



# SmartRPA: A Tool to Reactively Synthesize Software Robots from User Interface Logs

Simone Agostinelli<sup>(✉)</sup>, Marco Lupia, Andrea Marrella, and Massimo Mecella

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

**Abstract.** Robotic Process Automation (RPA) is an emerging technology that automates intensive routine tasks (or simply *routines*) previously performed by a human user on the User Interface (UI) of a computer system, by means of a software (SW) robot. To date, RPA tools available in the market strongly relies on the ability of human experts to manually implement the routines to automate. Being the current practice time-consuming and error-prone, in this paper we present SmartRPA, a cross-platform software tool that tackles such issues by exploiting UI logs keeping track of many routine executions to generate executable RPA scripts that automate the routines enactment by SW robots.

## 1 Introduction

Robotic Process Automation (RPA) is an automation technology that operates on the user interface (UI) of software applications and replicates, by means of a software (SW) robot, mouse and keyboard interactions to remove high-volume routine tasks (a.k.a. *routines*) [3]. To take full advantage of this technology in the early stages of the RPA life-cycle, organizations leverage the support of skilled human experts to:

1. identify the candidate routines to automate by means of interviews and observation of workers conducting their daily work;
2. record the interactions that take place during routines' enactment on the UI of SW applications into dedicated *UI logs*, which are mainly used for debugging purposes only;
3. manually specify their conceptual and technical structure (often in form of flowchart diagrams), which will drive the development of dedicated *RPA scripts* reflecting the behavior of SW robots.

While this approach has proven to be effective to execute rule-based and well-structured routines [5], it becomes time-consuming and error-prone in presence of routines that are less deterministic and require decisions [7].

In this paper, we tackle the above issue by presenting SmartRPA, an open-source software tool that is able to reason over the UI logs keeping track of many

routine executions (cf. step 2), and to automatically synthesize SW robots that emulate the most suitable routine variant for any specific intermediate user input that is required during the routine execution, thus skipping completely the manual modeling activity of the flowchart diagrams (cf. step 3). SmartRPA implements the approach presented in [2] and is available for download at <https://github.com/bpm-diag/smartRPA/>.

The rest of the paper is organized as follows. Section 2 introduces a running example. Section 3 presents the tool architecture and the technical aspects of SmartRPA. Section 4 discusses some experiments performed to evaluate the robustness and feasibility of the tool. Finally, Sect. 5 concludes the paper.

## 2 Running Example

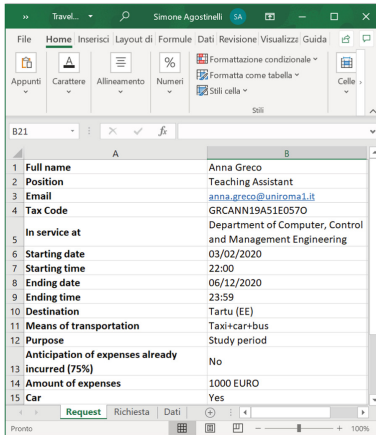
Below, we introduce a real-life scenario used to explain the functioning of our tool. The example is inspired by the work performed by the Administration Office of the Department of Computer, Control and Management Engineering (DIAG) of Sapienza Università di Roma, which consists of filling the travel authorization request form made by the personnel of DIAG for travel requiring prior approval. We specifically consider the task of filling a well-structured Excel spreadsheet (cf. Fig. 1(a)), manually performed by a request applicant that provides some personal information together with further information related to the travel. Then, the spreadsheet is sent via email to an employee of the Administration Office of DIAG, which is in charge of processing the request: 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. Fig. 1(b)). In addition, if the request applicant declares the need to use a personal car as one of the means of transport for the travel (by filling the dedicated row labeled with “Car” in the spreadsheet), then the employee has to activate the request on the Google form (in this case, a dialog box labeled “Own car request” appears on the UI, cf. Fig. 1(b)) and then accept or reject the personal car request. When the data transfer for a given travel authorization request has been completed, the employee presses the “Submit” button to confirm data and submit them into an internal database. Finally, a confirmation email is sent automatically to the applicant when data are submitted.

The above routine procedure (in the following, we will denote it as R) is usually performed manually, it is tedious (as it must be repeated for any new travel request) and prone to errors. A proper execution of R requires a path on the UI made by the following user actions:<sup>1</sup>

- `loginMail`, to access the client email;
- `accessMail`, to access the specific email with the travel request;
- `downloadAttachment`, to download the Excel file including the travel request;
- `openWorkbook`, to open the Excel spreadsheet;

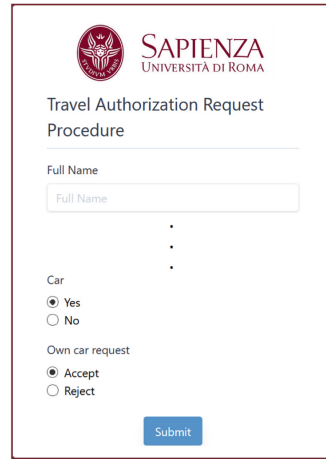
---

<sup>1</sup> Note that the user actions recorded in a UI log can have a finer granularity than the high-level ones used here just with the purpose of describing the routine’s behaviour.



	A	B
1	Full name	Anna Greco
2	Position	Teaching Assistant
3	Email	anna.greco@uniroma1.it
4	Tax Code	GRCANN19A51E0570
5	In service at	Department of Computer, Control and Management Engineering
6	Starting date	03/02/2020
7	Starting time	22:00
8	Ending date	06/12/2020
9	Ending time	23:59
10	Destination	Tartu (EE)
11	Means of transportation	Taxi+car+bus
12	Purpose	Study period
13	Anticipation of expenses already incurred (P5%)	No
14	Amount of expenses	1000 EURO
15	Car	Yes

(a) Excel spreadsheet



**SAPIENZA**  
UNIVERSITÀ DI ROMA

### Travel Authorization Request Procedure

Full Name

Car

Yes  
 No

Own car request

Accept  
 Reject

(b) Google form

**Fig. 1.** UIs involved in the running example

- `openGoogleForm`, to access the Google Form to be filled;
- `getExcelCell`, to select the cell in the  $i$ -th row of the Excel spreadsheet;
- `copy`, to copy the content of the selected cell;
- `clickGoogleFormTextField`, to select the specific text field of the Google form;
- `paste`, to paste the content of the cell into a text field of the Google form;
- `activateCarRequest`, to activate in the Google form the dialog box for approving or rejecting the car request;
- `accept`, to press the button on the Google form that approves the request;
- `reject`, to press the button on the Google form that rejects the request;
- `formSubmit`, to finally submit the Google form to the internal database.

The user actions `openWorkbook` and `openGoogleForm` can be performed in any order. Moreover, the sequence of actions `(getCell, copy, clickTextField, paste)` can be repeated for any travel information to be moved from the Excel spreadsheet to the Google form. Finally, in case of a car request to be evaluated (action `activateCarRequest`), the execution of `accept` or `reject` is exclusive.

### 3 SmartRPA Architecture

In this section, we give a detailed description of the architecture of SmartRPA (see Fig. 2) that consists in five main SW components implemented in Python.

The first SW component of the architecture is an **Action Logger** able to record different types of UI actions from multiple SW applications during the enactment of the routine under study. Specifically, a training session in which several users perform the routine to be automated is required to record the UI actions involved in its execution. The Action Logger provides a Graphical User

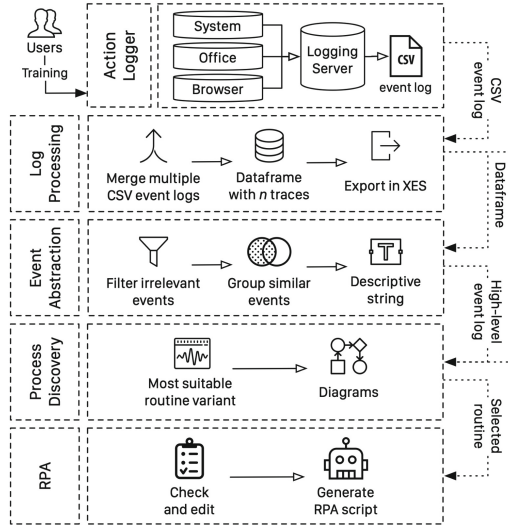


Fig. 2. SmartRPA architecture

Interface (GUI) that allows a user to select which SW applications s/he wants to record UI actions on (cf. Fig. 3). The Action Logger provides three different types of logging modules: (i) a *System Logger* able to detect those UI actions not related to specific SW applications, (ii) an *Office Logger* able to detect the UI actions performed within Microsoft Office applications, and (iii) a *Browser Logger* able to detect the UI actions on web browsers.

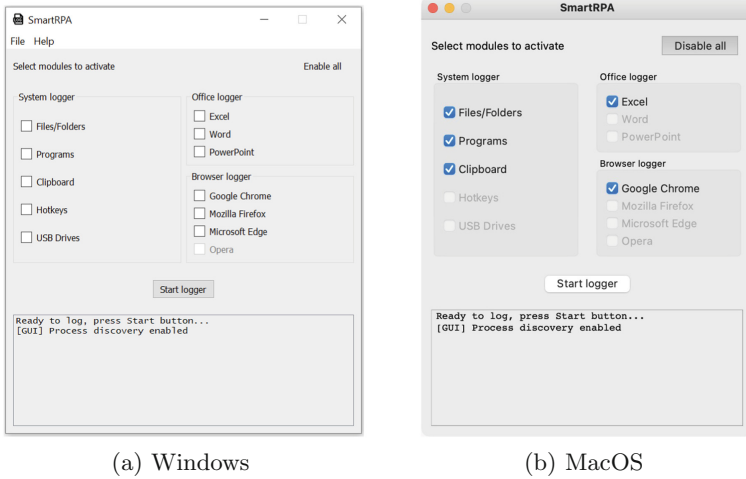
The UI actions recorded by the logging modules are sent to a Logging Server, implemented with the *Flask* framework,<sup>2</sup> in charge to store and organize them as *events* into several CSV event logs, i.e., the UI logs.

The exact steps to correctly perform R (cf. Sect. 2) are the following ones:

1. Open the Action Logger, tick the checkboxes related to Excel, Clipboard and the browser installed on the applicant's PC/MAC, and click "Start logger".
2. Open the Excel spreadsheet containing the information about the travel.
3. Open the Google form.
4. Copy and paste each value from the Excel spreadsheet to the Google form.
5. Accept or reject the personal car request (if required).
6. Submit the form. Once done, a confirmation email is sent to the applicant.
7. Push the "Stop logger" button to stop the Action Logger.

It is worth noticing that multiple users can run the Action Logger on their computer system many times performing R in different training sessions. Each CSV event log contains exactly one long trace of UI actions performed in a single training session by a single user. Technically speaking, (i) system events are

<sup>2</sup> <https://palletsprojects.com/p/flask>.



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

captured using *PythonCOM* (for Windows APIs and COM objects) and *MacFSEvents* (for MacOS); (ii) events generated by Microsoft Office applications are captured using the Office JavaScript APIs; and (iii) browser events are captured using JavaScript web extensions developed for each supported web browser.

The second SW component of the architecture is the **Log Processing** tool that is triggered when any training session is considered as completed. Specifically, after  $n$  training sessions, the Logging Server will deliver the  $n$  created CSV event logs to the Log Processing component, in charge of import them into a single Pandas dataframe.<sup>3</sup> A dataframe is a two-dimensional size-mutable and heterogeneous tabular data structure with labeled axes, which is used as the main artifact to represent event logs in SmartRPA. The dataframe created by the Log Processing component consists of low-level events with fine granularity associated one-by-one to a recorded UI action, including several columns representing the payload of the recorded event, i.e.: the timestamp, the application that generated the event, the resources involved, etc. SmartRPA is also able to produce a XES<sup>4</sup> (eXtensible Event Stream) version of the datastream that will contain exactly  $n$  traces, one for each recorded CSV event log and can be inspected using the most popular process mining tools, such as *ProM*,<sup>5</sup> or *Disco*<sup>6</sup>.

The third SW component is an **Event Abstraction** engine used to produce a high-level event log from the low-level one with the goal to: (i) filter out noise and irrelevant events for the routine execution. For example, during several

<sup>3</sup> <https://pandas.pydata.org/>.

<sup>4</sup> XES is the standard for the storage, interchange, and analysis of event logs.

<sup>5</sup> <http://www.promtools.org/>.

<sup>6</sup> <https://fluxicon.com/disco/>.

training sessions of R, applications related to the operating system may start in background while the Action Logger is being recording the UI log, and they may dirty the recording phase of the users during their training session. From a workflow perspective, these events are not relevant for any RPA analyst that aims to understand the general behaviour of the routine and thus they can be filtered out; *(ii)* group similar low-level events to the same high-level concept. For example, in a web page, the Action Logger can capture different types of clicks, based on the element clicked. From the RPA analyst perspective it is not relevant what kind of click was performed, thus the high-level workflow of the routine may just show the action “Click on button”; *(iii)* create descriptive labels. Any recorded event provides a low-level description of the UI action performed. To make the UI action underlying an event more descriptive for the RPA analyst, the payload information stored in the low-level 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 B2 on Sheet ‘Request’ with value ‘Full Professor’*”.

At this point, the **Process Discovery** component exploits the high-level event log to derive the underlying high-level workflow as a Directly-Follows Graph (DFG), by applying the heuristic miner (the decision to employ the heuristic miner has been driven by its ability to discover highly understandable flowcharts from a BPM analyst perspective [1]) implemented in PM4PY [4]. In addition, the knowledge of the workflow underlying the routine, coupled with the low-level version of the dataframe-based event log, will be used to support the identification of different *variation points*, thus leading to the detection of the most suitable routine variant according to intermediate user inputs observed in the low-level dataframe-based event log. A variation point is a point in the routine execution where a user choice needs to be made between multiple possible variants. For example, the routine under analysis in Sect. 2 consists of one variation point that contains three different user inputs that can led to three different routine variants of R: *(i)* the user performs the UI action `activateCarRequest` by clicking ‘No’ on the Google form, *(ii)* the user first performs the UI action `activateCarRequest` and then the UI action `accept`, *(iii)* the user first performs the UI action `activateCarRequest` and then the UI action `reject`.

Once the routine variant to automatize is selected, before its enactment with a SW robot, it is possible for an RPA analyst to personalize the values stored in its events, thanks to the **Script Generation** component. SmartRPA automatically detects the events that can be edited, such as pasting a text or editing an Excel cell, and let the RPA analyst editing them. After confirmation, the low-level dataframe-based event log is updated. Finally, the Python executable script based on the selected routine variant and 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. The script generation component relies on *Automagica*<sup>7</sup> and *Selenium*,<sup>8</sup> a popu-

<sup>7</sup> <https://github.com/automagica/automagica>.

<sup>8</sup> <https://www.selenium.dev/>.

**Table 1.** Experimental results for the *synthetic* case study for logs with 1000 routine executions. The time (in *milliseconds*) is the average per trace.

Event size: 40	Time				
Trace size	1	2	3	4	5
25	0.453	0.452	0.53	0.409	0.423
50	0.417	0.433	0.417	0.425	0.419
75	0.439	0.511	0.424	0.43	0.431
100	0.454	0.416	0.421	0.424	0.431
Event size: 80	Time				
Trace size	1	2	3	4	5
25	0.422	0.428	0.43	0.413	0.412
50	0.427	0.425	0.444	0.417	0.428
75	0.42	0.428	0.553	0.422	0.437
100	0.442	0.434	0.428	0.438	0.432
Event size: 120	Time				
Trace size	1	2	3	4	5
25	0.413	0.507	0.421	0.416	0.421
50	0.421	0.412	0.417	0.42	0.421
75	0.425	0.433	0.438	0.451	0.429
100	0.437	0.433	0.428	0.532	0.523

lar suite of tools for process and web browsers automation. Note that the Script Generation component considers only the platform where the SW robot is going to be run regardless of the operating system used to record the log, thus achieving cross-platform compatibility. SmartRPA is also able to generate RPA scripts compatible with the commercial tool *UiPath Studio*.<sup>9</sup>

## 4 Evaluation

SmartRPA has been tested using synthetic experiments employing UI logs of increasing complexity. We generated 240 different UI logs (containing in total 150.000 different routine executions), in a way that each UI log was characterized through a unique configuration obtained by varying the following input settings:

- *log\_size*: number of routine executions in the UI log (250/500/750/1000);
- *trace\_size*: number of events in each routine execution (25/50/75/100);
- *events\_size*: number of possible different events to be considered for the creation of a trace (40/80/120);
- *variation\_points*: number of variation points in the UI log (1/2/3/4/5).

<sup>9</sup> <https://www.uipath.com/product/studio>.

The amount of possible decisions to be taken in a variation point was generated randomly, ranging from 2 to 10 possible outgoing decisions. The synthetic UI logs generated for the test are available at: <https://github.com/bpm-diag/smartRPA/>. The target was to investigate if the amount and anatomy of variation points discovered by SmartRPA is the same that was syntetically introduced in the sample routine executions recorded in the UI logs (i.e., *robustness*), and to measure the performance of the tool to generate a SW robot by solely using the UI logs (i.e., *feasibility*). Concerning the robustness of the tool, for all the 240 tested logs the tool was able to always discover the correct variation points to be considered for the synthesis of SW robots. Concerning the feasibility, it was measured in terms of the computation time required to generate a SW robot starting from UI logs of growing complexity. The results, which are summarized in Table 1,<sup>10</sup> indicate that the tool scales well in case of an increasing number of variation points and routine executions/alphabet of events of growing size.

## 5 Concluding Remarks

SmartRPA offers an innovative contribution to RPA technology with the goal of mitigating some of its core downsides related to the implementation of SW robots made by expert users. Close to SmartRPA there is Robidium [6], a tool that generates RPA scripts based on the *most frequent* routine variant observed in the UI log. Conversely, SmartRPA enables to generate the *best observed* routine variant, employing the input conditions available before the routine enactment. The main weakness of SmartRPA is correlated with the quality of information recorded in real-world UI logs. Since a UI log is fine-grained, routines executed with many different strategies may potentially affect the robustness of our tool to the detection of variation points. For this reason, as a future work, we are going to perform a robust evaluation of the tool on real-world case studies including heterogeneous UI logs obtained from different application domains.

**Acknowledgments.** This work has been supported by the “Dipartimento di Eccellenza” grant, the H2020 project DataCloud and the Sapienza grant BPbots.

## References

1. Agostinelli, S., Maggi, F.M., Marrella, A., Milani, F.: A user evaluation of process discovery algorithms in a software engineering company. In: EDOC (2019)
2. Agostinelli, S., Lupia, M., Marrella, A., Mecella, M.: Automated generation of executable RPA scripts from user interface logs. In: Asatiani, A., et al. (eds.) BPM 2020. LNBP, vol. 393, pp. 116–131. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-58779-6\\_8](https://doi.org/10.1007/978-3-030-58779-6_8)

<sup>10</sup> For the sake of space, the table includes only the results related to UI logs containing 1000 routine executions.



3. Agostinelli, S., Marrella, A., Mecella, M.: Research challenges for intelligent robotic process automation. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 12–18. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-37453-2\\_2](https://doi.org/10.1007/978-3-030-37453-2_2)
4. Berti, A., van Zelst, S.J., van der Aalst, W.: Process Mining for Python (PM4Py): Bridging the Gap Between Process and Data Science (2019)
5. 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: Giorgini, P., Weber, B. (eds.) CAiSE 2019. LNCS, vol. 11483, pp. 446–461. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-21290-2\\_28](https://doi.org/10.1007/978-3-030-21290-2_28)
6. Leno, V., Deviatykh, S., Polyvyanyy, A., La Rosa, M., Dumas, M., Maggi, F.M.: Robidium: automated synthesis of robotic process automation scripts from UI logs. In: BPM Demonstration Track (2020)
7. Marrella, A., Mecella, M., Sardiña, S.: Supporting adaptiveness of cyber-physical processes through action-based formalisms. *AI Commun.* **31**(1), 47–74 (2018)