Automated Generation of Executable RPA Scripts from User Interface Logs

Simone Agostinelli, Marco Lupia, Andrea Marrella, and Massimo Mecella

Sapienza Universitá di Roma, Rome, Italy
lupia.1694700@studenti.uniroma1.it
{agostinelli,marrella,mecella}@diag.uniroma1.it

Abstract. Robotic Process Automation (RPA) operates on the user interface (UI) of software applications and automates - by means of a software (SW) robot - mouse and keyboard interactions to remove intensive routine tasks (or simply *routines*). With the recent advances in Artificial Intelligence, the automation of routines is expected to undergo a radical transformation. Nonetheless, to date, the RPA tools available in the market are not able to automatically learn to automate such routines, thus requiring the support of skilled human experts that observe and interpret how routines are executed on the UIs of the applications. Being the current practice time-consuming and error-prone, in this paper we present SmartRPA, a cross-platform tool that tackles such issues by exploiting UI logs to automatically generate executable RPA scripts that automate the routines enactment by SW robots.

Keywords: Robotic Process Automation (RPA) · Automated RPA script generation · User Interface (UI) Logs · Process Mining

1 Introduction

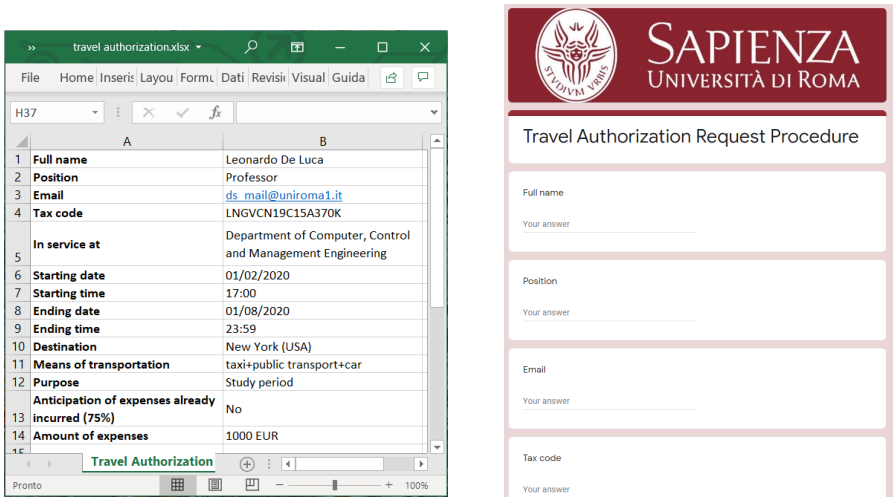
Robotic Process Automation (RPA) is a fast-emerging automation technology in the field of Business Process Management (BPM) that uses software (SW) robots to mimic and replicate the execution of highly repetitive routine tasks (we refer to them as *routines*) performed by human users in their applications' user interfaces (UIs). The RPA technology is still in its infancy [1], even if similar solutions have been around for a long time. For instance, closely related to SW robots, chatbots have been using for years to accept voice-based or keyboard inputs and guide customers to find relevant information in web-based applications [14]. Differently from chatbots, RPA can be seen as an evolution of screen scraping solutions [9], which sought to visualize screen display data from legacy applications (having no means for automated interfacing) to display such data using modern UIs. The strength of RPA is that it does not replace existing applications or manipulate their code, but rather works with them in a way similar to a human user.
In recent years there was an increased interest around RPA, resulting in many industry-specific deployments for financial and business services [19,5,12]. In this direction, according to [6], the market of RPA solutions has developed rapidly

and today includes more than 50 vendors developing tools that provide SW robots with advanced functionalities for automating office tasks in operations like accounting, billing and customer service. Nonetheless, when considering state-of-the-art RPA technology, it becomes apparent that the current generation of RPA tools is driven by predefined rules and manual configurations made by expert users rather than automated techniques [3]. To be more specific, the traditional workflow to conduct a RPA project can be summarized as follows [18]:

1. Determine which routines are good candidates to be automated.
2. Record the mouse/key events that happen on the UI of the SW applications involved in a routine execution, i.e., the *UI logs*.
3. Model the selected routines in the form of flowchart diagrams, which involve the specification of the actions, routing constructs (e.g., parallel and alternative branches), data flow, etc. that define the behavior of a SW robot.
4. Develop each modeled routine by generating the SW code required to concretely enact the associated SW robot on a target computer system.
5. Deploy the SW robots in their environment to perform their actions.
6. Monitor the performance of SW robots to detect bottlenecks and exceptions.
7. Maintain the routines, which takes into account the SW robots performance and error cases to eventually enhance their behaviour.

The majority of the previous steps, particularly the ones involved in the early stages of the RPA life-cycle (i.e., steps 1 and 3), require the support of skilled human experts, which need to: *(i)* understand the anatomy of the candidate routines to automate by means of interviews, walk-troughs, and detailed observation of workers conducting their daily work; and *(ii)* define manually the flowchart diagrams representing the structure of such routines, which will drive the development of the SW code, often in form of executable scripts (also called *RPA scripts*), allowing the concrete enactment of SW robots at run-time (cf. step 4). While this approach is effective to execute simple rules-based logic in situations where there is no room for interpretation, it becomes time-consuming and error-prone in presence of routines that are less predictable or require some level of human judgment [25,4]. Indeed, the designer should have a global vision of all possible variants of the routines to define the appropriate behaviours of the SW robot, which becomes complicated when the number of variants increases. The issue is that in case where the flowchart diagram does not contain a suitable response for a specific situation, e.g., because of a shallow modeling activity, then the associated RPA scripts would not properly reflect the behaviour of the potential routine variant, forcing SW robots to escalate to a human supervisor at run-time, in contrast with the RPA philosophy.

To tackle and mitigate this issue, in this paper we develop a cross-platform software tool, called SmartRPA, to automatically generate executable RPA scripts directly from the UI logs that record the user interactions with the SW applications involved in a routine execution (cf. step 2), thus skipping completely the (manual) modeling activity of the flowchart diagrams (cf. step 3). SmartRPA involves five consecutive stages that enable to: *(i)* record the UI logs that keep



(a) Excel spreadsheet (b) Google form

Fig. 1: UIs involved in the running example

track of the different routine executions on the UIs of the involved SW applications; *(ii)* processing such UI logs in form of a single *event log* with additional execution properties; *(iii)* filtering out those events not relevant for the routine of interest and grouping together similar events; *(iv)* detecting the most frequent routine variant from the log, leveraging process discovery and abstraction techniques; and *(v)* generating the executable RPA scripts necessary to enact the SW robot that implements the selected routine variant. SmartRPA is available for download at https://github.com/bpm-diag/smartRPA/.

The rest of the paper is organized as follows. Section 2 presents a motivating running example. In Section 3, we analyze the architecture and the technical aspects of SmartRPA, together with the approach underlying the working of the tool. Section 4 examines the instantiation of SmartRPA on the running example. Finally, in Section 5 we present the related works, while Section 6 concludes the paper by discussing the weaknesses of the tool and the potential future works.

2 Running Example

Below, we describe an RPA use case inspired by a real-life scenario at Department of Computer, Control and Management Engineering (DIAG) of Sapienza Universitá di Roma. The scenario concerns the filling of the travel authorization request form made by professors, researchers and PhD students of DIAG for travel requiring prior approval.

The request applicant must fill a well-structured Excel spreadsheet (cf. Figure 1(a)) providing some personal information, such as her/his bio-data and the

email address, together with further information related to the travel, including the destination, the starting/ending date/time, the means of transport to be used, the travel purpose, and the envisioned amount of travel expenses, associated with the possibility to request an anticipation of the expenses already incurred (e.g., to request in advance a visa). When ready, the spreadsheet is sent via email to an employee of the Administration Office of DIAG, which is in charge of approving it and (only in this case) elaborating the request. Concretely, for each row in the spreadsheet, the employee manually copies every cell in that row and pastes that into the corresponding text field in a dedicated Google form (cf. Figure 1(b)), accessible just by the Administration staff. Once the data transfer for a given travel request has been completed, the employee presses the "Submit" button to submit the data into an internal database. Once the form is submitted, a confirmation email is sent automatically to the applicant.
The above routine procedure is usually performed manually, it is tedious (as it must be repeated for any new travel request) and prone to errors. We will use it to show how the proposed SmartRPA tool is able to automatically develop the executable RPA scripts for automating the data transfer task of the routine, requiring in input just the UI logs that record the previous executions of such routine performed by several human users during dedicated training sessions.

3 SmartRPA Approach and Architecture

The architecture of SmartRPA integrates five main SW components developed in Python that enable to automatically generate executable RPA scripts that will drive the working of SW robots in emulating the users' observed behavior (previously recorded in dedicated UI logs) during the enactment of a routine of interest. An overview of the SmartRPA architecture is shown in Figure 2.

The first SW component of the architecture is an **Action Logger** that can be used to record a wide range of UI actions from multiple SW applications during the enactment of a routine. This means that SmartRPA belongs to the category of those RPA tools that learn to automate routines "by examples" (see also our discussion in Section 5). To be more specific, a training session in which several users perform the routine to be automated is required to record the UI actions involved in its execution. To this aim, the Action Logger provides a Graphical User Interface (GUI) that allows a user to select which SW applications s/he wants to record UI actions on. All the applications that are not available in the host operating system of the user's PC/MAC are disabled by default. Then, the user can start the training session by clicking on the *"Start logger"* button (see Figure 3). The Action Logger provides three categories of logging modules:

- *System Logger*: It detects those UI actions not related to specific SW applications, i.e.: copy and paste of files/folders; creation, renaming, movement and deletion of files/folders; usage of double-click and hotkeys; opening and closing of applications; printing activities; insertion/remotion of USB drives.
- *Office Logger*: It detects the UI actions performed within Microsoft Office applications, i.e.: Excel, Word and PowerPoint.

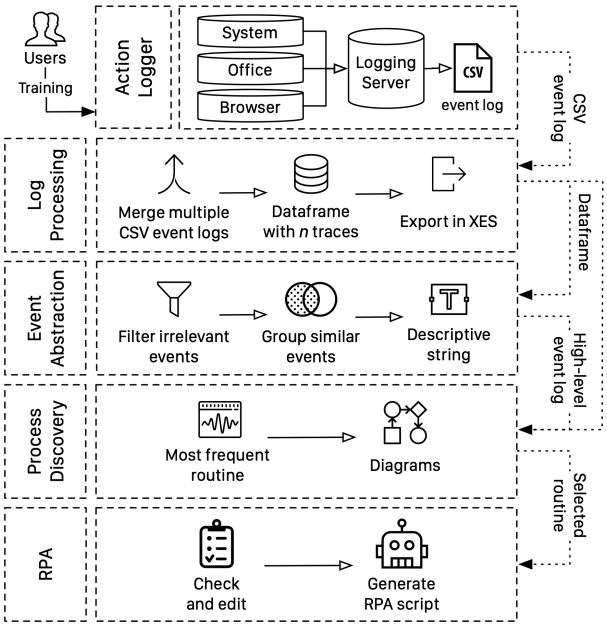


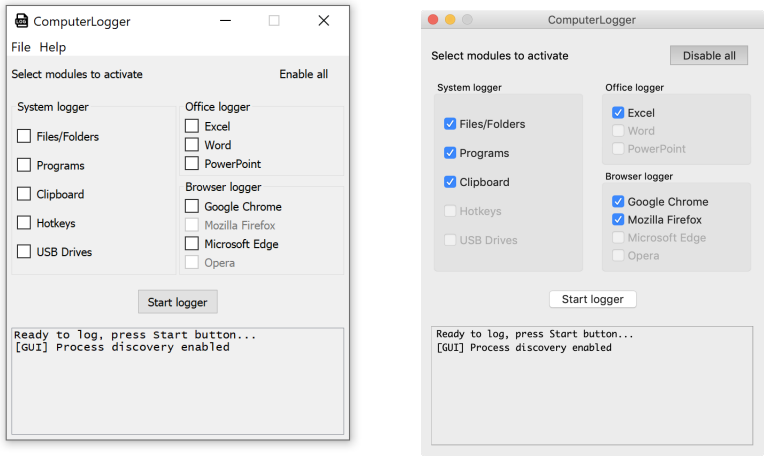
Fig. 2: SmartRPA architecture

– *Browser Logger*: It detects the UI actions performed on web browsers, i.e.: Google Chrome, Mozilla Firefox, Microsoft Edge and Opera.

Of course, multiple users can run the Action Logger on their PC/MAC many times performing the same routine in different training sessions. When a training session is completed, i.e., when the routine of interest has been executed from the start to the end, the user can push the *"Stop logger"* button to stop the recording of UI actions. The logging modules interact with a Logging Server implemented with the *Flask* framework,[1] which is in charge to store the UI actions captured by the logging modules and organize them as *events* into several CSV event logs. Each CSV event log contains exactly one (long) trace of UI actions performed in a single training session by a single user. From a technical point of view, *(i)* system events are recorded using different Python modules, including *PythonCOM* (to access the Windows APIs and COM objects like the Microsoft Office suite), and *MacFSEvents* for MacOS; *(ii)* events generated by Microsoft Office applications are recorded using the Office JavaScript APIs; and *(iii)* browser events are recorded using dedicated JavaScript web extensions developed for each supported web browser.

The second SW component of the architecture is a **Log Processing tool** that comes into play when any training session is considered as completed. Specifically, after n training sessions, the Logging Server will deliver the n created CSV

[1] https://palletsprojects.com/p/flask



(a) Windows (b) MacOS

Fig. 3: GUI of SmartRPA both on Windows and MacOS

event logs to the Log Processing tool, which uses Algorithm 1 to import them into a single Pandas dataframe.[2] A dataframe is a two-dimensional size-mutable and heterogeneous tabular data structure with labeled axes (rows and columns), which is used as the main artifact to represent event logs in SmartRPA. Of course, SmartRPA also produces an XES[3] (eXtensible Event Stream) version of the datastream, which will contain exactly n traces, one for each recorded CSV

Algorithm 1 Processing event logs

procedure HANDLELOG(*file_list*)
  create_directories() ▷ where files will be saved
  for any CSV log in *file_list* **do**
    df ← import a CSV log into Pandas dataframe
    df ← rename columns to match XES standard
    df ← sort rows by timestamp
    df ← create *case:concept:name* column based on the first timestamp
    df ← generate a dataframe including the UI actions of the CSV log
  end for
  combined_df ← combine all dataframes into a single dataframe
  export(combined_df) ▷ exported as XES file
end procedure

[2] https://pandas.pydata.org/
[3] XES is the standard for the storage, interchange, and analysis of event logs [15]

case:concept:name	time:timestamp	org:resource	category	application	concept:name
429102859961	2020-04-29T10:29:33.887	marco	Office	Excel	editCell
429102859961	2020-04-29T10:29:34.583	marco	Browser	Chrome	mouseClick
429102859961	2020-04-29T10:29:35.401	marco	Browser	Chrome	changeField
429102859961	2020-04-29T10:29:36.119	marco	Clipboard	Chrome	paste

Table 1: A partial view of a dataframe

event log, and can be inspected using the most popular process mining tools, such as *ProM*,[4] *Disco*[5] or *Apromore*.[6]

The dataframe created by Algorithm 1 consists of low-level events with fine granularity associated one-by-one to a recorded UI action (e.g., mouse clicks, file selections, etc.). Each row of the dataframe includes 45 columns with relevant data about the recorded event, such as: the timestamp, the application that generated the event, the resources involved, etc. A partial view of a dataframe, describing only the first 6 columns recorded for each event, is shown in Table 1.

At this point, an **Event Abstraction component** is used to convert the low-level dataframe recording the event log (that will be used later for generating the excutable RPA scripts) into a high-level one to be exploited for diagnostic and analysis purposes by expert RPA analysts. In particular, the high-level event log can be used to derive the flowchart representing the abstract workflow underlying the routine execution. Specifically, the Event Abstraction component performs the following steps to produce a high-level event log:

1. *Filtering irrelevant events.* The Action Logger records many low-level events in the dataframe-based event log, such as the interaction with the browser windows (e.g., UI actions "resize", "open", "close"), tabs (e.g., UI actions "move", "open", "close") and content (page zoom, installing extensions). From a workflow perspective, these events are not relevant for any RPA analyst that aims to understand the general behaviour of the routine. For this reason, they are filtered out by the high-level event log under construction.
2. *Grouping similar events.* Within a dataframe-based event log, different low-level events can refer to the same high-level concept. For example, in a web page, the Action Logger can capture 7 different types of clicks, based on the element that's being clicked ("clickButton", "clickTextField", "doubleClick", "clickTextField", "mouseClick", "clickCheckboxButton", "clickRadioButton"). All these events just indicate that the user, during the training session, has clicked on an interactive element on the UI, thus the high-level workflow of the routine may just show the action "Click on button", because from the RPA analyst perspective it is not relevant what kind of click was performed.

[4] http://www.promtools.org/
[5] https://fluxicon.com/disco/
[6] https://apromore.org/

3. *Creating descriptive labels.* Any recorded event provides a low-level description of the nature of the UI action performed. For example, if the user edits a cell in Excel, the Action Logger records one of these events: "editCellSheet", "editCell", or "editRange". From the RPA analyst perspective, all such events refer to the same concept of "Editing a cell". To this aim, to make the UI action underlying an event more descriptive for the RPA analyst, further information (stored in the low-level dataframe-based event log) can be added to its label, such as the cell and the sheet edited, the value inserted, etc. This allows us to create a (more) descriptive label for any event in the high-level event log, e.g., "*Edit cell B12 on Sheet 2 with value 'test'*".

Concretely, the Event Abstraction component is realized enacting the above steps through Algorithm 2, and the outcome will be an high-level event log to be used by the next component of the architecture.

Algorithm 2 Event Abstraction

```
procedure getHighLevelEvents(dataframe)
    df ← filter irrelevant rows from the dataframe
    df ← group similar events in the the dataframe
    for row in df do
        descriptive_row ← create descriptive string for each event
    end for
    return a high-level dataframe-based event log
end procedure
```

At this point, the **Process Discovery** component of the architecture has a twofold objective:

- It takes in input the high-level event log generated by the Event Abstraction component and applies the heuristic miner algorithm implemented in PM4PY [8] to derive the high-level workflow describing the overall users' observed behavior as a Directly-Follows Graph (DFG). This flowchart can be analyzed by an RPA analyst to investigate the high-level structure of the routine under analysis. The decision to employ the heuristic miner has been driven by its ability to discover highly understandable flowcharts from a BPM analyst perspective [2].
- It selects the most frequent routine variant among all the different execution traces stored in the low-level dataframe-based event log, as shown in Algorithm 3. On the one hand, if only traces having exactly the same flow are recorded, the one with the shortest duration is selected. If, on the other hand, every recorded trace is different by the others, they are compared using the Levenshtein distance algorithm [23], which defines the distance of the textual version of two traces (built by concatenating the actions' name associated to the events in the trace) as the minimal number of edit operations necessary to transform a (textual) trace into the other. The most similar

traces (a threshold percentage of similarity can be customized depending on the routine's context) are grouped into a single set, and the shortest trace (from the duration perspective) in that set is selected as the representative routine variant to be later enacted by a SW robot. If there are not similar traces in the log, the one with the shortest duration is selected among all the available ones.

The working of the Process Discovery component is shown in Algorithm 3.

Algorithm 3 Finding the most frequent routine variant

```
procedure selectMostFrequentVariant(dataframe, threshold)
    df ← flatten dataframe
    df1 ← group rows with same caseID into single row
    df1 ← calculate duration for each trace
    df2 ← compute variants
    if ∃ predominant variant with equal traces then
        min_duration_trace ← select trace in that variant with shortest duration
    else if ∃ similar traces (by a certain threshold) then
        df3 ← group similar traces into a single variant
        min_duration_trace ← select trace in that variant with shortest duration
    else                                              ▷ All traces are different
        min_duration_trace ← select trace with shortest duration among all variants
    end if
    return min_duration_trace
end procedure
```

Once the routine to automatize is selected, before its enactment with a SW robot, it is possible for a RPA analyst to personalize the values stored in its events through a custom dialog window (cf. Figure 4). The tool automatically detects the events that can be edited, such as typing something in a web page, renaming a file, pasting a text or editing an Excel cell, and dynamically builds the GUI to let the RPA analyst editing them. After confirmation, the dataframe-based event log is updated.

Finally, the Python executable scripts based on the most frequent RPA routine (updated with the RPA analyst's edits) is generated by scanning the recorded low-level events in the dataframe-based log and converting them into executable pieces of SW code in Python. To properly work, the script generation algorithm (here omitted for the sake of space) relies on *Automagica*,[7] an Open Source framework for process automation, and *Selenium*,[8] a popular suite of tools for automating web browsers. Note also that the script generation algorithm takes into account only the platform where the SW robot is going to be run, regardless of the operating system used to capture the log. For example, if the (selected)

[7] https://github.com/automagica/automagica
[8] https://www.selenium.dev/

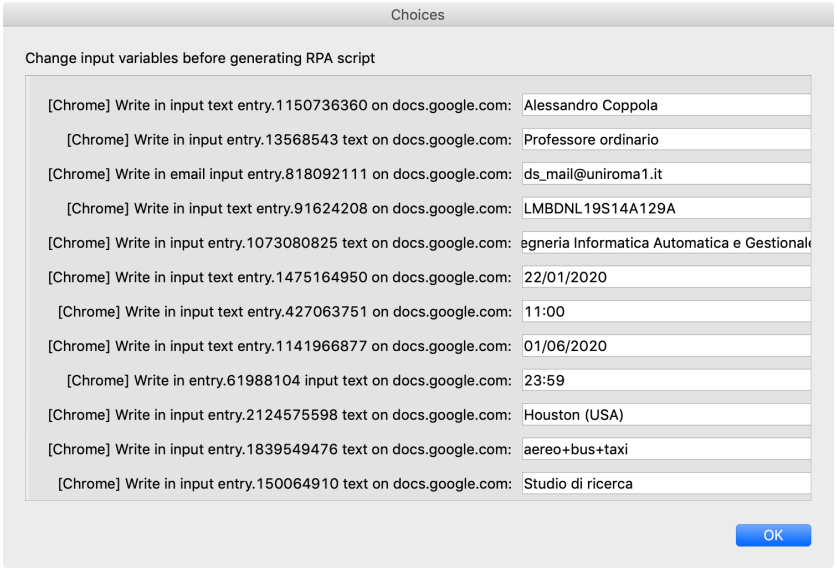


Fig. 4: Custom dialog window to personalize editable fields of a routine variant

most frequent routine variant was recorded on a Windows operating system, but the tool is being executed on macOS, the RPA scripts will be generated taking into account this aspect, e.g., by converting the information about the system paths. This guarantees cross-platform compatibility across event logs recorded on different platforms.

4 SmartRPA in action

SmartRPA was tested with the running example presented in Section 2. We provided the tool to 25 different end users that were instructed to fill the Google Form using the data from the Excel spreadsheet containing the information to apply for a travel request. We selected this routine because, for recording the UI actions to emulate, it is required to exploit all the logging modules provided by the Action Logger. Specifically, *(i)* actions to copy and paste data from the spreadsheet to the web form (System Logger), *(ii)* web navigation actions to access to Google Form (Browser Logger), and *(iii)* actions for moving between the cells of the spreadsheet to access the single values of the travel request (Office Logger). The exact steps to correctly perform the routine and record the UI actions involved are the following ones:

1. Open the Action Logger, tick the checkboxes related to Clipboard, Excel and the browser installed on the applicant's PC/MAC, and click *"Start logger"*.
2. Open the Excel spreadsheet that contains the data of a travel request.
3. Open Google Form.

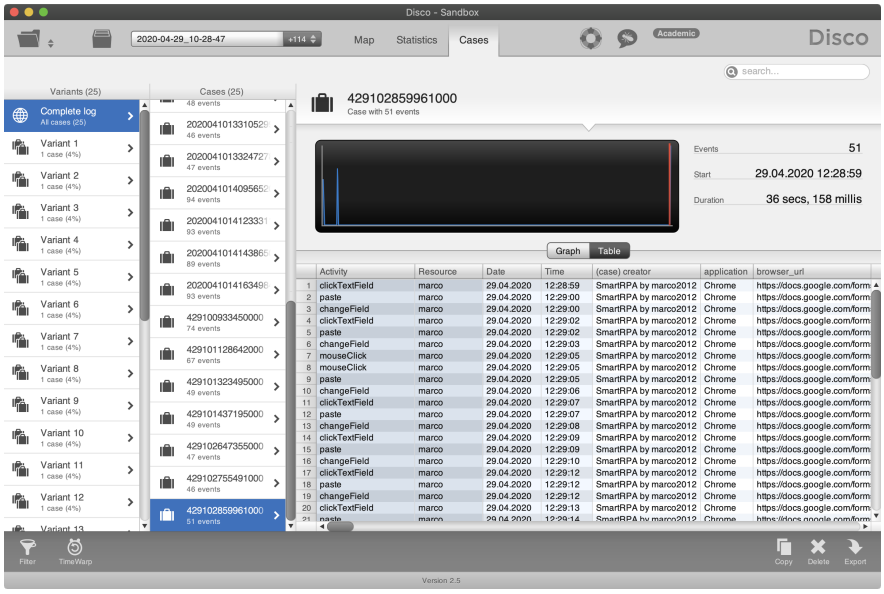


Fig. 5: An overview of the low-level event log opened in Fluxicon Disco

4. Copy and paste each value from the Excel spreadsheet to its respective field on the web form.
5. Submit the form. Once done, a confirmation email is sent to the applicant.

All the UI actions were recorded on 25 different computer systems having different features and operating systems, and stored in 25 event logs in CSV format. Then, we merged the CSV event logs into a single dataframe-based event log (and a corresponding XES file) using the Log Processing tool. An overview of the final event log has been analyzed through Disco, as shown in Figure 5. In our test, we found 25 slightly different execution traces, resulting in 25 potential variants to properly complete the routine.

At this point, according to the working of SmartRPA, the Process Discovery component executed Algorithm 3, grouping together 7 traces out of the 25 available because they were similar by at least 90%. It is worth to notice that this particular threshold was set by us a-priori, and it is customizable depending on the specific routine's application context. Finally, among the 7 variants selected, the one having the shortest duration was chosen by the tool (specifically, the one with case ID `429102859961000` in Figure 5). Figure 4 shows the custom dialog window to personalize the editable values of the most frequent routine variant of the running example. Taking into account the last edits made, SmartRPA can finally generate the required executable scripts to run the SW robot that emulates the routine execution on the UI. A screencast with installation instructions and showing the working of SmartRPA against the running example is available in the github repository of the tool at: https://github.com/bpm-diag/smartRPA/.

5 Related work

The state-of-the-art in RPA is plenty of recent works that are focused on optimizing specific BPM aspects of a RPA project. In the literature, there exist three main groups of approaches that are targeted to automatically derive the behaviour of SW robots.
The first group of approaches aims at learning how to automate routines by observing human users that perform routine tasks in their computer systems. SmartRPA falls in this category. Specifically:

- The works [21,10] present a method to record UI actions performed within Excel and Google Chrome into an event log, and enable the use of process mining techniques to detect which fragments of a routine can be automated. Conversely, SmartRPA records only those UI actions that is known at the outset that can be automated, and consequently the associated routines. In addition, SmartRPA enables to record a much larger spectrum of UI actions, not just limited to Excel and Google Chrome (cf. Section 3).
- The work [18] proposes a method to improve the early stages of the RPA lifecycle by reducing the effort to analyze the actual system using process mining techniques based on a-priori models. SmartRPA focuses on automating the best (in terms of frequency and time duration) recorded routine variant without requiring any a-priori model.
- In [24], the authors present the *Desktop Activity Mining* tool, which records the desktop-based UI actions of users performing an office-based routine task, and employs process mining techniques to discover an integrated process model describing the behaviour of such routine. However, Desktop Activity Mining does not use events to keep track of UI actions, but it is based on recording the mouse click coordinates on the screen, and thus it can not replicate the same user's observed behavior performed in different computer systems. On the contrary, SmartRPA records the events happened during a UI interaction, so it can work across different computer systems. In addition, the identification of similar routine variants is not done using the screenshots of the user's desktop (like happens in [24]) that may differ between different computer systems, but it is performed in a way that guarantees cross-platform compatibility of the recorded event logs.
- In [11], the authors propose a self-learning approach to automatically detect high-level RPA-rules from captured historical low-level behaviour logs. An if-then-else deduction logic is used to infer rules from behaviour logs by learning relations between the different routines performed in the past. Then, such rules are employed to facilitate the SW robots instantiation. A similar approach is adopted in [20], where the *FlashExtract* framework is presented. FlashExtract allows to extract relevant data from semi-structured documents using input-output examples, from which one can derive the relations underlying the working of a routine. SmartRPA adopts a different approach: multiple variants of a routine execution are considered and the most frequent one is chosen for being executed by a SW robot, with the possibility of customizing some of its input values.

– The work [26] identifies repetitive edits to text documents by keeping track of a graph of edits and suggests automation rules for SW robots. While this work focuses on supporting expert users in the manual development of SW code, SmartRPA is targeted to automatically generate executable scripts for SW robots.

It is worth to quickly discuss the other two groups of approaches towards SW robots automation, even if they focus on different challenges than SmartRPA. The *second* group of approaches focus on learning the anatomy of routine tasks from natural language descriptions of the procedures underlying such routines. In this direction, the work [16] defines a new grammar for complex workflows with chaining machine-executable meaning representations for semantic parsing. In [22], the authors provide an approach to learn activities from text documents employing supervised machine learning techniques such as feature extraction and support vector machine training. Similarly, in [13] the authors adopt a deep learning approach based on Long Short-Term Memory (LSTM) recurrent neural networks to learn the relationship between activities of a routine task.
Finally, a *third* group of approaches exist that aim to eliminate human-dependent training [7,17]. They rely on probabilistic and machine learning algorithms to automatically train SW robots, so that any manual effort is avoided. These approaches are currently the least mature if compared with the others discussed above, but potentially with the best promises for realizing fully automated intelligent RPA approaches.

6 Discussion and Concluding Remarks

While RPA is currently used for automating routines and high-volume tasks requiring a manual intervention of expert users, the aim of SmartRPA is to automatically develop SW robots directly from the user's observed behavior. SmartRPA offers an innovative contribution to RPA technology with the goal of mitigating some of its core downsides. Notably, using SmartRPA, all the routine executions recorded by the tool can be automated, an high-level flowchart diagram is presented to expert users for potential diagnosis operations, and the executable RPA scripts to drive the working of a SW robot are generated based on the most frequent routine variant. In addition, the tool is cross-platform and allows to personalize some input fields of the selected routine variant before executing the related RPA scripts, thus supporting those steps that require manual user inputs. As a consequence, this makes the working of SW robots more flexible and adaptable to several real-world situations.
Thanks to its Action Logger, SmartRPA aims also at improving the *auditability* of RPA tools, since all routine tasks executed by human users on a UI are previously recorded in dedicated event logs, making them auditable to external users. It is worth to notice that the logs produced by the state-of-the-art RPA tools have usually a poor quality (actions may be missing or not recorded properly), since they are mainly used for debugging purposes [4]. Conversely, SmartRPA

aims at logs at the highest possible quality level thanks to its detailed recording phase performed during the training sessions.
Of course, the tool presents some weaknesses that we are tackling as future works. First of all, the executable RPA scripts for implementing SW robots are developed based on the most frequent routine variant recorded in a dataframe-based event log. However, a more accurate approach to derive the SW robot's behaviour would consist of interpreting at run-time the flowchart discovered from many routine executions stored in the event log, and selecting step-by-step the most suitable flowchart fragment (i.e., the sub-routine) to be executed by the SW robots. A second weakness, which strongly depends by the first one, relies on the fact that SmartRPA is currently able to emulate routines where the procedure to be automated is the same for all applicants, i.e., the only difference is in the values entered by the users performing the training session into fixed pre-defined fields. This limitation can be observed also in the running example, where the fields to be filled in the Excel sheet are static (they are always the same ones), and only their content can vary from applicant to applicant.
Despite the weaknesses, we consider this work as an important first step towards a more complete approach and tool towards the fully automated generation of executable RPA scripts.

Acknowledgments. This work has been supported by the "Dipartimento di Eccellenza" grant, the H2020 projects DESTINI and FIRST, the Italian project RoMA - Resilience of Metropolitan Areas, and the Sapienza grant BPbots.

References

1. van der Aalst, W.M.P., Bichler, M., Heinzl, A.: Robotic Process Automation. Bus. Inf. Syst. Eng. **60**(4), 269–272 (2018)
2. Agostinelli, S., Maggi, F.M., Marrella, A., Milani, F.: A User Evaluation of Process Discovery Algorithms in a Software Engineering Company. In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC). pp. 142–150 (2019). https://doi.org/10.1109/EDOC.2019.00026
3. Agostinelli, S., Marrella, A., Mecella, M.: Research Challenges for Intelligent Robotic Process Automation. In: Business Process Management (BPM 2019) Int. Workshops. pp. 12–18 (2019). https://doi.org/10.1007/978-3-030-37453-2_2
4. Agostinelli, S., Marrella, A., Mecella, M.: Towards Intelligent Robotic Process Automation for BPMers (2020), http://arxiv.org/abs/2001.00804
5. Aguirre, S., Rodriguez, A.: Automation of a Business Process Using Robotic Process Automation (RPA): A Case Study. In: Applied Computer Sciences in Engineering. pp. 65–71. Springer (2017)
6. AI-Multiple: All 52 RPA Software Tools & Vendors of 2020: Sortable List (2019), https://blog.aimultiple.com/rpa-tools/
7. Ayub, A., Wagner, A.R.: Teach Me What You Want to Play: Learning Variants of Connect Four through Human-Robot Interaction (2020), https://arxiv.org/abs/2001.01004
8. Berti, A., van Zelst, S.J., van der Aalst, W.: Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science (2019), http://arxiv.org/abs/1905.06169

9. Bisbal, J., Lawless, D., Wu, B., Grimson, J.: Legacy Information Systems: Issues and Directions. IEEE Software **16**(5), 103–111 (1999)
10. Bosco, A., Augusto, A., Dumas, M., Rosa, M.L., Fortino, G.: Discovering Automatable Routines from User Interaction Logs. In: Business Process Management Forum - BPM Forum 2019. pp. 144–162 (2019)
11. Gao, J., van Zelst, S.J., Lu, X., van der Aalst, W.M.P.: Automated Robotic Process Automation: A Self-Learning Approach. In: On the Move to Meaningful Internet Systems: OTM 2019 Conferences. pp. 95–112. Springer (2019)
12. Geyer-Klingeberg, J., Nakladal, J., Baldauf, F., Veit, F.: Process Mining and Robotic Process Automation: A Perfect Match. In: 16th Int. Conf. on Business Process Management (BPM'18), Dissertation/Demos/Industry track (2018)
13. Han, X., Hu, L., Dang, Y., Agarwal, S., Mei, L., Li, S., Zhou, X.: Automatic Business Process Structure Discovery using Ordered Neurons LSTM: A Preliminary Study (2020), https://arxiv.org/abs/2001.01243
14. Hill, J., Ford, W.R., Farreras, I.G.: Real conversations with artificial intelligence: A comparison between human-human online conversations and human-chatbot conversations. Comput. Hum. Behav. **49**, 245–250 (2015)
15. IEEE Digital Library: Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. IEEE Std 1849-2016 (2016). https://doi.org/10.1109/IEEESTD.2016.7740858
16. Ito, N., Suzuki, Y., Aizawa, A.: From natural language instructions to complex processes: Issues in chaining trigger action rules (2020), https://arxiv.org/abs/2001.02462
17. Jenkins, P., Wei, H., Jenkins, J.S., Li, Z.: A Probabilistic Simulator of Spatial Demand for Product Allocation (2020), https://arxiv.org/abs/2001.03210
18. Jimenez-Ramirez, A., Reijers, H.A., Barba, I., Del Valle, C.: A Method to Improve the Early Stages of the Robotic Process Automation Lifecycle. In: 31st Int. Conf. on Advanced Information Systems Engineering (CAiSE'19). pp. 446–461 (2019)
19. Kirchmer, M.: Robotic Process Automation-Pragmatic Solution or Dangerous Illusion. BTOES Insights, June'17 (2017)
20. Le, V., Gulwani, S.: FlashExtract: a framework for data extraction by examples. In: ACM SIGPLAN PLDI '14. pp. 542–553 (2014)
21. Leno, V., Polyvyanyy, A., Rosa, M.L., Dumas, M., Maggi, F.M.: Action logger: Enabling process mining for robotic process automation. In: Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at 17th Int. Conf. on Business Process Management, (BPM'19). pp. 124–128 (2019)
22. Leopold, H., van der Aa, H., Reijers, H.A.: Identifying Candidate Tasks for Robotic Process Automation in Textual Process Descriptions. In: Enterprise, business-process and information systems modeling, pp. 67–81. Springer (2018)
23. Levenshtein, V.: Efficient implementation of the levenshtein-algorithm, fault-tolerant search technology, error-tolerant search technologies (2007), http://www.levenshtein.net/
24. Linn, C., Zimmermann, P., Werth, D.: Desktop activity mining - A new level of detail in mining business processes. In: Workshops der INFORMATIK 2018 - Architekturen, Prozesse, Sicherheit und Nachhaltigkeit, 26.-27. pp. 245–258 (2018)
25. Marrella, A., Mecella, M., Sardiña, S.: Supporting adaptiveness of cyber-physical processes through action-based formalisms. AI Commun. **31**(1), 47–74 (2018). https://doi.org/10.3233/AIC-170748
26. Miltner, A., Gulwani, S., Le, V., Leung, A., Radhakrishna, A., Soares, G., Tiwari, A., Udupa, A.: On the fly synthesis of edit suggestions. In: ACM Program. Lang. **3**(OOPSLA), 143:1–143:29 (2019)