\bar{u}$ within c_i (line 8). Then, for each routine instance $r \in \mathscr{R}_{c_i}$, we iterate over all its UIs preceding $\bar{u}$ to collect the data required to identify a possible transformation function (lines 9 to 36). We store this data in three sets: T, K, and Π. Set T contains the transformation examples for the target element of $\bar{u}$ that are extracted similarly as in Algorithm 4. Specifically, we extract only the transformation examples related to the given target element and we save only the input and output values into queue I and variable O respectively (lines 27,34 and line 13). Set K contains all the instances of $\bar{u}$ (line 11), and set Π stores all the instances of UIs preceding $\bar{u}$ (line 18), alongside the routine instance they belong to (i.e., we store a pair (r, u_2) in Π). To find the data transformation, we leverage two state-of-the-art tools: Foofah [58][6] and TANE [69]. First, we try to identify the data transformation function using Foofah. Then – if Foofah fails – we use TANE.

These two approaches complement each other, as Foofah discovers syntactic transformations (manipulations over strings or tables), and TANE identifies semantical transformations (also known as *functional dependencies* [69]). TANE requires in input a table where each row contains $n-1$ input data values and an

[6]Foofah was discussed in Section 6.1

Algorithm 6: Check UI of Type Edit

```
input  : Normalized UI $\bar{u}$, Candidate Routine $c_i$, Routine Instances Set $\mathscr{R}_{c_i}$
output : Boolean $d$, Transformation Step $\lambda$
Boolean $d \leftarrow$ false;
Set $C \leftarrow$ { “copy cell”, “copy range”, “copy field” };
Set $E \leftarrow$ { “edit cell”, “edit range”, “edit field” };
Set $P \leftarrow$ { “paste into cell”, “paste into range”, “paste” };
Set $T \leftarrow \varnothing$;
Set $\Pi \leftarrow \varnothing$;
Set $K \leftarrow \varnothing$;
Integer $n \leftarrow$ getPosition($\bar{u}$, $c_i$);
foreach $r \in \mathscr{R}_{c_i}$ do
    UI $u_1 \leftarrow$ get(r, n);
    $K \leftarrow K \cup \{u_1\}$;
    $t_1 \leftarrow$ getTargetElement($u_1$);
    $O \leftarrow$ getParameterValue($u_1$, “Value”);
    Queue $S \leftarrow \varnothing$;
    Queue $I \leftarrow \varnothing$;
    for $i \leftarrow n$ to $1$ do
        UI $u_2 \leftarrow$ get(r, i);
        $\Pi \leftarrow \Pi \cup \{(r, u_2)\}$;
        if getType($u_2$) $\in P$ then
            $t_2 \leftarrow$ getTargetElement($u_2$);
            if $t_2 = t_1$ then
                for $j \leftarrow i$ to $1$ do
                    UI $u_3 \leftarrow$ get(r, j);
                    if getType($u_3$) $\in C$ then
                        $s \leftarrow$ getTargetElement($u_3$);
                        append $s$ to $S$;
                        append getParameterValue($u_3$, “Value”) to $I$;
                        break
        else
            if getType($u_2$) $\in E$ then
                $t_2 \leftarrow$ getTargetElement($u_2$);
                if $t_2 = t_1$ then
                    push $t_2$ to $S$;
                    push getParameterValue($u_2$,“Value”) to $I$;
                    break
    $T \leftarrow T \cup \{(I, O)\}$;
Transformation $\chi \leftarrow$ discoverTransformation($T$);
if $\chi \neq null$ then
    $d \leftarrow$ true;
    $\lambda \leftarrow (S, target, \chi)$;
else
    Set $D \leftarrow$ discoverDependencies($K$, $\Pi$);
    if $D \neq \varnothing$ then
        $d \leftarrow$ true;
        $S \leftarrow$ getSources($D$);
        $\chi \leftarrow$ extractTransformation($D$);
        $\lambda \leftarrow (S, target, \chi)$;
return $(d, \lambda)$
```

output data value in column n (this is conceptually similar to the input and output series required by Foofah). TANE analyzes each row of such a table to check if there exists any dependency between the values in the first $n-1$ columns and the value in column n.[7] An example of a semantical data transformation function discovered by TANE would be: if the value of column i is X, then the value of column n is always Y.

In our context, the input table for TANE is a table where each row represents the output data observed in all the UIs preceding $\bar{u}$ in a routine instance, and the last element of the row is the output data of the $\bar{u}$ instance in that routine (i.e., the value of the element edited by the execution of $\bar{u}$ in that routine instance). To build such a table, we require in input all the instances of $\bar{u}$ (which we stored in the set K) as well as all the instances of any UI preceding $\bar{u}$ (which we stored in the set Π). If TANE identifies a semantical data transformation function (line 43), we set $\bar{u}$ as deterministic (through the boolean d), and we compose the data transformation step using the output of TANE (see lines 44 to 47).

Table 14 shows an example of the dependency table that we would build from the log captured in Table 6 (assuming that the full-length UI log contains nine instances of the routine shown in rows 1 to 24). For example, given Table 14 as input to TANE, it would identify that the value of the last column (i.e., the type of student, domestic or international) can be deterministically generated by observing the value of column four (i.e., *country of residence*).

Table 14: Example of a dependency table

Full name	Date	Phone	Country of residence	Target
Albert Rauf	11-04-1986	043-512-4834	Germany	International
John Doe	11-03-1986	024-706-5621	Australia	Domestic
Steven Richards	18-06-1986	088-266-0827	Australia	Domestic
Hilda Diggle	31-07-1993	073-672-5593	New Zealand	International
Luca Bianchi	19-10-1998	029-211-4904	Italy	International
Igor Honchar	13-08-1993	040-656-3417	Ukraine	International
Ben Stanley	03-12-1991	244-557-2104	Australia	Domestic
Olga Mykolenchuk	11-04-2000	956-045-0703	Ukraine	International
Daniel Brown	06-04-1994	032-660-0403	New Zealand	International

If TANE does not discover any data transformation function, it means that we cannot automatically determine the value of the element edited by the execution of $\bar{u}$. Consequently, we assume that $\bar{u}$ is not deterministic. Otherwise, we output the data transformation step discovered.

Table 15: Transformation steps

Transformation step	**Sources**	**Target**	**Transformation function**
λ_1	Cell A	Full Name	χ_1
λ_2	Cell B	Date	χ_2
λ_3	Cell C	Phone	χ_3
λ_4	Cell D	Country	χ_4
λ_5	Country	Status	χ_5

[7]For more details about TANE, refer to [69].

```
χ1(I) =
  O = I
χ2(I) =
  For pattern <d>+.<d>+.<d>+:
      t = f_split(I, 0, '.')
      t = f_join_char(t,1, '-')
      O = f_join_char(t, 0, '-')
  For pattern <d>+/<d>+/<d>+:
      t = f_split(I, 0, '/')
      t = f_join_char(t,1, '-')
      O = f_join_char(t, 0, '-')
χ3(I) =
    t = f_split_first(I, 0, ' ')
    t = f_drop(t, 0)
    t = f_split_w(t, 0)
    t = f_join_char(t, 1, '-')
    O = f_join_char(t, 0, '-')
χ4(I) =
  O = I
χ5(I) =
  I → O:
    ["Germany"] → "International"
    ["Australia"] → "Domestic"
    ["New Zealand"] → "International"
    ["Italy"] → "International"
    ["Ukraine"] → "International"
```

Figure 20: Transformation functions discovered from the running example

Figure 20 shows the data transformations functions discovered by Foofah (t1 to t4) and by TANE (t5) when running Algorithm 6 on a hypothetical extended version of the UI log in Table 6 and giving as input the routine shown in rows 1 to 24 (Table 6) along with all its instances, and the *edit* UIs at rows 6, 11, 16, 21, 23 (respectively, for identifying the data transformation functions from t1 to t5). Each data transformation function shows how the input data is turned into output data.

Finally, the data transformation functions are integrated into the data transformation steps, including the instantiation of the input and the output of the function, as shown in Table 15. The degree of the automatability of a routine can be then quantified in the form of *Routine Automatability Index (RAI)*. RAI can be used to rank the routines according to their automatability potential and is calculated as the ratio of automatable UIs ($|c_i|_A$) to all UIs in the routine ($|c_i|$):

$$RAI = \frac{|c_i|_A}{|c_i|} \tag{6.1}$$

RAI is a real number that lies in the range [0, 1]. Naturally, the routine is fully automatable when it consists only of automatable UIs, and its RAI is equal to 1.

6.3. Routines aggregation

When a routine can be performed by executing a set of UIs without following a strict order, we may observe multiple execution variants of the same routine in the log. For example, if a worker needs to copy the *first name*, the *last name*, and the *phone number* of a set of customers from a spreadsheet to different web-forms, she may choose to copy the data of each customer in any order (e.g., *first name*, *phone number*, and *last name*, or *last name*, *phone number*, and *first name*). In such a scenario, the UI log would record several different execution variants of the same routine. Routine execution variants do not bring any additional value; rather, they generate redundancy within the log, leading to discovering different routine specifications that would execute (once deployed as software bots) the same routine. Such routine specifications are hence considered as duplicates and have to be removed.

To identify duplicate routine specifications, we start by generating for each discovered routine its *data transformation graph*.

Definition 6.3.1 (Data Transformation Graph). *Given a routine specification* (c_i, Λ)*, its* data transformation graph *is a graph* $G_\Lambda = (D_\Lambda, L_\Lambda)$*, where:* D_Λ *is the set of vertices of the graph, and each vertex* $d \in D_\Lambda$ *maps one data transformation step* $\lambda \in \Lambda$*;* $L_\Lambda \subseteq D_\Lambda \times D_\Lambda$ *is the set of edges of the graph, and each edge* $(d_i, d_j) \in L_\Lambda$ *represents a dependency between two data transformation steps capturing the fact that the target of the data transformation step mapped by* d_i *is (one of) the source(s) of the data transformation step mapped by* d_j*.*

Figure 21 shows the data transformation graph of the routine we discovered in the previous step in our running example.

Data transformation graphs can be used to check whether two routine specifications are equivalent, in fact, two routine specifications, (c_i, Λ_1) and (c_j, Λ_2), are equivalent if and only if the following two relations hold: i) their data transformation graphs are the same, i.e., $D_{\Lambda_1} = D_{\Lambda_2}$ and $L_{\Lambda_1} = L_{\Lambda_2}$; ii) their candidate routines c_i and c_j contain the same set of UIs, and all the UIs of type *click button* appear in the same order in both c_i and c_j.

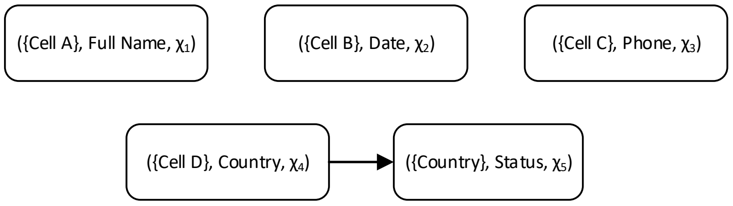


Figure 21: Data transformation graph example

By comparing each pair of routine specifications, we first create sets of equivalent routine specifications, and, for each set, we discard all the routine specifications but one. Ideally, we would like to retain the best routine specification of each set. In this regard, we need to define what it means to be the *best* one. We can select the best routine specification by relying on different quantitative metrics, such as frequency, length, or duration of the candidate routine of a routine specification. For example, we can choose frequency as a selection criterion and retain from each set the routine specification whose candidate routine is the most frequent in the UI log.

Intuitively, the most frequent candidate routine represents the common routine execution, so that one may be tempted to use that criterion by default. However, the most frequent routine execution is not necessarily the optimal execution. For example, length or duration could represent better selection criteria. Length prioritizes short candidate routines over long ones, assuming that a candidate routine should comprise as few steps as possible. Duration prioritizes execution times over the number of steps. The duration of a candidate routine can be estimated as the average execution time of each routine instance of the candidate routine recorded in the UI log. Note, however, that the duration could not always be reliable since, during the routine execution, the worker might perform activities that do not appear in the log or are not relevant for the routine execution, thus involuntarily increasing the observed execution time of the routine. For this reason, we implemented a combination of length and frequency to select the best routine specification from each set. Precisely, we use length first and then compare the frequencies of the candidate routines having the same length.

6.4. Evaluation

In this section, we report an empirical evaluation of the proposed approaches. In Section 6.4.1, we evaluate the global transformations approach, and, in Section 6.4.2, we assess the local transformations approach. Finally, Section 6.4.3 reports the threats to validity.

6.4.1. Global transformation approach

We conducted a series of experiments to evaluate the performance of our approaches (and proposed optimizations) when discovering different types of transformations. In addition, we tested the approaches on a synthetically recorded UI log that simulates a real-life use case to verify its applicability in real-life scenarios. We built a dataset using the data transferring task presented in Section 6.1. In this scenario, the students' contact information stored in an Excel spreadsheet is transferred into a Web form linked to a student management system. The task involves a number of data transformations of various types. We recorded 50 executions of such task by using the Action Logger tool [22]. We segmented this log and filtered out the noise. The row and preprocessed logs are publicly available[8].

Using this dataset, we evaluated the performance of three approaches: i) the baseline approach as per Section 6.1.1 (Baseline), ii) the approach that involves target grouping as per Section 6.1.2 (Opt 1), and iii) the approach that uses both target grouping and grouping by input structure (Opt 1 + Opt 2). We measured the time required to discover the transformation program using all three approaches for different transformation types as well as for the entire UI log. The experiments were conducted on a PC with Intel Core i5-5200U CPU 2.20 GHz and 16GB RAM, running Windows 10 as a host OS and a VM with Ubuntu 16.04 LTS (64-bit) with 8GB RAM and JVM 11 (4GB RAM). We used the Foofah tool[9] with a timeout of one hour. The tool we developed for this experiment, which implements the three approaches, is publicly available.[10]

Table 16 describes how the three approaches perform on different types of transformations, while Table 17 shows their computational efficiency on the entire

[8]`https://figshare.com/articles/UI_logs/10269845`

[9]`https://github.com/umich-dbgroup/foofah`

[10]`https://github.com/volodymyrLeno/RPM`

UI log. The latter table also reports on the discovery quality of the approaches. The execution time is shown in seconds.

Table 16: Global transformations approach discovery results

Transformation type	**Example**	**Execution time**		
		Baseline	**Opt 1**	**Opt 1 + Opt 2**
N - 1	“Igor", “Honchar" ⇒ “Igor Honchar"	**1.295**	1.584	1.745
1 - 1	“18/08/1992" ⇒ “18-08-1992"	6.584	6.639	**0.476**
1 - 1	“+61 029 211 4904" ⇒ “029-211-4904"	N/A (2306.036)	N/A (2271.19)	**0.5086**
1 - 1	“New Zealand" ⇒ “New Zealand"	**0.347**	0.392	0.704
1 - 1	“wmacdonald@gmail.com" ⇒ “wmacdonald@gmail.com"	**0.34**	0.391	0.397
1 - N	“122 Albert St, Port Melbourne, VIC 3207, Australia" ⇒ “122 Albert St", “Port Melbourne", “VIC", “3207"	timeout	7504.934	**85.423**
1 - 1	“122 Albert St, Port Melbourne, VIC 3207, Australia" ⇒ “122 Albert St"	-	**1.243**	1.55
1 - 1	“122 Albert St, Port Melbourne, VIC 3207, Australia" ⇒ “Port Melbourne"	-	N/A (1983.501)	**54.777**
1 - 1	“122 Albert St, Port Melbourne, VIC 3207, Australia" ⇒ “VIC"	-	timeout	**26.603**
1 - 1	“122 Albert St, Port Melbourne, VIC 3207, Australia" ⇒ “3207"	-	N/A (1884.397)	**2.49**

From Table 16, we can see that the baseline approach performs better than the two optimizations for very simple examples (three out of ten cases). This is because it does not require any additional steps (e.g., grouping transformation examples into equivalence classes). However, in the case of complex transformations, its efficiency drops significantly. It spent around 2300 seconds discovering the transformation to convert the phone numbers into the required format and did not manage to synthesize any transformation program. At the same time, it resulted in a timeout when identifying the transformation for the address. The second approach (Opt 1) discovered more transformations compared to the baseline. It outperforms the baseline in the case of complex 1-N transformations as it decomposes the corresponding transformation into a set of small transformations of 1-1 type and then discovers them separately. However, this approach discovered only a fraction of such transformations, as it could not deal with heterogeneous data (e.g., for cities/suburbs) and ambiguous outputs (e.g., for zip codes). In these cases, it took around 30 minutes to conclude that there is no transformation or that it is impossible to discover one. In contrast, the third approach (Opt 1 + Opt 2) discovered all data transformations within a reasonable time (up to 85 seconds). By clustering the patterns, this approach handles heterogeneous data, while by providing only one example from each equivalence class, it deals with ambiguous output and speeds up the transformation discovery step. However, in some cases, it was slightly slower than the other two approaches because of the high number of input patterns. Moreover, even when the transformation is simple, this approach discovers it for all patterns, thus requiring more time.

The baseline did not discover any transformation for the entire UI log because it was too complex and involved many source and target elements, thus resulting in a timeout. Since grouping by target allows us to split a large transformation problem into smaller problems so that each problem can be solved in isolation,

Table 17: Transformation discovery use case

Approach	Execution time (sec)	Discovered transformations
Baseline	3742.669	0/9
Opt 1	10551.536	5/9
Opt 1 + Opt 2	130.854	9/9

Opt 1 could discover five out of nine data transformations. However, it took much longer than the baseline because some minor problems were too difficult to solve and required a considerable amount of time (nearly three hours). In contrast, Opt 2 discovered all nine transformations in around two minutes.

6.4.2. Local transformation approach

When evaluating the local transformation approach presented in Section 6.2, we aimed at assessing whether it can correctly identify automatable (and non-automatable) UIs within candidate routines discovered by the approach in Chapter 5. Considering this task as a classification problem, we assess the quality of the approach by measuring precision, recall, and F-score metrics. For each discovered routine, we compute the corresponding confusion matrix, where *true positives* (TP) are correctly identified automatable UIs, *true negatives* (TN) are correctly identified non-automatable UIs, *false positives* (FP) are the UIs that were wrongly marked as automatable, and *false negatives* (FN) are the UIs that were incorrectly marked as non-automatable. From the constructed confusion matrix, we calculate precision, recall, and F-score as follows:

$$Precision = \frac{TP}{TP+FP} \tag{6.2}$$

$$Recall = \frac{TP}{TP+FN} \tag{6.3}$$

$$F\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{6.4}$$

We report the averages of these metrics for all the discovered routines in the log. In addition, we also report the average RAI for the routines in the log. Finally, to show the performance of the approaches, we present the execution times measured in seconds. All experiments were conducted on a Windows 10 laptop with an Intel Core i5-5200U CPU 2.20 GHz and 16GB RAM, using cohesion as a routine selection criterion with the minimum support threshold set to 0.1 and the minimum coverage threshold equal to 0.05. The approach is implemented as an open-source tool.[11]

Table 18 shows the quality of the automatable routines discovery. We correctly identified all the automatable and non-automatable user interactions for the CPN3, CPN6, SR, and S1 logs. The routines recorded in the CPN3 and SR logs are fully

[11]Available at `https://github.com/volodymyrLeno/RPM_Miner`

automatable. Although the RT log contains automatable routines only, our approach failed to discover some of the underlying transformations and, therefore, incorrectly marked some interactions as non-automatable. On the other hand, some of the user interactions of the synthetic logs were wrongly identified as automatable. Although the data values of such interactions can be deterministically computed, the locations of the edited elements were completely random as intended in the corresponding models. Thus, in practice, such interactions are non-automatable. The routines discovered from the CPN5 log are characterized by the lowest number of automatable user interactions, and we achieved the lowest recall for this log (0.805). Overall, F-score is high, above 0.85 for all the logs, except CPN7 and CPN8. We also achieved the lowest recall for these logs, meaning that some interactions of the corresponding routines were wrongly identified as non-automatable. Although in the CPN models used to generate the artificial logs, some of the interactions are non-deterministic, they are automatable in the context of the discovered routines. For example, for the CPN9 log, we discovered six routines that correspond to the different branches within the model. For all the executions of a branch, we use the same data values, and hence, the corresponding user interactions are automatable.

Table 18: Local transformations approach discovery results

UI Log	**# Discovered specifications**	**RAI (avg)**	**Precision (avg)**	**Recall (avg)**	**F-score (avg)**	**Execution time (sec)**
CPN1	1	1.000	0.928	1.000	0.963	7.148
CPN2	2	0.931	0.926	1.000	0.961	17.408
CPN3	3	1.000	1.000	1.000	1.000	15.545
CPN4	4	0.786	1.000	0.846	0.917	18.409
CPN5	8	0.728	0.812	1.000	0.896	19.761
CPN6	2	0.742	1.000	1.000	1.000	6.102
CPN7	7	0.546	0.907	0.805	0.841	17.594
CPN8	6	0.612	0.897	0.823	0.845	17.399
CPN9	6	0.741	0.951	0.886	0.916	18.798
SR	2	1.000	1.000	1.000	1.000	845.255
RT	2	0.967	1.000	0.967	0.983	1066.041
S1	5	1.000	1.000	1.000	1.000	21.400

While the execution time is reasonably low for all the artificial logs (less than 20 seconds), it substantially increases for the SR and RT logs. This is caused by the fact that the underlying transformations in these two logs were very complex, often involving regular expressions or long sequences of manipulations. In contrast, all the transformations in the CPN1-CPN9 logs were simple copy-paste operations. Our approach marked the routines discovered from the S1 log as fully automatable, and an employee of the university confirmed that the results are correct. Since for the S2 log we did not manage to identify any meaningful routine, we could not discover any executable specification.

6.4.3. Threats to validity

One of the threats to validity is a low number of the logs used in evaluation (especially when evaluating the global transformations approach), which leads to limited generalizability of the findings. However, these logs are characterized by different complexity and size. The tasks recorded in these logs intentionally involve different types of data transformations to check the ability of our technique to discover them. Although the amount of data transformations is limited, these transformations are the most commonly seen in practice. Another potential threat is that we do not use any baseline approach in our evaluation. However, this is justified because there are no available approaches to discover executable routines that can be directly implemented in software scripts. Finally, we used Foofah as data transformation discovery technique because it is open-source, and thus, it can be integrated into our approach. It is possible, however, that there exist more efficient transformation discovery techniques.

6.5. Summary

In this chapter, we addressed **RQ2** posed in the thesis[12]. We presented two alternative approaches to discover executable specifications of routines. The first approach is focused on the data transferring tasks and discovers specification in the form of a data transformation program that converts the values from the source application into the values of the target application where the data is transferred to. This approach exploits a state-of-the-art transformation discovery technique called Foofah and proposes several optimizations to deal with Foofah's limitations and to improve its efficiency. The second approach covers a broader range of routines that can be automated by discovering sequential routine specifications that capture all activities that were performed. We showed how the transformation discovery could be used to assess the automatability of routines and presented a method to identify and filter out routines with identical effects.

Both approaches are implemented in the form of open-source tools and evaluated using synthetic and real-life logs. For the global transformation approach, the evaluation results demonstrate that it can discover various transformation types and that the proposed optimizations significantly improve its efficiency. The evaluation of the local transformation approach shows that it identifies automatable and non-automatable UIs of the discovered routines with high accuracy. For the majority of the tested logs, the execution time does not exceed 20 seconds. The only exceptions are the logs with complex underlying data transformations.

The proposed approaches have two main limitations. First, when assessing the automatability of a routine, they assume that the values of the edited fields are entirely derived from the (input) fields that are explicitly accessed (e.g., via copy operations) during the routine's execution. Hence, they will fail to automate UIs where a worker visually reads from a field (without performing a copy operation) and writes what he sees into another fields. An avenue for addressing this limitation is to complement the proposed methods with optical character recognition

[12]*Given a routine how to discover its executable specification that can be executed via an RPA tool?*

techniques over screenshots taken during the UI log recording, to be able to detect that some of the outputs of a routine come from fields that have not been explicitly accessed via a copy-to-clipboard operation. The second limitation is that the proposed approaches cannot discover conditional behavior, where the transformation function for the target field depends on the value of another field. Consider, for example, a routine that involves copying delivery data. If the delivery country is USA, then the month comes before the day (MM/DD/YYYY), otherwise the day comes before the month. Here, the transformation function depends on a condition of the form "country = USA", which the proposed approaches cannot discover. In a similar vein, the proposed approaches are able to discover transformations that depend on the structural pattern of the value of the input field(s), but they fail to distinguish the patterns that, although having the same syntactical structure, have different semantics. Following the example above, our approaches will put both date types into the same equivalence class. Addressing this limitation would require the development of more sophisticated data transformation discovery techniques, beyond the capabilities of Foofah.

There are several criteria for determining whether or not a routine should be automated, e.g., as presented in [47, 51]. For example, one may select the most frequently performed routines for automation. Alternatively, an amount of effort spent on a routine (i.e., how much time is required to perform a routine from start to end) can be considered when prioritizing routines for automation. This thesis focuses on the automatability criterion, which captures the feasibility and cost of the automation effort, and only considers the other criteria indirectly. In order to maximize the benefits of task automation, the discovered automatable routines may be ranked according to an organization's own priorities.

7. SOFTWARE IMPLEMENTATION

In this chapter, we present the tools developed during this project.[1]. First, in Section 7.1, we talk about Action Logger, a tool to capture user interactions with IT systems and to record them in the form of UI logs. Then, in Section 7.2, we give an overview of the Robidium, a tool that implements the rest of the RPM pipeline and integrates the approaches presented in this thesis. Finally, Section 7.3 concludes the chapter.

7.1. Action Logger

The creation of an RPA bot requires an in-depth knowledge of the tasks to be automated, the IT systems involved, their interfaces, and how users interact with them. In the current practice, this knowledge is gained via interviews and workshops with the stakeholders, and analysis of unstructured data, e.g., video recordings of users working with the systems. Unfortunately, this approach is time-consuming and error-prone, which significantly affects the quality of the bots developed. In addition, this leads to a significant amount of time spent on developing and testing the bots.

Alternatively, the information about task executions can be saved in user interactions (UI) logs. In this thesis, we presented a family of techniques, called Robotic Process Mining, to identify automatable tasks from such logs and synthesize their executable specifications that can be used as a starting point for the automation efforts.

Hence, a question arises on how to obtain such logs. A typical UI log can be recorded by a logging tool that captures all the actions (e.g., click a button, edit field) performed by a user across multiple applications during the execution of her daily tasks. To generate UI logs suitable for Robotic Process Mining, a logging tool should ensure the following functionality requirements:

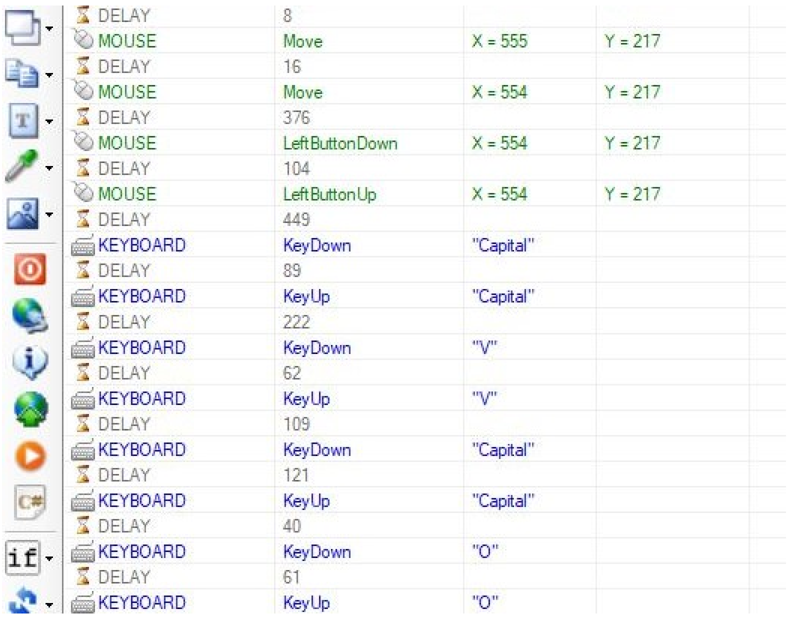
- **REQ1: Relevance.** A tool should only record meaningful, value-adding actions. For example, moving the mouse or clicking on the background of a web site should not be recorded as it does not impact the outcome of a task. However, button clicks and URI link clicks are essential actions and, therefore, should be captured.
- **REQ2: Granularity.** A tool should record actions at a level of detail sufficient to fully reconstruct the performed task. For example, the logger should differentiate different types of mouse clicks, e.g., clicking on a button versus clicking a link in a web browser.
- **REQ3: Data-awareness.** In addition to performed actions, a tool should record the data that supports them. This data is crucial in discovering the

[1]Corresponding to [22] and [70]

rule-based decision-making logic of the process. The data can also be used to discover data transformations to enhance the quality of the discovered process model. To enable performance analysis, the tool should also record timestamps associated with the performed actions.

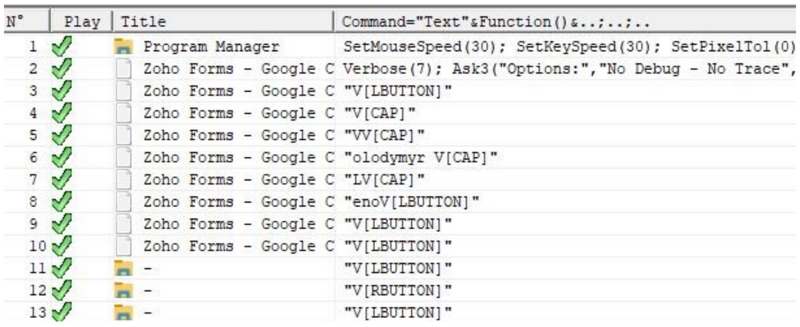
- **REQ4: Context-independence.** A tool should record actions so that they can be replayed with the same effects on different machines and platforms under various circumstances and contexts, e.g., different UI layouts.
- **RESQ5: Interoperability.** A tool should record UI logs in a structured format, for example, CSV or XES [71].

To the best of our knowledge, no solution satisfies all the above requirements. Available UI action recording tools, like WinParrot (`www.winparrot.com`) and JitBit Macro Recorder (`www.jitbit.com/macro-recorder`), record low-level actions only, e.g., clickstreams and keystrokes (see Figures 22 and 23). The recorded actions refer to pixel coordinates (e.g., click the mouse at coordinates 341, 568) that depend on screen resolution and window size. Some tools, like WinParrot, save information regarding the application where the action was performed. However, they do not identify application-specific functionality, e.g., editing or copying a cell in a spreadsheet. Most of the tools do not capture timestamps. However, some of them save the delays between actions. None of the existing tools generates files in a format that robotic process mining techniques can directly consume.



```
DELAY      8
MOUSE      Move             X = 555    Y = 217
DELAY      16
MOUSE      Move             X = 554    Y = 217
DELAY      376
MOUSE      LeftButtonDown   X = 554    Y = 217
DELAY      104
MOUSE      LeftButtonUp     X = 554    Y = 217
DELAY      449
KEYBOARD   KeyDown          "Capital"
DELAY      89
KEYBOARD   KeyUp            "Capital"
DELAY      222
KEYBOARD   KeyDown          "V"
DELAY      62
KEYBOARD   KeyUp            "V"
DELAY      109
KEYBOARD   KeyDown          "Capital"
DELAY      121
KEYBOARD   KeyUp            "Capital"
DELAY      40
KEYBOARD   KeyDown          "O"
DELAY      61
KEYBOARD   KeyUp            "O"
```

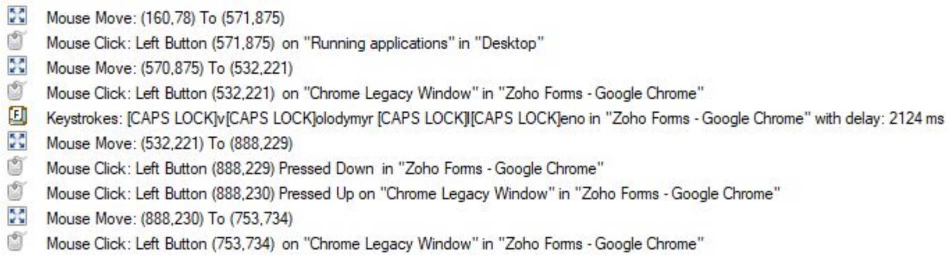
Figure 22: Fragment of a log recorded by JitBit



N°	Play	Title	Command="Text"&Function()&..;..;..
1	✓	Program Manager	SetMouseSpeed(30); SetKeySpeed(30); SetPixelTol(0)
2	✓	Zoho Forms - Google C	Verbose(7); Ask3("Options:","No Debug - No Trace",
3	✓	Zoho Forms - Google C	"V[LBUTTON]"
4	✓	Zoho Forms - Google C	"V[CAP]"
5	✓	Zoho Forms - Google C	"VV[CAP]"
6	✓	Zoho Forms - Google C	"olodymyr V[CAP]"
7	✓	Zoho Forms - Google C	"LV[CAP]"
8	✓	Zoho Forms - Google C	"enoV[LBUTTON]"
9	✓	Zoho Forms - Google C	"V[LBUTTON]"
10	✓	Zoho Forms - Google C	"V[LBUTTON]"
11	✓	-	"V[LBUTTON]"
12	✓	-	"V[RBUTTON]"
13	✓	-	"V[LBUTTON]"

Figure 23: Fragment of a log recorded by WinParrot

RPA tools, like Automation Anywhere (`www.automationanywhere.com`) and UIPath (`www.uipath.com`), provide a wide range of instruments to program the RPA bots manually. They also provide recording capabilities to generate bots automatically. However, the generated logs are only readable within the environment of the RPA solutions themselves. Moreover, such logs capture user interactions at a low level of granularity and often do not save the data used during the execution of the routine. Figure 24 shows a fragment of a log recorded by the Automation Anywhere RPA tool.



```
Mouse Move: (160,78) To (571,875)
Mouse Click: Left Button (571,875)  on "Running applications" in "Desktop"
Mouse Move: (570,875) To (532,221)
Mouse Click: Left Button (532,221)  on "Chrome Legacy Window" in "Zoho Forms - Google Chrome"
Keystrokes: [CAPS LOCK]v[CAPS LOCK]olodymyr [CAPS LOCK]l[CAPS LOCK]eno in "Zoho Forms - Google Chrome" with delay: 2124 ms
Mouse Move: (532,221) To (888,229)
Mouse Click: Left Button (888,229) Pressed Down  in "Zoho Forms - Google Chrome"
Mouse Click: Left Button (888,230) Pressed Up on "Chrome Legacy Window" in "Zoho Forms - Google Chrome"
Mouse Move: (888,230) To (753,734)
Mouse Click: Left Button (753,734)  on "Chrome Legacy Window" in "Zoho Forms - Google Chrome"
```

Figure 24: Fragment of a log recorded by Automation Anywhere

Based on the identified requirements, we designed and implemented our own recording tool called Action Logger[2,3]. The tool records user actions performed in Excel and Chrome web browser, two of the most often used applications for office tasks. It includes two separate plug-ins, one for each application. The plug-ins are implemented as event listeners and send the information about performed actions as JSON objects to the logging component, generating and updating the UI log on the fly. To record the actions, the logger uses APIs of the corresponding applications. The browser actions are recorded at the Document Object Model (DOM) level, capturing the involved web elements, e.g., text fields, buttons, and links. The tool also monitors the clipboard to record relevant actions, e.g., copying and pasting of data.

[2]The tool is available at `https://github.com/apromore/RPA_UILogger/releases`

[3]The video tutorial on how to use the tool can be found at `https://youtu.be/SvPuOdWfByc`

The architecture of the tool and the envisaged pipeline for employing the logger for RPA are shown in Figure 25. The tool stores all the data values used in the context of every recorded action. For example, for an action performed in a spreadsheet, the tool captures the information about the cell, its current value, workbook, and active sheet in which the action took place. The generated logs are stored in a CSV format suitable for process mining investigations. This design choice was made to improve the efficiency and speed of the logger since the actions are recorded at run time, and the XES format is too verbose.

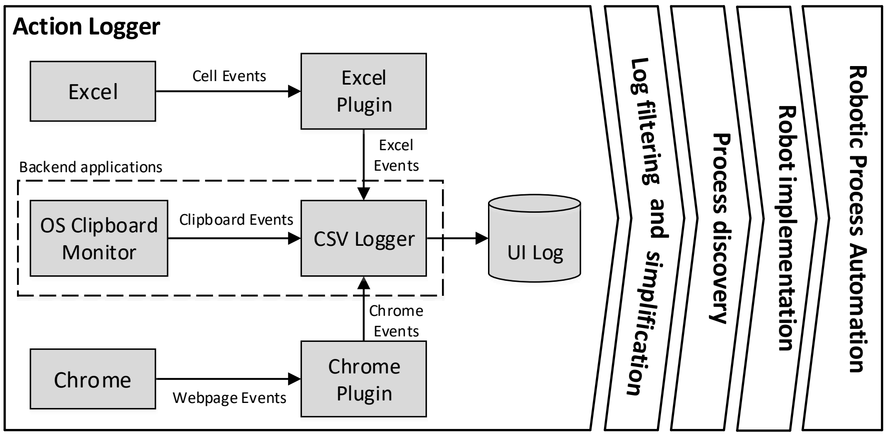
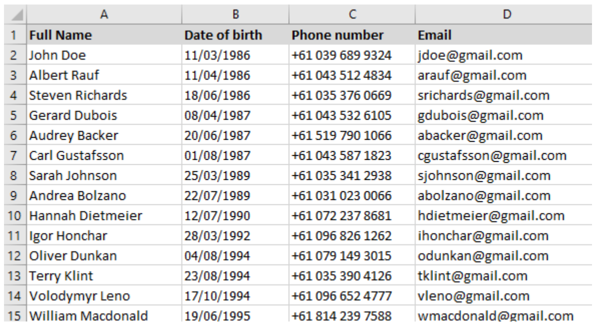


Figure 25: Action Logger architecture within the RPA pipeline

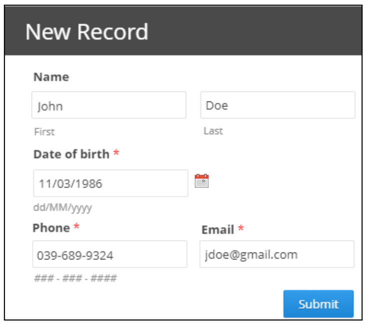
One typical task automated using RPA is the task of transferring data from one system to another, for example, from a spreadsheet to a form of a web-based information system. To demonstrate Action Logger, we use an Excel spreadsheet containing students' contact details, e.g., full name, date of birth, phone number, email, and web form. We manually transferred the data about the first student in the spreadsheet from Excel into the web form (refer to Figure 26) and recorded all the performed actions using Action Logger. Figure 27 shows a fragment of the UI log produced by the logger; for simplicity, we do not show all the recorded UI parameters (you can find the list of all recorded parameters and their description in Table 19).

To validate Action Logger in practice, we established a cooperation with the University of Melbourne, Australia. In particular, we worked closely with the University Services team responsible for the university's admission and scholarship allocation processes. The team used Action Logger to record the workers while performing routine operations and provided feedback with respect to validity (all recorded actions are correct and relevant) and completeness (all the important and relevant actions are recorded) of the recorded logs; note that many of these routines involve work with spreadsheets and web-based front-ends of the university IT systems, similar to the example presented in Figure 26. The collected feedback was used to improve the tool. As a result, with Action Logger, we produced most of the UI logs used to evaluate the approaches presented throughout the thesis.



	A	B	C	D
1	**Full Name**	**Date of birth**	**Phone number**	**Email**
2	John Doe	11/03/1986	+61 039 689 9324	jdoe@gmail.com
3	Albert Rauf	11/04/1986	+61 043 512 4834	arauf@gmail.com
4	Steven Richards	18/06/1986	+61 035 376 0669	srichards@gmail.com
5	Gerard Dubois	08/04/1987	+61 043 532 6105	gdubois@gmail.com
6	Audrey Backer	20/06/1987	+61 519 790 1066	abacker@gmail.com
7	Carl Gustafsson	01/08/1987	+61 043 587 1823	cgustafsson@gmail.com
8	Sarah Johnson	25/03/1989	+61 035 341 2938	sjohnson@gmail.com
9	Andrea Bolzano	22/07/1989	+61 031 023 0066	abolzano@gmail.com
10	Hannah Dietmeier	12/07/1990	+61 072 237 8681	hdietmeier@gmail.com
11	Igor Honchar	28/03/1992	+61 096 826 1262	ihonchar@gmail.com
12	Oliver Dunkan	04/08/1994	+61 079 149 3015	odunkan@gmail.com
13	Terry Klint	23/08/1994	+61 035 390 4126	tklint@gmail.com
14	Volodymyr Leno	17/10/1994	+61 096 652 4777	vleno@gmail.com
15	William Macdonald	19/06/1995	+61 814 239 7588	wmacdonald@gmail.com

(a) Student records spreadsheet



New Record

Name
John — First
Doe — Last

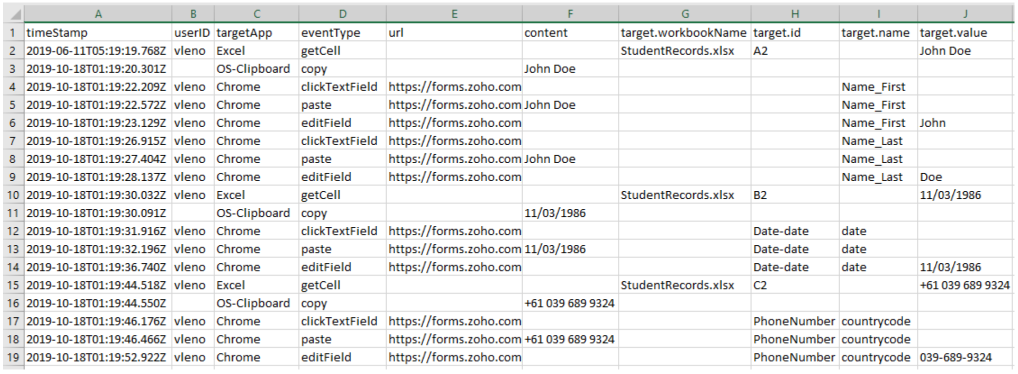
Date of birth *
11/03/1986
dd/MM/yyyy

Phone *
039-689-9324
- ### -

Email *
jdoe@gmail.com

(b) New Record creation form

Figure 26: Use case to demonstrate the work of Action Logger



	A	B	C	D	E	F	G	H	I	J
1	timeStamp	userID	targetApp	eventType	url	content	target.workbookName	target.id	target.name	target.value
2	2019-06-11T05:19:19.768Z	vleno	Excel	getCell			StudentRecords.xlsx	A2		John Doe
3	2019-10-18T01:19:20.301Z		OS-Clipboard	copy		John Doe				
4	2019-10-18T01:19:22.209Z	vleno	Chrome	clickTextField	https://forms.zoho.com				Name_First	
5	2019-10-18T01:19:22.572Z	vleno	Chrome	paste	https://forms.zoho.com	John Doe			Name_First	
6	2019-10-18T01:19:23.129Z	vleno	Chrome	editField	https://forms.zoho.com				Name_First	John
7	2019-10-18T01:19:26.915Z	vleno	Chrome	clickTextField	https://forms.zoho.com				Name_Last	
8	2019-10-18T01:19:27.404Z	vleno	Chrome	paste	https://forms.zoho.com	John Doe			Name_Last	
9	2019-10-18T01:19:28.137Z	vleno	Chrome	editField	https://forms.zoho.com				Name_Last	Doe
10	2019-10-18T01:19:30.032Z	vleno	Excel	getCell			StudentRecords.xlsx	B2		11/03/1986
11	2019-10-18T01:19:30.091Z		OS-Clipboard	copy		11/03/1986				
12	2019-10-18T01:19:31.916Z	vleno	Chrome	clickTextField	https://forms.zoho.com			Date-date	date	
13	2019-10-18T01:19:32.196Z	vleno	Chrome	paste	https://forms.zoho.com	11/03/1986		Date-date	date	
14	2019-10-18T01:19:36.740Z	vleno	Chrome	editField	https://forms.zoho.com			Date-date	date	11/03/1986
15	2019-10-18T01:19:44.518Z	vleno	Excel	getCell			StudentRecords.xlsx	C2		+61 039 689 9324
16	2019-10-18T01:19:44.550Z		OS-Clipboard	copy		+61 039 689 9324				
17	2019-10-18T01:19:46.176Z	vleno	Chrome	clickTextField	https://forms.zoho.com			PhoneNumber	countrycode	
18	2019-10-18T01:19:46.466Z	vleno	Chrome	paste	https://forms.zoho.com	+61 039 689 9324		PhoneNumber	countrycode	
19	2019-10-18T01:19:52.922Z	vleno	Chrome	editField	https://forms.zoho.com			PhoneNumber	countrycode	039-689-9324

Figure 27: Fragment of a log recorded by Action Logger

7.2. Robidium

Robidium is a Software as a Service (SaaS) tool[4,5,6] that implements the Robotic Process Mining pipeline presented in Chapter 3. It identifies and automates the routine tasks present in UI logs. Unlike simple macro-recording tools that allow one to record and replay an already well-scoped routine, Robidium discovers routines from long-running recordings of user interactions, for example, a recording of a full working day. Given a UI log, Robidium proceeds by identifying recorded task instances and filtering out redundant behavior. Next, it discovers frequently repeated sequences of UIs (with gaps), which are then tagged as candidates for automation. Finally, each candidate pattern is assessed for its amenability to automation. To this end, the tool discovers dependencies between data elements within each candidate and uses this information to synthesize automatable specifications. Such specifications are then compiled into executable RPA scripts.

[4]Available at `http://robidium.cloud.ut.ee/`

[5]Tutorial available at `https://github.com/volodymyrLeno/Robidium`

[6]A video about the tool can be found at `https://youtu.be/3M8qveb1meY`

Table 19: UI parameters recorded by Action Logger

Attribute	Description
timeStamp	Date and time when the action was performed
userID	Name of the user who performed the action
targetApp	Application where the action was performed
eventType	Type of the action (e.g., copyCell, paste)
url	URL of the website where the action was performed
content	Content copied or pasted during the action
target.workbookName	Excel workbook where the action was performed
target.SheetName	Excel sheet where the action was performed
target.id	Identifier of the element involved in the action (id of a web element or address of a cell in Excel)
target.class	Class of the element involved in the action
target.tagName	Tag of the element involved in the action
target.type	Type of the element involved in the action
target.name	Name of the element involved in the action
target.value	Value of the element involved in the action
target.innerText	Label of the element involved in the action (the text visible on the page)
target.checked	If the involved element is a check box or a radio button, shows whether it is checked
target.href	The URL of the page the element leads to
target.option	Shows the possible options for select action
target.title	Title of the element involved in the action
target.innerHTML	HTML representation of the element involved in the action

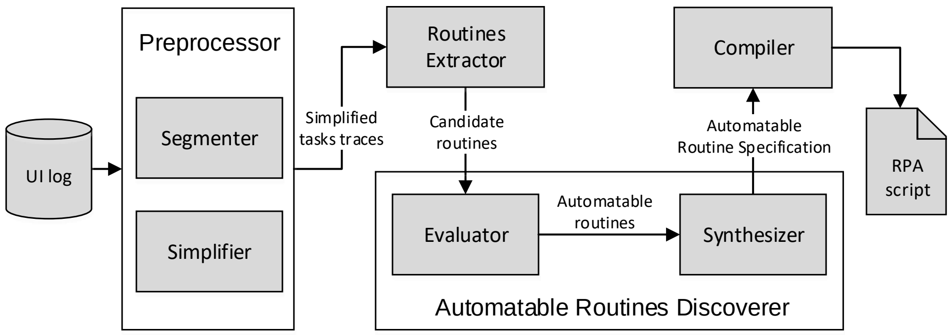


Figure 28: Robidium architecture

Robidium's architecture consists of six components (Figure 28) as detailed below.

Segmenter. Robidium takes as input a UI log in which each row includes a timestamp, one or more attributes that (combined) denote an action (e.g., "Edit cell" + cell ID in Excel), and other attributes capturing the action's payload (e.g., the value of the Excel cell after the action). UI logs that fulfill these requirements can be produced using the Action Logger tool presented in Section 7.1, supporting Excel and the Chrome browser. Other loggers can be used provided that they are converted to the Action Logger's format. By default, Robidium takes as input a UI log consisting of a single sequence of actions recorded during a working session. This session may contain multiple executions of one or more tasks (e.g., creating a new student record, adding new credentials to an existing student record). The *Segmenter* assumes that the user only performs one instance of one task at a time (no overlapping task instances), that the instances of multiple tasks do not share any identical actions, and that instances of multiple tasks do not always appear contiguously, but are rather separated by some events that are not part of a task instance. Under these assumptions, the *Segmenter* breaks down the single-sequence UI log into a set of sequences.

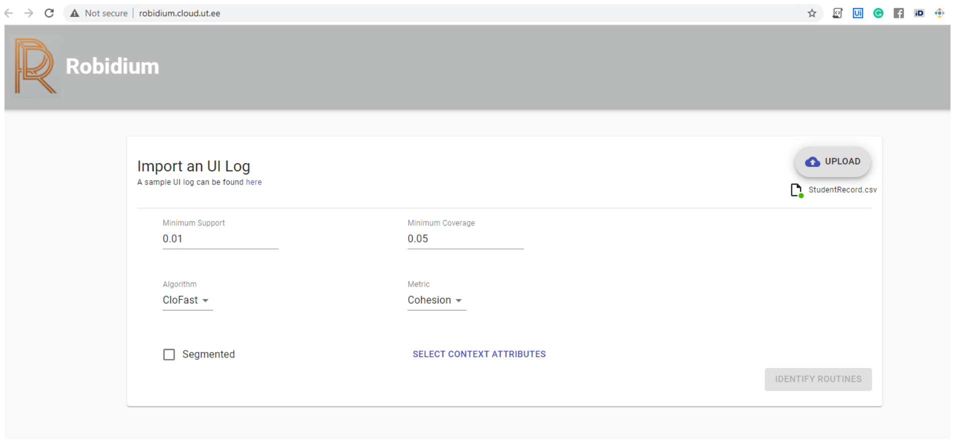
Simplifier. A UI log may contain redundant behavior that does not affect the outcome of the recorded task. For example, the user could fill in the field with the wrong value by mistake and then correct it. This can lead to incorrect identification of routines. The *Simplifier* component eliminates such redundant subsequences of actions in a semantics-preserving manner. The *Simplifier* works in two steps. First, it converts operating system level actions into application-level actions (e.g., a *get cell* and a *clipboard copy* actions are merged into a *copy cell* action). Next, it removes redundant actions from the log by applying a set of regular expression find-and-replace rules. Some of these expressions are purely control-flow-based (e.g., navigation events and double copying), while others are data-aware (e.g., double editing of a text field with replacement of its value). The *Simplifier* consists of three sub-modules responsible for different types of redundancies related to read, write, and navigation actions.

Routines extractor. The *Simplifier* returns a list of task traces without redundant actions. These task traces are then provided as input to *Routines extractor*, which identifies routine candidates for automation. Each user interaction in the log is converted into its symbolic representation by combining type and context attributes that capture where the action was performed (e.g., application, URL, field name, and button label). The user selects the context attributes. To find repeats, user interactions across the traces must have identical symbolic representations. Therefore, we do not use the attributes that contain the data used during the execution of an action (e.g., the value of a field, copied content). The tool applies sequential pattern mining to identify frequently repetitive execution patterns from sequences of symbolic representations of user interactions. Such patterns are then considered to be candidates for automation. The routine candidates can be selected accordingly to different criteria such as length, frequency, coverage, or cohesion.

Evaluator. Each candidate routine then must be assessed for its amenability to automation by the *Evaluator*. The *Evaluator* extracts its instances from the log for each candidate and verifies whether all the actions are automatable. In particular, an action can be automated if its input values can be computed from the outcomes of the previous actions using a constant or deterministic function. To this aim, the *Evaluator* discovers the data transformations between the actions in the instances of the routine candidate. It discovers the syntactic and semantical transformations

as described in Section 6.2. By default, all non-edit actions (e.g., copy cell, click button) are considered to be automatable. For each routine, it then calculates a routine automatability index (RAI) as the ratio of its automatable actions.
Synthesizer. Given a set of candidate routines annotated with RAI, the user can select which routine should be implemented. The *Synthesizer* then prepares the automatable specification for the selected routine. Finally, it annotates the actions of a routine with the corresponding data transformations and extracts the information required to map the actions to the application elements involved during the routine execution (e.g., button or text field in the web form).
Compiler. The automatable routine specifications are then given as input to the *Compiler* that generates an RPA bot by mapping each action of the routine into the corresponding executable command of the selected RPA tool. At the moment, Robidium creates RPA bots for the UIPath Enterprise RPA Platform.[7] These scripts can then be executed via command line or via the UIPath interface. The *Compiler* also identifies the variables in the script (e.g., row in the spreadsheet) that can then be used as input parameters during its execution.

Figure29 shows the Robidium's interface. Here, a user can upload a UI log, select the context attributes and specify the input parameters for the algorithm that will be used to identify candidate routines for automation.


Not secure | robidium.cloud.ut.ee

Robidium

Import an UI Log
A sample UI log can be found here

StudentRecord.csv

Minimum Support: 0.01
Minimum Coverage: 0.05
Algorithm: CloFast
Metric: Cohesion
☐ Segmented

Figure 29: Robidium interface

7.3. Summary

In this chapter, we presented two major tools that were developed as a result of this project. These tools aim at enabling analysis techniques to improve the early stages of the RPA lifecycle, specifically the identification of automatable routines and their implementation.

[7] `www.uipath.com`

The main obstacle in introducing such techniques is the absence of tools capable of recording UI logs of a quality sufficient for insightful analysis. To close this gap, we developed our logging tool, called Action Logger, that captures user interactions with Excel and Web browser applications and stores them in UI logs. Unlike most existing recording tools, Action Logger captures UIs at the application level instead of clickstreams and keystrokes. This allows one to understand what activities were executed by a worker and implement the bots with better awareness. The bots generated from such logs are more flexible and do not depend on the environment (e.g., screen resolution, applications currently opened). Action Logger extracts and saves all the information vital for automating the captured UIs, such as the elements involved in the action, their exact locations, and the values. However, to correctly capture the UIs, the tool heavily relies on the APIs of the applications under recording. Hence, it is challenging to extend it to applications without available APIs (e.g., pdf readers). To address this problem, one may consider using image recognition techniques and OCR alongside the Action Logger tool.

Robidium combines all the approaches and methods presented in this thesis into an end-to-end pipeline for automating routine tasks. The tool aims at reducing the amount of time spent on the identification and analysis of the candidates for automation, and allows instead to focus on their implementation. Unlike record-and-replay features provided by commercial RPA tools, Robidium takes as input a UI log that is not explicitly recorded to capture a pre-identified task. Instead, the log may contain mixtures of automatable and non-automatable routines, interspersed with UIs that are not part of any routine, as well as redundant or irrelevant UIs. The output of Robidium is an executable bot that implements a routine selected by a user. This bot can then be used as a starting point for the automation effort and can be further refined by RPA developers. For future work, we plan to address some of the Robidium's limitations. First, Robidum can only generate scripts of fully automatable routines. It does not support steps that require intermediate user input. Second, the tool automates only one variant of a routine at a time. If a routine has multiple variants (e.g., handling different use cases), multiple bots are generated. We plan to add a functionality to combine multiple variants of a routine into a single more complex executable specification that can be compiled into a single bot. Finally, we plan to improve the efficiency of the various algorithms implemented in the tool to support larger and more complex UI logs.

8. CONCLUSION

8.1. Summary of contributions

In this thesis, we have exposed a vision for a new class of techniques, namely Robotic Process Mining (RPM), capable of analyzing logs of fine-grained user interactions with IT systems in order to identify routines that can be automated using RPA tools. RPM aims at helping RPA developers and analysts in the early stages of the RPA lifecycle. Specifically, it assists the analysts in drawing a systematic inventory of the tasks that can be automated with RPA and synthesizes executable specifications of such tasks that can be used as a starting point for their automation.

This thesis makes four contributions to the RPA community. The first contribution is the design of a format to store UI logs and the development of a tool to record them. The main challenge in introducing RPM is the lack of logging tools capable of recording UI logs suitable for analysis. The logs recorded by such tools capture the user's interactions at a too low level of granularity (e.g., clickstreams and keystrokes), thus losing the information about semantics and effects of the performed actions. Hence, it is nearly impossible to extract any valuable insights about tasks' execution from such logs. The format we propose specifies what information is required to be captured by the logging tool to analyze, improve and automate the underlying user interactions. Our tool records user interactions performed in two selected applications, namely Chrome and MS Exce that cover the majority of routine tasks performed on top of these applications. It captures all the data values used during the execution of these interactions so that the corresponding routines can be analyzed with respect to their automatability and implemented in the form of executable scripts.

The second contribution is an end-to-end pipeline to automate routines. The underlying pipeline takes a UI log as input and produces executable specifications of the automatable routines captured in the log. It starts from removing redundant activities that are not important for analysis and segmenting the log into traces, each representing an execution instance of the task under recording. Given a set of task traces, it then identifies automatable routines, collects variants of each identified routine, standardizes and streamlines the identified variants, and synthesizes an executable specification for each streamlined and standardized variant. The pipeline is abstract, meaning that any step can have multiple different instantiations. We implemented our own instantiation in the form of an open-source tool called Robidium, which is publicly available.

The literature review conducted in this thesis shows that existing approaches to identify routines assume that the UI log given in input is already segmented, i.e., consists of a set of task traces, each representing an instance of a task execution. In practice, however, a UI log is a single sequence of UIs capturing multiple hours of work. Thus, although the current approaches work in theory, they have no practical applications. Accordingly, the third contribution of this thesis is an approach capable of identifying candidate routines for automation from unsegmented UI logs. The approach consists of two main phases: segmentation of the log into a set of task traces and discovery of the candidate routines. It is also possible to use the segmentation technique separately so that it can be combined with any other discovery technique. The results of the evaluation show that the proposed approach efficiently discovers routines present in the UI logs.

The fourth and final contribution of the thesis is an approach to evaluate the degree of automatability of routines and discover their executable specifications. The proposed approach exploits state-of-the-art data transformation discovery techniques to discover the deterministic functions to compute the values produced during the execution of a routine (e.g., what value should be assigned to a text field or a cell). Using such information, our approach identifies automatable (and non-automatable) UIs and quantifies the degree of routine automatability. The discovered transformation functions together with the UIs of the routine are then used to construct the executable specification. Besides, we propose a technique to identify duplicate routines that lead to identical effects. The technique leverages the produced specifications to compare routines' effects, identifies duplicates, and removes them, leaving the most optimal variant of routine execution.

The approaches presented in the thesis were successfully applied in practice at the University of Melbourne, Australia. Specifically, they were used to discover potentially automatable routines in scholarship assessment and allocation processes within the university. The proposed techniques promise to significantly reduce the time spent on the first phases of a task automation project, in particular the discovery of routines that can be automated, and the automated synthesis of RPA bot scripts. This will allow the project stakeholders to focus their efforts more on the validation of the automation opportunities identified by our approaches, and subsequent refinement and testing of the synthesized RPA scripts, rather than on the actual implementation of the scripts.

8.2. Future work

This thesis has laid the foundations for Robotic Process Mining, a new family of tools to facilitate RPA initiatives. Our research contributions open up a number of directions for future work. The segmentation approach presented in this thesis is designed to deal with logs that capture consecutive executions of routines. However, in practice, it is possible that some routine executions may overlap, e.g., when a user works on multiple tasks in parallel. A possible avenue to address this limitation is to search for overlapping frequent patterns directly in the unsegmented log instead of first segmenting it and then finding patterns in the segmented log. However, this task is very challenging and computationally heavy when the patterns are not exact (i.e., contain gaps between their UIs).

Another limitation of the segmentation approach is that it assumes every task captured in a UI log to have clear start and end points. If such points do not exist, e.g., there is no strictly defined way of performing the underlying task, we may discover incorrect segments, affecting the quality of the discovered routines.

The presented approaches for discovering automatable routines allow the identification of syntactical data transformations and the transformations in the form of substitution mappings. This set of transformations is limited and should be extended. In particular, one possible type of transformations that can be discovered is the transformations where the output value is computed as a result of a mathematical function over the input values (e.g., currency conversion, a sum of the values in a row).

RPM heavily relies on the information present in a UI log and follows the following principle: "Anything that is not captured in the log does not exist." Thus, it may discover imprecise routines when some of their UIs were performed with applications not supported by a logging tool or without explicitly accessing the information (e.g., a worker visually reading the content of a cell and then manually editing the corresponding field of a form with the value that she saw). Therefore, a natural direction for future work is to extend our Action Logger to support more applications. In this regard, one can explore the possibility of using optical character recognition techniques over screenshots taken during the UI log recording to support the applications without available APIs and to track user activities that are not performed explicitly.

The proposed RPM pipeline focuses on discovering 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 specific 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 does not find the 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.

In the context of partial automation, a possible direction for future work is to discover automation blueprints, i.e., step-by-step human-readable descriptions of routines to be automated, that can be used by RPA developers to implement industry-level bots. Such descriptions may be used as specifications and guidelines on how to implement routines, while leaving the decision making to the RPA developers. In this way, the RPA developers can decide, for example, whether the routine should be automated entirely via UI-level automation scripts, or via a combination of UI-level automated steps and API calls, in cases where one of the applications involved in the routine provides suitable APIs. This approach also provides more flexibility when it comes to implementing RPA bots for semi-automatable routines, as the RPA developer can decide how the hand-offs between the automated and the manual part of the routine should be implemented.

The vision of RPM exposed in this thesis focuses on discovering automatable routines, which is only one 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?”.

BIBLIOGRAPHY

[1] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition.* Springer, 2018.

[2] L. Willcocks and M. Lacity, *Service Automation: Robots and the Future of Work.* A Steve Brookes Publishing book, Steve Brookes Publishing, 2016.

[3] W. M. Aalst, M. Bichler, and A. Heinzl, "Robotic process automation," *Business & Information Systems Engineering*, vol. 60, no. 4, pp. 269–272, 2018.

[4] M. Lacity and L. P. Willcocks, "Robotic process automation at telefónica O2," *MIS Quarterly Executive*, vol. 15, no. 1, 2016.

[5] S. Aguirre and A. Rodriguez, "Automation of a business process using robotic process automation (RPA): A case study," in *Proceedings of the 4th Workshop on Engineering Applications (WEA)*, pp. 65–71, Springer, 2017.

[6] L. P. Willcocks, M. Lacity, and A. Craig, "Robotic process automation at xchanging," lse research online documents on economics, London School of Economics and Political Science, LSE Library, 2015.

[7] M. Lacity, L. Willcocks, and A. Craig, "Service automation: cognitive virtual agents at seb bank," *The Outsourcing Unit Working Research Paper Series*, 2017.

[8] L. P. Willcocks, M. Lacity, and A. Craig, "Robotizing global financial shared services at royal dsm," *Journal of Financial Transformation*, no. Automation# 46, 2017.

[9] M. Schmitz, C. Dietze, and C. Czarnecki, "Enabling digital transformation through robotic process automation at deutsche telekom," in *Digitalization cases*, pp. 15–33, Springer, 2019.

[10] A. M. Radke, M. T. Dang, and A. Tan, "Using robotic process automation (rpa) to enhance item master data maintenance process," *LogForum*, vol. 16, no. 1, 2020.

[11] B. Agaton and G. Swedberg, "Evaluating and developing methods to assess business process suitability for robotic process automation: A design research approach," Master's thesis, Chalmers University of Technology, 2018.

[12] R. H. Von Alan, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[13] C. Tornbohm, "Gartner market guide for robotic process automation software," Report G00319864, Gartner, 2017.

[14] W. M. Aalst, *Process Mining - Data Science in Action, Second Edition.* Springer, 2016.

[15] I. Verenich, M. Dumas, M. L. Rosa, F. M. Maggi, and I. Teinemaa, "Survey and cross-benchmark comparison of remaining time prediction methods

in business process monitoring," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 4, pp. 34:1–34:34, 2019.

[16] I. Teinemaa, M. Dumas, M. L. Rosa, and F. M. Maggi, "Outcome-oriented predictive process monitoring: Review and benchmark," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 2, pp. 17:1–17:57, 2019.

[17] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *IEEE Trans. Kn. Data Eng.*, vol. 31, no. 4, pp. 686–705, 2019.

[18] M. Kerremans and T. Srivastava, "Discover the differences and use cases of process mining versus task mining," Research Note G00723821, Gartner, April 2020.

[19] R. Syed, S. Suriadi, M. Adams, W. Bandara, S. J. J. Leemans, C. Ouyang, A. H. M. ter Hofstede, I. van de Weerd, M. T. Wynn, and H. A. Reijers, "Robotic process automation: Contemporary themes and challenges," *Comput. Ind.*, vol. 115, p. 103162, 2020.

[20] J. Geyer-Klingeberg, J. Nakladal, F. Baldauf, and F. Veit, "Process mining and robotic process automation: A perfect match," in *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018*, pp. 124–131, CEUR-WS.org, 2018.

[21] V. Leno, A. Polyvyanyy, M. Dumas, M. L. Rosa, and F. M. Maggi, "Robotic process mining: Vision and challenges," *Business & Information Systems Engineering*, 2020.

[22] V. Leno, A. Polyvyanyy, M. L. Rosa, M. Dumas, and F. M. Maggi, "Action logger: Enabling process mining for robotic process automation," in *Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019*, vol. 2420 of *CEUR Workshop Proceedings*, pp. 124–128, CEUR-WS.org, 2019.

[23] J. M. López-Carnicer, C. D. Valle, and J. G. Enríquez, "Towards an open-source logger for the analysis of RPA projects," in *Business Process Management: Blockchain and Robotic Process Automation Forum - BPM 2020 Blockchain and RPA Forum, Seville, Spain, September 13-18, 2020, Proceedings*, vol. 393 of *Lecture Notes in Business Information Processing*, pp. 176–184, Springer, 2020.

[24] M. Spiliopoulou, B. Mobasher, B. Berendt, and M. Nakagawa, "A framework for the evaluation of session reconstruction heuristics in web-usage analysis," *Informs journal on computing*, vol. 15, no. 2, pp. 171–190, 2003.

[25] J. Shen, L. Li, and T. G. Dietterich, "Real-time detection of task switches of desktop users," in *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pp. 2868–2873, 2007.

[26] 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 *Proceedings of the 10th International Conference on Intelligent User Interfaces, IUI 2005, San Diego, California, USA, January 10-13, 2005*, pp. 75–82, 2005.

[27] H. Cao, N. Mamoulis, and D. W. Cheung, "Discovery of periodic patterns in spatiotemporal sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 4, pp. 453–467, 2007.

[28] Y. Zhu, M. Imamura, D. Nikovski, and E. Keogh, "Matrix profile vii: Time series chains: A new primitive for time series data mining," in *2017 IEEE International Conference on Data Mining (ICDM)*, pp. 695–704, IEEE, 2017.

[29] D. R. Ferreira and D. Gillblad, "Discovering process models from unlabelled event logs," in *Proceedings of the 7th International Conference on Business Process Management (BPM)*, pp. 143–158, Springer, 2009.

[30] D. Bayomie, C. Di Ciccio, M. La Rosa, and J. 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.

[31] D. Bayomie, A. Awad, and E. Ezat, "Correlating unlabeled events from cyclic business processes execution," in *Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAiSE)*, pp. 274–289, Springer, 2016.

[32] S. J. van Zelst, F. Mannhardt, M. de Leoni, and A. Koschmider, "Event abstraction in process mining: literature review and taxonomy," *Granular Computing*, vol. 6, no. 3, pp. 719–736, 2021.

[33] B. Fazzinga, S. Flesca, F. Furfaro, E. Masciari, and L. Pontieri, "Efficiently interpreting traces of low level events in business process logs," *Information Systems*, vol. 73, pp. 1–24, 2018.

[34] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. van der Aalst, and P. J. Toussaint, "Guided process discovery–a pattern-based approach," *Information systems*, vol. 76, pp. 1–18, 2018.

[35] G. Tello, G. Gianini, R. Mizouni, and E. Damiani, "Machine learning-based framework for log-lifting in business process mining applications," in *Business Process Management - 17th International Conference, BPM 2019, Vienna, Austria, September 1-6, 2019, Proceedings*, vol. 11675 of *Lecture Notes in Computer Science*, pp. 232–249, Springer, 2019.

[36] M. de Leoni and S. Dündar, "Event-log abstraction using batch session identification and clustering," in *SAC '20: The 35th ACM/SIGAPP Symposium on Applied Computing, online event, [Brno, Czech Republic], March 30 - April 3, 2020*, pp. 36–44, ACM, 2020.

[37] C. Linn, P. Zimmermann, and D. Werth, “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, 2018.

[38] A. Rebmann, J.-R. Rehse, M. Pinter, M. Schnaubelt, K. Daun, and P. Fettke, “Iot-based activity recognition for process assistance in human-robot disaster response,” in *International Conference on Business Process Management*, pp. 71–87, Springer, 2020.

[39] S. Agostinelli, “Automated segmentation of user interface logs using trace alignment techniques (extended abstract),” in *Proceedings of the ICPM Doctoral Consortium and Tool Demonstration Track 2020* (C. D. Ciccio, B. Depaire, J. D. Weerdt, C. D. Francescomarino, and J. Munoz-Gama, eds.), vol. 2703 of *CEUR Workshop Proceedings*, pp. 13–14, CEUR-WS.org, 2020.

[40] H. Dev and Z. Liu, “Identifying frequent user tasks from application logs,” in *Proceedings of IUI 2017*, pp. 263–273, Springer, 2017.

[41] J. Han, H. Cheng, D. Xin, and X. Yan, “Frequent pattern mining: current status and future directions,” *Data mining and knowledge discovery*, vol. 15, no. 1, pp. 55–86, 2007.

[42] S. D. Lee and L. De Raedt, “An efficient algorithm for mining string databases under constraints,” in *International Workshop on Knowledge Discovery in Inductive Databases*, pp. 108–129, Springer, 2004.

[43] E. Ohlebusch and T. Beller, “Alphabet-independent algorithms for finding context-sensitive repeats in linear time,” *Journal of Discrete Algorithms*, vol. 34, pp. 23–36, 2015.

[44] J. Wang and J. Han, “Bide: Efficient mining of frequent closed sequences,” in *Proceedings of the 20th international conference on data engineering*, pp. 79–90, IEEE, 2004.

[45] F. Fumarola, P. F. Lanotte, M. Ceci, and D. Malerba, “Clofast: closed sequential pattern mining using sparse and vertical id-lists,” *Knowledge and Information Systems*, vol. 48, no. 2, pp. 429–463, 2016.

[46] A. Jimenez-Ramirez, H. A. Reijers, I. Barba, and C. Del Valle, “A method to improve the early stages of the robotic process automation lifecycle,” in *International Conference on Advanced Information Systems Engineering*, pp. 446–461, Springer, 2019.

[47] D. Choi, H. R’bigui, and C. Cho, “Candidate digital tasks selection methodology for automation with robotic process automation,” *Sustainability*, vol. 13, no. 16, p. 8980, 2021.

[48] M. de Leoni, M. Dumas, and L. García-Bañuelos, “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, 2013.

[49] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. 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*, pp. 545–560, Springer, 2017.

[50] H. Leopold, H. van der Aa, and H. A. Reijers, “Identifying candidate tasks for robotic process automation in textual process descriptions,” in *Proceedings of BPMDS and EMMSAD*, pp. 67–81, Springer, 2018.

[51] W. M. van der Aalst, “On the pareto principle in process mining, task mining, and robotic process automation.,” in *DATA*, pp. 5–12, 2020.

[52] A. Bosco, A. Augusto, M. Dumas, M. L. Rosa, and G. Fortino, “Discovering automatable routines from user interaction logs,” in *Proceedings of the Business Process Management Forum (BPM Forum)*, Springer, 2019.

[53] T. Chakraborti, V. Isahagian, R. Khalaf, Y. Khazaeni, V. Muthusamy, Y. Rizk, and M. Unuvar, “From robotic process automation to intelligent process automation,” in *International Conference on Business Process Management*, pp. 215–228, Springer, 2020.

[54] S. 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*, pp. 317–330, 2011.

[55] D. W. Barowy, S. Gulwani, T. Hart, and B. 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*, pp. 218–228, 2015.

[56] N. Martin, B. Depaire, and A. Caris, “The use of process mining in business process simulation model construction - structuring the field,” *Business & Information Systems Engineering*, vol. 58, no. 1, pp. 73–87, 2016.

[57] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker, “Dataxformer: A robust transformation discovery system,” in *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pp. 1134–1145, 2016.

[58] Z. Jin, M. R. Anderson, M. J. Cafarella, and H. V. Jagadish, “Foofah: Transforming data by example,” in *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pp. 683–698, 2017.

[59] J. Gao, S. J. van Zelst, X. Lu, and W. M. van der Aalst, “Automated robotic process automation: A self-learning approach,” in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pp. 95–112, Springer, 2019.

[60] S. Agostinelli, M. Lupia, A. Marrella, and M. Mecella, “Automated generation of executable RPA scripts from user interface logs,” in *Business Process Management: Blockchain and Robotic Process Automation Forum - BPM 2020*, vol. 393 of *Lecture Notes in Business Information Processing*, pp. 116–131, Springer, 2020.

[61] S. Agostinelli, M. Lupia, A. Marrella, and M. Mecella, “Smartrpa: A tool to reactively synthesize software robots from user interface logs,” in *International Conference on Advanced Information Systems Engineering*, pp. 137–145, Springer, 2021.

[62] V. Leno, A. Augusto, M. Dumas, M. L. Rosa, F. M. Maggi, and A. 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*, pp. 153–160, IEEE, 2020.

[63] V. Leno, A. Augusto, M. Dumas, M. La Rosa, F. M. Maggi, and A. Polyvyanyy, “Discovering data transfer routines from user interaction logs,” *Information Systems*, p. 101916, 2021.

[64] M. Sharir, “A strong-connectivity algorithm and its applications in data flow analysis,” *Computers & Mathematics with Applications*, vol. 7, no. 1, pp. 67–72, 1981.

[65] R. Tarjan, “Depth-first search and linear graph algorithms,” *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[66] N. Tax, N. Sidorova, R. Haakma, and W. M. P. van der Aalst, “Mining local process models,” *J. Innov. Digit. Ecosyst.*, vol. 3, no. 2, pp. 183–196, 2016.

[67] V. Leno, M. Dumas, M. L. Rosa, F. M. Maggi, and A. Polyvyanyy, “Automated discovery of data transformations for robotic process automation,” *ArXiv*, vol. abs/2001.01007, 2020.

[68] V. Raman and J. M. Hellerstein, “Potter’s wheel: An interactive data cleaning system,” in *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases*, pp. 381–390, 2001.

[69] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, “TANE: an efficient algorithm for discovering functional and approximate dependencies,” *Comput. J.*, vol. 42, no. 2, pp. 100–111, 1999.

[70] V. Leno, S. Deviatykh, A. Polyvyanyy, M. L. Rosa, M. Dumas, and F. M. Maggi, “Robidium: Automated synthesis of robotic process automation scripts from UI logs,” in *Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track at BPM 2020*, vol. 2673 of *CEUR Workshop Proceedings*, pp. 102–106, CEUR-WS.org, 2020.

[71] C. W. Günther and E. Verbeek, “XES standard definition,” 2014. http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf.

ACKNOWLEDGEMENT

I am genuinely grateful to everyone who has encouraged and helped me to write this thesis. First of all, I would like to thank my supervisors, Marlon Dumas, Marcello La Rosa, Artem Polyvyanyy, and Fabrizio Maria Maggi, for their ideas, feedback, and guidance. Second, I am grateful to the University of Tartu and the University of Melbourne for their financial support and outstanding educational base. I would also like to thank Stanislav Deviatykh, Andre McGuire, and Iliyar Yi for their help with the implementation. Finally, this would not have been possible without my family. Thank you for your encouragement, support, and belief in me, no matter what.

SISUKOKKUVÕTE

Robot Protsesside Kaevandamine: Robot Protsesside Automatiseerimise kasutuselevõtu kiirendamine

Tänapäeval koosnevad suur enamus äriprotsessidest üksluisetest ja korduvatest tegevustest. Nendeks võivad olla andmete viimine läbi erinevate süsteemide või andmete tõstmine ühest formaadist teise. Selliste ülesannete automatiseerimine võiks vähendada vigasid, tõsta jõudlust ja standardiseerida töövoosid ning tõsta nende läbipaistvust.

Korduvate tegevuste automatiseerimise muudavad lihtsamaks viimase aja tehnoloogilised arengud, peamiselt aga Robootiline Protsesside Automatiseerimine (Robotic Process Automation, RPA). RPA on tehnoloogia, mis võimaldab organisatsioonidel automatiseerida korduvaid digitaalseid tegevusi käivitades tarkvaraskripte, mis koosnevad interaktsioonidest (veebi)rakendustega.

Kuigi RPA annab võimaluse automatiseerida mitmeid erinevaid protseduure, on nende tuvastamine ja piiritlemine ajamahukas töö. Analüütikud võivad identifitseerida kandidaatprotseduure intervjuude, ülesannete proovimise või töö varjutamise abil. Need meetmed ei ole aga sobivad suuremahuliste kontekstide jaoks.

Väitekiri käsitleb, kuidas RPA tööriistade abil leida protseduure interaktsioonilogidest. Antud probleem on jagatud alamprobleemide jadaks, kuhu kuuluvad logide kogumine ja eeltöötlemine, kandidaatrutiinide tuvastamine, protseduuride hindamine, et mõista automatiseerimise võimalikkust ja täide viivate spetsifikatsioonide koostamine protseduuride elluviimiseks. Väitekiri analüüsib iga alamprobleemi ja toob välja valdkonnad mida saab parandada.

Tehtud analüüsi põhjal esitleb väitekiri nelja panust. Esimeseks panuseks on formaat milles ladustada sündmuste interaktsioonilogisid ja tööriist nende salvestamiseks. Olemasolevate lahenduste analüüs näitas, et need ei suuda salvestada interaktsioonilogisid mida saaks kasutada kasulike rutiinide avastamiseks. Nad salvestavad kasutajate toiminguid väga granulaarsel tasemel, klahvivajutused ja lehed, mida kasutajad külastavad, jättes välja andmed mis luuakse nende interaktsioonide jooksul. Väitekirjas esitletud formaat kirjeldab millist informatsiooni logimistööriist peab salvestama, et analüüsida, parandada ja automatiseerida kasutajate interaktsioone. Kirjeldatud logimistööriist töötab Chrome ja MS Excel rakendustes. See salvestab kasutajate sisendeid rakendusse loogilisel tasemel. Tööriist salvestab andmed, mida kasutaja sisestab, et tuletatud protseduure oleks võimalik analüüsida automatiseerimise seisukohast ja luua nende jaoks spetsifikatsioonid.

Teiseks panuseks on läbitöötatud protsess rutiinsete tegevuste automatiseerimiseks. Loodud protsess kaardistab automatiseeritavad protseduurid sündmuste logi põhjal. Lisaks, protseduurid piiritletakse ning iga protseduuri eri variandid kogutakse kokku, mis standardiseeritakse ja muudetakse efektiivseks. Lõpuks avastatakse protseduuri variantidele spetsifikatsioonid. Protseduurid kirjutatakse keeles, mis on platvormidest iseseisev, ja mida saab kompileerida skriptiks ja käivitada RPA tööriistas.

Kaks viimast panust on meetodid pakutud protsessile. Üks neist aitab avastada kandidaatprotseduurid RPA-le sündmuste logist. Logi jagatakse lõikudeks kasutades graafiteooria meetodeid. Iga lõik esindab konkreetset töö teostust. Lõikudest kaevandatakse mustrid, mis esindavad potentsiaalseid protseduure. Teine meetod kontsentreerub protseduuride analüüsimisele automatiseerimise seisukohast ja neile spetsifikatsioonide loomisele. Antud meetod kasutab moodsaid andmete transformatsioonide avastamise tehnikaid ja esitleb mitmeid parandusi nende toimimisse ja kvaliteeti. Samuti esitleb väitekiri meetodit, kuidas leida ja eemaldada protseduure mis on semantiliselt samaväärsed.

Kirjeldatud meetodid loovad uue meetodite perekonna, mida saab nimetada Robootiliseks Protsesside Kaevandamiseks (Robotic Process Mining RPM). RPM eesmärk on aidata RPA arendajaid ja analüütikuid RPA elutsükli alguses. RPM abil saavad analüütikud kaardistada tegevused mida saab RPA abil automatiseerida ja sünteesida spetsifikatsioonid tegevustele mida saab automatiseerimisprotsesside lähtepunktina kasutada.

Kõik väitekirja tulemid on avalikult kättesaadavad käsurea rakendused avatud lähtekoodiga. Nad on koondatud tarkvara teenuse põhilisse tööriista, mille nimi on Robidium. Kavandatud meetodite tõhusust ja potentsiaalset kasulikkust on hinnatud sünteetiliste ja tegelike kasutajate interaktsioonilogide abil. Eksperimendid näitasid, et need meetodid suudavad leida automatiseeritavaid protseduure elulisetest interaktsioonilogidest, mida kasutajad tunnistavad omast kogemustest.

CURRICULUM VITAE

Personal data

Name:	Volodymyr Leno
Date of Birth:	17.10.1994
Citizenship:	Ukrainian
Language:	Ukrainian, English

Education

2018–2021	joint doctor of philosophy program in computer science – University of Tartu and University of Melbourne
2015–2017	master's degree in software engineering – University of Tartu
2011–2015	bachelor's degree in computer science – Lviv Polytechnic University

Employment

2021–present day	software developer – Apromore
2021	research support – University of Melbourne
2016–2017	researcher and analyst – Minitlabs

Scientific work

Main fields of interest:

- robotic process automation
- process mining
- task mining

ELULOOKIRJELDUS

Isikuandmed

Nimi:	Volodymyr Leno
Sünniaeg:	17.10.1994
Kodakondsus:	Ukrainlane
Keelteoskus:	ukrainlane, inglise

Haridus

2018–2021	Tartu Ülikooli ja Melbourne'i Ülikooli ühine doktoriõpe informaatika erialal
2015–2017	Tartu Ülikooli, tarkvaratehnika magistriõpe
2011–2015	Lviv Polütehniline Ülikooli, informaatika bakalaureuseõpe

Teenistuskäik

2021–tänapäev	Apromore, tarkvara arendaja
2021	Melbourne'i Ülikooli, teadusuuringute tugi
2016–2017	Minitlabs, teadlane ja analüütik

Teadustegevus

Peamised uurimisvaldkonnad:

- robotprotsesside automatiseerimine
- protsessi kaevandamine
- ülesannete kaevandamine

LIST OF ORIGINAL PUBLICATIONS

Publications in the scope of the thesis

I **Leno, V.** (2018). Multi-perspective process model discovery for Robotic Process Automation. In *CEUR Workshop Proceedings*, vol. 2114, pp. 37-45.

II **Leno, V.**, Polyvyanyy, A., La Rosa, M., Dumas, M., & Maggi, F. M. (2019). Actiong logger: Enabling Process Mining for Robotic Process Automation. In *Proc. of the Dissertation Award, Doctoral Consortium, and Demonstration Track of the International Conference on Business Process Management (BPM) 2019*, pp. 124-128.

III **Leno, V.**, Dumas, M., La Rosa, M., Maggi, F. M., & Polyvyanyy, A. (2020) Automated Discovery of Data Transformations for Robotic Process Automation. In *Proc. of the AAAI Workshop on Intelligent Process Automation (IPA) 2020*. [**Awarded as the Best Paper**]

IV **Leno, V.**, Augusto, A., Dumas, M., La Rosa, M., Maggi, F. M., & Polyvyanyy, A. (2020). Identifying candidate routines for Robotic Process Automation from unsegmented UI logs. In *International Conference on Process Mining (ICPM) 2020*, pp. 153-160, IEEE.

V **Leno, V.**, Deviatykh, S., Polyvyanyy, A., La Rosa, M., Dumas, M., & Maggi, F. M. (2020) Robidium: Automated Synthesis of Robotic Process Automation Scripts from UI logs. In *Proc. of the Dissertation Award, Doctoral Consortium, and Demonstration Track of the International Conference on Business Process Management (BPM) 2020*. [**Awarded as the Best Demo Paper**]

VI **Leno, V.**, Polyvyanyy, A., Dumas, M., La Rosa, M., & Maggi, F. M. (2021). Robotic Process Mining: Vision and Challenges. *Business and Information Systems Engineering*, pp. 1-14, Springer.

VII **Leno, V.**, Augusto, A., Dumas, M., La Rosa, M., Maggi, F. M., & Polyvyanyy, A. (2021). Discovering data transfer routines from user interaction logs. *Information Systems*, p. 101916.

Publications out of the scope of the thesis

I **Leno, V.**, Dumas., M., & Maggi, F. M. (2018). Correlating activation and target conditions in data-aware declarative process discovery. In *International Conference on Business Process Management (BPM) 2018*, pp. 176-193. Springer.

II **Leno, V.**, Dumas, M., Maggi, F. M., La Rosa, M., & Polyvyanyy, A. (2020). Automated discovery of declarative process models with correlated data conditions. *Information Systems*, 89, p. 101482.

DISSERTATIONES INFORMATICAE PREVIOUSLY PUBLISHED IN DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS