Discovering Two-Level Business Process Models from User Interface Event Logs

Irene Barba[1(✉)], Carmelo Del Valle[1], Andrés Jiménez-Ramírez[1], Barbara Weber[2], and Manfred Reichert[3]

[1] Departamento de Lenguajes y Sistemas Informáticos, University of Seville, Seville, Spain
{irenebr,carmelo,ajramirez}@us.es

[2] Institut for Computer Science, University of St. Gallen, St. Gallen, Switzerland
barbara.weber@unisg.ch

[3] Institute of Databases and Information Systems, Ulm University, Ulm, Germany
manfred.reichert@uni-ulm.de

Abstract. The widespread adoption of Robotic Process Automation (RPA) to automate repetitive tasks has surged, utilizing front-end interfaces to replicate human actions. Task mining is integral to the RPA lifecycle, capturing and analyzing fine-grained user interactions. As opposed to traditional process mining, which focuses on business processes, task mining relies on low-level events from user interface (UI) logs. Bridging the gap between task mining and process mining, this paper introduces a novel approach to generate two-level process models from UI logs. This method addresses challenges related to granularity and screen ambiguities by considering two levels of abstraction. It identifies each activity at the screen level for a comprehensive overview and delves into fine-grained user actions for a deeper understanding. This dual-level model aids in resolving ambiguities in inferred user intention, offering labeled activities for enhanced understandability. The proposed method was validated using synthetically generated UI event logs for an example with real-world characteristics. The research highlights the importance of contextual information in UI event logs and provides a solution to effectively map low-level UI interactions to higher-level user activities for meaningful analysis and automation.

Keywords: Robotic process automation · Hierarchical business process models · Event abstraction · Process discovery

1 Introduction

Robotic Process Automation (RPA) has received attention for automating repetitive tasks by mimicking human actions through front-end interfaces [9,16,18]. Task mining, crucial in the RPA lifecycle [1], captures fine-grained user interactions using UI event logs. In contrast to traditional process mining focusing on end-to-end business processes (BPs), task mining relies on low-level UI events like mouse clicks and keystrokes, allowing for detailed insights and the identification of automation candidates [12]. However, the abundance of fine-grained event data poses challenges in obtaining a holistic understanding of BPs, which is indispensable for any process automation, requiring manual inspection of models for accurate automation. Abstracting low-level UI interactions to higher-level user activities is crucial for meaningful analysis and automation [2]. Different process stakeholders may prioritize varying levels of detail. Moreover, in processes with complex user interfaces, ambiguity can arise from similar sequences of low-level events leading to different outcomes. Incorporating multiple granularity levels can resolve such ambiguities, providing a nuanced understanding of user intentions and enhancing user-friendliness and adaptability in the analysis [8]. Such an integration would allow bridging the gap between task mining (with its focus on understanding how tasks are performed) and process mining (with its focus on obtaining an end-to-end view of a BPs).

Existing process mining literature (e.g., [24,27,30]) predominantly focuses on BP abstraction, with some works emphasizing event log abstraction through clustering or hierarchical representation (e.g., [10,20,21,28]). Notably, these works overlook UI event logs, which pose an additional challenge due to the importance of event context, specifically the user screen and its information. In this regard, the same screen may be used for different purposes (e.g., the same screen is used for both opening and closing a case). Hence, *ambiguity* regarding user intentions [22] might exist, which may be better inferred, for example, after checking subsequent user actions (e.g., the button pushed). In this context, providing semantics to activities that aid in inferring the intention behind is vital for enhancing model understandability.

This paper introduces a method to generate two-level process models from UI event logs, where each event is associated with a screenshot [16]. Although a screen-level analysis of the log may produce useful models for a general understanding of the process —serving as an analytical level, e.g., to assess the suitability of the process to be automated— it may not reveal further concrete details that might be required at a functional level (e.g., to design and implement the automation effectively). For this, the current approach operates at two abstraction levels. At the higher level, each screen serves as a proxy for an activity, providing a comprehensive overview of the process model and capturing the flow between screens, which is particularly valuable in RPA processes. For the lower level of abstraction, standard process mining is applied individually for each screen, considering all relevant low-level events. This granular analysis addresses ambiguities in inferred user intentions. To improve clarity, providing meaningful names to all activities in the generated models is proposed.

The current work builds on our prior work on discovering flat models from UI event logs [16] by discovering two-level process models instead of flat ones, offering a more detailed view of process behavior. These two-level models overcome activity ambiguity issues seen in [16], where similar screens led to the incorrect identification of distinct activities. This proposal also adds labels to activities, improving model interpretation.

Therefore, the primary research question (PRQ) addressed in this paper is: *Can the proposed approach improve the analysis phase of an RPA project?*

The paper is structured as follows: Sect. 2 provides a succinct overview of related works. Section 3 provides insights into the process of generating an event log along a running example. Section 4 outlines our proposed methodology. Experimental results are presented in Sect. 5. The paper concludes in Sect. 6.

2 Related Work

This section presents related work on (1) event abstraction, (2) routine segmentation, (3) BP model discovery from UI event logs, and (4) hierarchical BP model discovery.

[30] investigates approaches to handle event logs with different granularity levels in the context of process mining. Usually, the granularity level is fixed during log preprocessing using event abstraction techniques [11,24,29], abstracting data based on factors like temporal proximity. Furthermore, there exist proposals for event abstraction that explore hierarchical clustering for events or event classes [13,14,25]. These hierarchical clustering methods rely on various techniques, including event correlation, predictive clustering trees, and spatial proximity of events within the log. In contrast, our method utilizes an image-similarity clustering algorithm, ensuring that events with similar screenshots are assigned to the same screen ID [16].

In the context of RPA, dealing with segmentation in the management of UI event logs has been identified as a significant challenge [4,19]. Addressing this challenge, studies such as [5] have delved into routine segmentation. This involves automatically analyzing which user actions contribute to which routines inside a UI event log and cluster such user actions into well-bounded routine traces. In the proposed approach we consider separating the stream of user actions into different cases (i.e., into different routine traces), ensuring that each event in the log is associated with a unique case identifier necessary for the subsequent process mining tasks. This is performed by considering a predefined sequence of events that delineate the start and the end of the cases [16]. However, unlike [5], we do not analyze the particular routine associated with each case since this is not required for the proposed approach. It is important to note that we are focusing on scenarios where a UI event log documents numerous executions of diverse routines, with certain user actions recurring across these routines.

In the realm of discovering BP models from UI event logs, the approach presented in [16] associates each activity with one screen during process mining, creating process models at the screen level (cf. Sect. 3.3). Such approach is then used as our baseline for addressing research question 1 in Sect. 5. Additionally, [9] proposes a methodology for RPA task selection based on UI event logs and process mining, whereas [26] incorporates large language models for task grouping, activity labeling, and connector recommendation. However, none of these works considers more than one level of hierarchy in the discovery of process models from UI event logs.

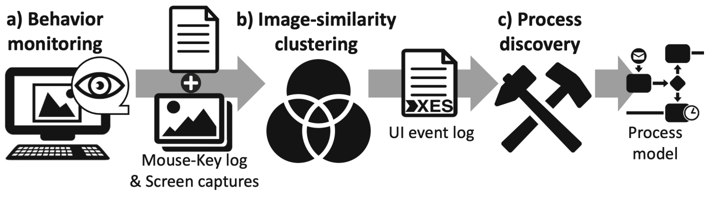


Fig. 1. Overview of the framework for process model discovery (adapted from [16]).

[28] focuses on the discovery of hierarchical BPMN models from logs that contain basic information such as caseID, task name, start time, and end time. [28] detects multi-instance models and builds a hierarchy tree of events and uses heuristic miner for the discovery of the sub-models. Similarly, [10] obtains BPMN models with sub-processes and multi-instances, but the logs need to contain certain data attributes (e.g., primary and foreign keys) to discover event dependencies, not being observable in many logs. In a related way, [21] proposes FlexHMiner, which is based on activity trees to define the components (activities or sub-processes) of the (sub)processes. FlexHMiner achieves better results if domain knowledge is provided, as explained in Sects. 3.3 and 5.4. [21] further allows for randomly grouping components of (sub)processes. Finally, [20] defines Hierarchical Petri Nets (HPNs) and proposes an approach to discover HPNs from event logs with lifecycle information, enabling the detection of nesting activity relations. Unlike these works, the proposed approach starts from UI event logs and considers the particularities of this kind of logs (e.g., image-similarity of screenshots) as valuable information for generating the process model.

3 Event Log Generation for a Running Example

As running example we consider a business context in which a process with three variants needs to be executed to manage client requests (see [7] for the related BPMN model): (1) the client requests unsubscription; after checking that she is up to date with payments, the unsubscription is performed (the client is notified by e-mail); (2) the client requests unsubscription; after checking that she is not up to date with payments, the unsubscription is not performed (the client is notified by e-mail); and (3) the client requests billing information related to her last payment receipt; this receipt is sent to her by e-mail.

Although the process is not real, it simulates a realistic situation based on our experience and previous RPA work. The chosen scenario presents a common challenge in real-world scenarios: the screens have an $n : m$ relationship with business activities, meaning the same screen is used for different goals. This introduces ambiguity in inferred user intentions (cf. Sect. 1). If we model screens as activities [16], the resulting model may depict distinct goals represented by the same activity (screen). To be more precise, within this context, we define a process activity as *ambiguous* if it can be utilized for different purposes, i.e., if the user's intention for it is varied. When screens are used as process activities, the same meaning of *ambiguity* is applied to the screens. For instance, in the example [7], the same screen, "S5", is linked to business activities "Get Client Information" and "Delete Client", leading to potential confusion and compromising the quality of the model. To gain comprehensive insights into process execution, it is crucial to analyze the actions within each screen, focusing on user interactions.

Regarding the running example, we presume that the input UI event log is clean [16], i.e., all log traces represent sequences of events related to the considered process variants. This suggests, for instance, that if the worker checks her email, we can infer that it is necessary for managing the ongoing case.

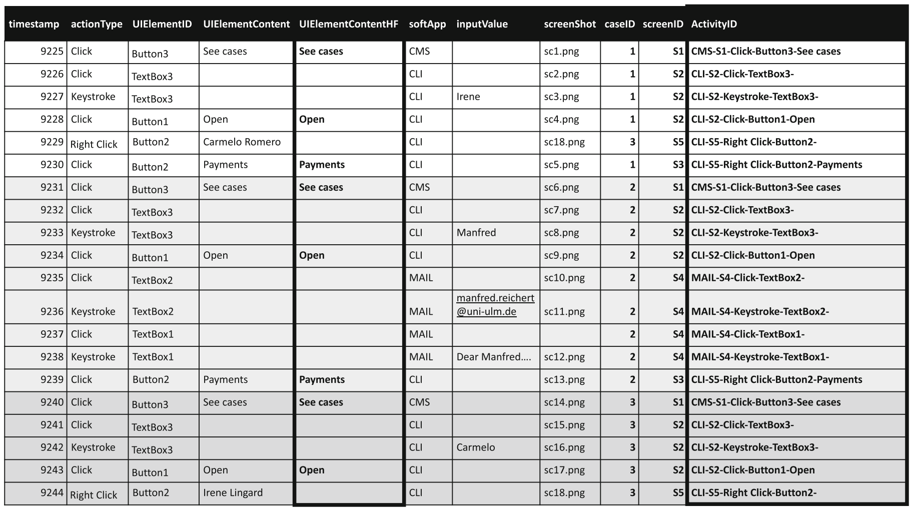
To address the analysis of this process, the framework described in [16] can be used (cf. Fig. 1). Unlike similar frameworks for process mining in RPA [3,12,16] considers the information in the screen valuable. At a glance, it consists of three main phases, i.e., (a) behavior monitoring of the users executing the process (cf. Sect. 3.1), (b) image-similarity clustering of the behavior to infer similar activities and cases (cf. Sect. 3.2), and (c) process model discovery to represent the observed behavior with a BP model (cf. Sect. 3.3).

Initially, we build upon our prior work [16] for steps (a) and (b), and subsequently extend step (c) to handle ambiguities, meaningful labels, and multiple granularity levels when discovering the process model.

3.1 Behavior Monitoring

The BP is supported by an IT system. As suggested by different authors [3,16, 19], recording the user interactions while the user is executing the BP might be more efficient, comprehensive, and less error-prone than conducting traditional interviews with the users. For this, a monitoring tool (cf. Fig. 1a) can be used to record the events produced by the user on the graphical user interface. Several tools are available for such respect [3,19] that produce UI event logs. To be more precise, we consider UI event logs containing the following attributes per event:

- *timestamp*: time at which the action took place.
- *actionType*: action performed by the user. For this attribute, we consider values Click, Keystroke, and RightClick. This categorization aligns with common distinctions in literature, where actions are frequently classified into mouse and keyboard inputs [2].
- *UIElementID*: id related to the UI element the action is performed on, e.g., Button1.
- *UIElementContent*: data contained within the element of the UI on which the action is performed, e.g., 'Send Form'.
- *softApp*: application used for the action, e.g., 'Excel sheet 1'.
- *inputValue*: the data or information provided by a user as input to a system or application, e.g., text the user writes into a text field.
- *screenShot*: image of the screen (and its content) at the moment the event took place (related to the UI Hierarchy attribute [2]).



timestamp	actionType	UIElementID	UIElementContent	UIElementContentHF	softApp	inputValue	screenShot	caseID	screenID	ActivityID
9225	Click	Button3	See cases	**See cases**	CMS		sc1.png	**1**	**S1**	**CMS-S1-Click-Button3-See cases**
9226	Click	TextBox3			CLI		sc2.png	**1**	**S2**	**CLI-S2-Click-TextBox3-**
9227	Keystroke	TextBox3			CLI	Irene	sc3.png	**1**	**S2**	**CLI-S2-Keystroke-TextBox3-**
9228	Click	Button1	Open	**Open**	CLI		sc4.png	**1**	**S2**	**CLI-S2-Click-Button1-Open**
9229	Right Click	Button2	Carmelo Romero		CLI		sc18.png	**3**	**S5**	**CLI-S5-Right Click-Button2-**
9230	Click	Button2	Payments	**Payments**	CLI		sc5.png	**1**	**S3**	**CLI-S5-Right Click-Button2-Payments**
9231	Click	Button3	See cases	**See cases**	CMS		sc6.png	**2**	**S1**	**CMS-S1-Click-Button3-See cases**
9232	Click	TextBox3			CLI		sc7.png	**2**	**S2**	**CLI-S2-Click-TextBox3-**
9233	Keystroke	TextBox3			CLI	Manfred	sc8.png	**2**	**S2**	**CLI-S2-Keystroke-TextBox3-**
9234	Click	Button1	Open	**Open**	CLI		sc9.png	**2**	**S2**	**CLI-S2-Click-Button1-Open**
9235	Click	TextBox2			MAIL		sc10.png	**2**	**S4**	**MAIL-S4-Click-TextBox2-**
9236	Keystroke	TextBox2			MAIL	manfred.reichert @uni-ulm.de	sc11.png	**2**	**S4**	**MAIL-S4-Keystroke-TextBox2-**
9237	Click	TextBox1			MAIL			**2**	**S4**	**MAIL-S4-Click-TextBox1-**
9238	Keystroke	TextBox1			MAIL	Dear Manfred....	sc12.png	**2**	**S4**	**MAIL-S4-Keystroke-TextBox1-**
9239	Click	Button2	Payments	**Payments**	CLI		sc13.png	**2**	**S3**	**CLI-S5-Right Click-Button2-Payments**
9240	Click	Button3	See cases	**See cases**	CMS		sc14.png	**3**	**S1**	**CMS-S1-Click-Button3-See cases**
9241	Click	TextBox3			CLI		sc15.png	**3**	**S2**	**CLI-S2-Click-TextBox3-**
9242	Keystroke	TextBox3			CLI	Carmelo	sc16.png	**3**	**S2**	**CLI-S2-Keystroke-TextBox3-**
9243	Click	Button1	Open	**Open**	CLI		sc17.png	**3**	**S2**	**CLI-S2-Click-Button1-Open**
9244	Right Click	Button2	Irene Lingard		CLI		sc18.png	**3**	**S5**	**CLI-S5-Right Click-Button2-**

Fig. 2. Example of Extended UI Event Log.

Based on this, we generated an entire dataset, i.e., an event log —aligned with the proposal presented in [2]— simulating the execution of the process with its different variants.[1]

3.2 Image-Similarity Clustering

Before applying process discovery algorithms, the previous log may include case IDs —identifying different instances of the process—, and activity IDs —identifying different activities. To facilitate this distinction, we leverage the *screenID* attribute, as introduced in [16]. Two events share the same screenID if their screenshots exhibit *similarity.* At a glance, to calculate the screenID (1) all screenshots are hashed into an array of bits (aka. fingerprint), (2) the hashes are clustered using a similarity function called Hamming distance [15] (i.e., counting the number of bits two hashes differ), and then (3) two screenshots are considered similar if they are in the same cluster. In [16], this screenID was used to identify the activities, which means that each low-level event was related to a higher-level group. Altogether, the considered UI event log attributes can be grouped [18,24] in process-related attributes (*timestamp*, *caseID*, *screenID*), context attributes (*actionType*, *softApp*, *screenShot*, *UIElementID*, *UIElementContent*), and data attributes (*inputValue*). Figure 2 shows an example UI event log related to 3 cases of the process detailed in [7]. The 3 cases were collected by monitoring the behavior and adding the derived attributes, i.e., it is a subset of the generated dataset. Note that the highlighted columns (i.e., *UIElementContentHF* and *ActivityID*) are also included as part of the proposed approach (cf. Sect. 4.1).

[1] The dataset and the related mockups of the IT system are available in [7].

3.3 Process Model Discovery

For applying a process mining algorithm to the UI event log[2], the activity and case columns need to be identified. The model discovered from the UI event log, considering *screenIDs* as activities, shows the flow between screens, but is missing the detailed actions performed on them and lacking of semantics in the activities. This approach (denoted as *FlatScreen* from now on) is used as one of the baselines (cf. Sect. 4.2) and considered in the evaluation (cf. Sect. 5). A more recent approach [21] allows discovering models with a hierarchical representation, which effectively bridges the gap between low-level operational details and high-level process insights. However, the efficacy of this approach significantly depends on the incorporation of business knowledge for the meaningful grouping of low-level activities. [21] further allows for randomly grouping components of (sub)processes. This approach (denoted as *RandomHier* from now on) is also used as one of the baselines (cf. Sect. 4.2) and considered in the evaluation (cf. Sect. 5). In conclusion, although hierarchical mining offers a more comprehensive perspective, it poses the following challenges:

1. The absence of business knowledge can significantly hinder the ability to group low-level activities in a meaningful way. This knowledge is crucial for understanding and accurately representing the relationships within the process as well as for properly dealing with ambiguity.
2. When event logs lack descriptive labels, the process model might not truly reflect the underlying process. Descriptive labels are essential for providing context and clarity, enabling the hierarchical model to capture the process accurately.

4 From UI Event Logs to Two-Level BP Models

As mentioned, improving the analysis phase of an RPA project involves tackling distinct challenges. These encompass dealing with (1) low-level events from front-end user interfaces to create a multi-granular and easily understandable model, and (2) resolving ambiguities in inferred user intentions. To tackle these challenges, we propose a method for uncovering two-level BP models from UI event logs. Our approach comprises following key steps:

1. Generating extended UI event logs (cf. Sect. 4.1). This step entails enriching the properties of each event by incorporating additional information necessary for subsequent stages.
2. Discovering hierarchical BP models (cf. Sect. 4.2). This step outlines the process of uncovering two-level BP models from the extended UI event log. It elucidates how activities at various granularity levels are labeled to enhance understandability.

[2] We consider using Inductive Miner due to its ability to guarantee sound process models and its balance to capture essential process behavior while maintaining model simplicity [17].

Algorithm 1. Generating Extended UI Event Logs

```
1: procedure ExtendUILog(Input: UIEventLog, Output: Modified UIEventLog)
2:     $addAttributes(UIEventLog, "UIElementContentHF", "ActivityID")$
3:     $freqThreshold \leftarrow calculateFreqThres(UIElementContent, UIEventLog)$
4:     for each entry $e$ in UIEventLog do
5:         if frequency($e.UIElementContent, UIEventLog$) $\geq freqThreshold$ then
6:             $valueUIElementContentHF \leftarrow e.UIElementContent$
7:         else
8:             $valueUIElementContentHF \leftarrow \emptyset$
9:         end if
10:        $valueActivityID \leftarrow concatenate(e.softApp, e.screenID, e.actionType, e.UIElementID, valueUIElementContentHF)$
11:        $extend(UIEventLog, e, valueUIElementContentHF, valueActivityID)$
12:    end for
13: end procedure
```

4.1 Generating Extended UI Event Logs

Before delving into the discovery of two-level BP models, several preparatory steps are essential including the extension of the UI event log (cf. Algorithm 1). Specifically, the creation of a distinct identifier for user actions, denoted as *ActivityID* attribute, shall represent the activities at the most fine-grained level (Line 2 of Algorithm 1). This identifier serves a dual purpose—supporting process discovery at the user action level and providing semantics to activities in the discovered model.

To formulate the user action ID, we must identify relevant properties within the log while ignoring irrelevant ones. As stated in [2], a user action is defined by both the performed action (i.e., the *actionType* attribute) and the target object the action is executed on. In the considered event log, this target object can be unambiguously identified by attributes *UIElementID*, *softApp*, and *screenID*. While these attributes suffice for identifying user actions, adding semantics to related activities in discovered models can benefit from including the *UIElementContent* attribute. Specifically, the contents of *UIElementContent* serve a dual role in describing user actions. When it involves elements associated with the consistent labeling of UI components (e.g., button labels like "See cases" or "Open"), its content becomes essential for conveying meaning. In such cases, this content frequently occurs in the log. Conversely, when *UIElementContent* contains data-dependent elements (e.g., names or surnames), its contribution to action descriptions is minimal, with infrequent occurrences. To selectively consider the content crucial for describing user actions, a new attribute, *UIElementContentHF*, is introduced (Line 2 of Algorithm 1). This attribute replicates the value of *UIElementContent* only when its frequency in the UI event log exceeds a threshold calculated from the input data, otherwise it remains empty (Lines 3–9 of Algorithm 1). This inclusion is based on the consideration that the content is relevant for providing semantics to the associated user action. Altogether, the new attribute *ActivityID* is computed by concatenating the values

Algorithm 2. Discovering Two-level BP Models

```
1: procedure DiscoverHierModels(Input: Extended $UIEventLog$, Output: Two-level BP Model $hierModel$)
2:     $screens \leftarrow getDifferentScreens(UIEventLog)$
3:     for each $screen$ in $screens$ do
4:         $newLabelScreen \leftarrow generateLabelScreen(UIEventLog, screen)$
5:         $replace(UIEventLog, screen, newLabelScreen)$
6:     end for
7:     $highLevelModel \leftarrow discover(UIEventLog, "ScreenID")$
8:     $hierModel \leftarrow createTwoLevelModelHL(highLevelModel)$
9:     for each $highLevelAct$ in $highLevelModel$ do
10:        $filteredLog \leftarrow filterLog(UIEventLog, highLevelAct)$
11:        $lowLevelModel \leftarrow discover(filteredLog, "ActivityID")$
12:        $replaceLabels(lowLevelModel)$
13:        $addSubprocess(hierModel, highLevelAct, lowLevelModel)$
14:    end for
15: end procedure
```

from *softApp*, *screenID*, *actionType*, *UIElementID*, and *UIElementContentHF* (Line 10 of Algorithm 1), see Fig. 2 for examples. Subsequently, the event information is expanded by incorporating the previously computed values for both *UIElementContentHF* and *ActivityID* (Line 11 of Algorithm 1).

When we generate the fine-grained model by applying process mining directly to the extended UI event log using *ActivityID* as the activity ID, this results in a model resolving ambiguity, but lacking semantics. However, the model discovered when considering *screenID* as activity ID is more coarse-grained, but presents ambiguity issues and lacks label semantics.

4.2 Discovering Two-Level BP Models

This section delineates the procedure for discovering two-level BP models from the expanded UI event log (cf. Algorithm 2). These models, unlike flat models that can be directly learned from the raw UI event log, enable the analysis of the process at two levels of granularity while assigning semantics to the different activities they encompass to improve understandability.

In the realm of RPA, processes typically entail a sequence of steps or actions executed across various screens or interfaces. Consequently, we find it fitting to construct the process model by treating each screen as an individual activity. To achieve this, first, for the sake of understandability, we label each *screenID* as follows (Lines 2–6 in Algorithm 2): for every sequence of events sharing the same *screenID* within the same case, we take the last one with a non-empty value of *UIElementContentHF*, as it reflects one of the possible goals of the process behind the screen. The set of values obtained for the complete log can be used to define the semantics of the screen. Therefore, we concatenate these values and use them as the label of the screen. Additionally, we add at the beginning the *appName* to each group of *UIElementContentHF* values associated with

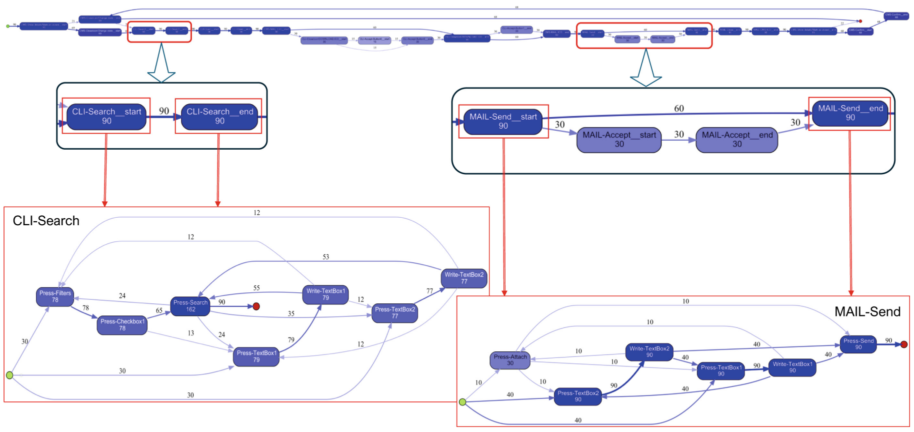


Fig. 3. Two-level model that is discovered for the running example, showcasing subprocesses associated with screens "CLI-Search" and "MAIL-Send".

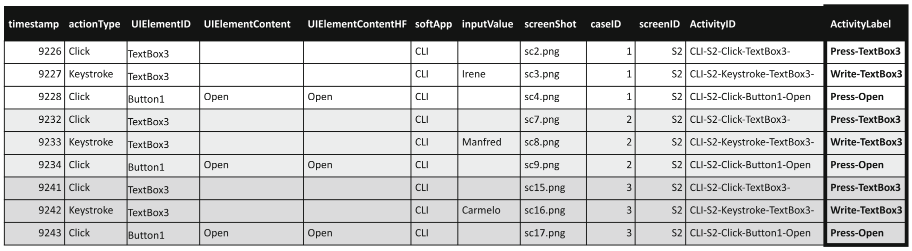
the same application. In case several screens have the same label, we follow the approach presented in [24] and use other attributes to disambiguate the different classes obtained. If multiple screens still have the same label, we use the *UIElementID* attribute of the same event as the *UIElementContentHF* value, which is more screen-specific. Finally, if there are still several screens with the same label, we add the *screenID* value to resolve the ambiguity.

Then, we suggest process mining on an extended UI event log, where the screen ID is treated as the activity ID to unveil a high-level model (Line 7 in Algorithm 2) that shows the flow between screens.

While the preceding step yields a high-level model depicting the flow between screens, the diverse roles within a process entail varying user interests at different levels of detail. Operational teams, for instance, may demand detailed insights at lower granularity levels, facilitating a nuanced understanding of specific user interactions and finer subprocesses. Additionally, ambiguity in inferred user actions might arise from distinct activities with different aims, but being identified as the same activity due to screen similarities.

To tackle both multigranularity and ambiguity, we suggest creating distinct event logs for each screen, exclusively containing the events relevant to that specific screen (Line 10 in Algorithm 2). Subsequently, process mining is applied to the filtered logs, considering attribute *ActivityID* as the activity ID, to unveil the subprocess related to each screen (Lines 11–13 in Algorithm 2).

When labeling the activities in the resulting model, attributes *softApp* and *screenID* are irrelevant as user actions linked to the same screen ID share identical values in these attributes, i.e., these values do not contribute to identify the user actions performed during screen processing. On the contrary, the content of the *actionType* attribute is highly relevant for adding semantics to the user action. Similarly, the *UIElementContentHF* attribute is appropriate for giving semantics to the user action when it contains content (cf. Sect. 4.1). However,



timestamp	actionType	UIElementID	UIElementContent	UIElementContentHF	softApp	inputValue	screenShot	caseID	screenID	ActivityID	ActivityLabel
9226	Click	TextBox3			CLI		sc2.png	1	S2	CLI-S2-Click-TextBox3-	**Press-TextBox3**
9227	Keystroke	TextBox3			CLI	Irene	sc3.png	1	S2	CLI-S2-Keystroke-TextBox3-	**Write-TextBox3**
9228	Click	Button1	Open	Open	CLI		sc4.png	1	S2	CLI-S2-Click-Button1-Open	**Press-Open**
9232	Click	TextBox3			CLI		sc7.png	2	S2	CLI-S2-Click-TextBox3-	**Press-TextBox3**
9233	Keystroke	TextBox3			CLI	Manfred	sc8.png	2	S2	CLI-S2-Keystroke-TextBox3-	**Write-TextBox3**
9234	Click	Button1	Open	Open	CLI		sc9.png	2	S2	CLI-S2-Click-Button1-Open	**Press-Open**
9241	Click	TextBox3			CLI		sc15.png	3	S2	CLI-S2-Click-TextBox3-	**Press-TextBox3**
9242	Keystroke	TextBox3			CLI	Carmelo	sc16.png	3	S2	CLI-S2-Keystroke-TextBox3-	**Write-TextBox3**
9243	Click	Button1	Open	Open	CLI		sc17.png	3	S2	CLI-S2-Click-Button1-Open	**Press-Open**

Fig. 4. Labels given to some user actions related to screen S2.

as this attribute may be empty in some events, the *UIElementID* attribute is considered being a proper substitute for composing the user action name in such situations. Therefore, to name each user action, we propose concatenating the content of the *actionType* and *UIElementContentHF* attributes when *UIElementContentHF* is not empty. Conversely, if *UIElementContentHF* is empty, concatenation involves the *actionType* and *UIElementID* attributes. To improve the clarity of user action labels, we suggest replacing the identifier used for action types (Line 11 in Algorithm 2). Given that the values of this attribute are {Click, Keystroke, RightClick} (consistent with most related works [2]), we find it more descriptive to use the term "Press" in place of "Click" and the term "Write" instead of "Keystroke." Therefore, in the post-composition phase, such replacement is performed to enhance clarity. Figure 4 illustrates the labels assigned to some user actions associated with screen $S2$.

In Fig. 3, which shows the two-level model discovered for the running example when considering the UI event log as input and using the proposed approach (denoted as *ScreenHier* from now on), the high-level model and the labels generated for the high-level activities can be observed. It also contains the models of the subprocesses associated to screens "CLI-Search" and "MAIL-Send". We opted for these screens as illustrations as they represent the two subprocesses with the highest control-flow complexity and size.

Note that with some small adaptations the proposed approach can also be used in combination with FlexHMiner [21] by providing domain knowledge in such a way that user actions are grouped by *screenID* (approach denoted as *KnowScreenHier* from now on). However, FlexHMiner cannot independently discover the models aligned with the previously provided motivation; instead, its functionality is embedded within the framework detailed in this article.

5 Evaluation

As motivated in Sect. 1, this work addresses the following Primary Research Question (PRQ): *Can the proposed approach improve the analysis phase of an RPA project?* This inquiry can be tested by comparing the quality and understandability of process models generated from UI event logs with existing approaches against those produced using the proposed approach. To answer the PRQ, the evaluation explores the following Research Questions (RQs):

(**RQ1**): Do the models that can be generated with the proposed approach describe the process behind an UI event log better than other proposals for generating flat models in the area of RPA?

(**RQ2**): Do the models that can be generated with the proposed approach describe the process behind an UI event log better than other proposals for generating hierarchical models in the area of process mining?

5.1 Case Selection

We assess our approach by applying it to the BP exemplified in Sect. 3. This example is well-suited for evaluation as both the model directly derived by applying process mining to the raw UI event log level and the model obtained by considering *screenIDs* as activity IDs [16] are inadequate for comprehending the underlying process within the UI event log. Moreover, an $n : m$ relationship exists between activities and screen IDs (see [7]), posing a challenge in generating process models from UI event logs (cf. Sect. 1).

5.2 Experiment Design

As explained in Sect. 4.2, the proposed approach gives rise to *ScreenHier* and *KnowScreenHier* (cf. Sect. 4.2)–both discovering two-level models with activity names that carry semantic meaning.

To address RQ1, we compare the results obtained with the proposed approach with those obtained with *FlatScreen* (cf. Sect. 3.3). *FlatScreen* is the most closely related proposal in literature for generating process models from UI event logs in the context of RPA. In the model obtained with *FlatScreen*, the activity names lack semantics.

Regarding RQ2, we assess the results of the proposed approach against those obtained with *RandomHier* (cf. Sect. 3.3). We chose this approach as baseline as it is the most comparable proposal in literature for generating hierarchical process models from event logs. *RandomHier* discovers a hierarchical model by using FlexHMiner[3] [21] in ProM[4], where activities are randomly grouped based on user actions as activity IDs. The resulting model also suffers from a lack of semantics in its activity names.

As response variables for both RQ1 and RQ2 we consider quality and understandability of the generated models. For evaluating the quality of a model M concerning a log L we consider the following quality measures [21]:

– Fitness $Fi(M, L) \in [0, 1]$: represents the proportion of events in the event log that are correctly captured by the model.

[3] It is accessible at github.com/xxlu/prom-FlexHMiner.
[4] http://www.promtools.org/.

- Precision $Pr(M, L) \in [0, 1]$: measures the proportion of activities modeled by process discovery that actually belong to the underlying process.
- F1-score $F1(M, L) \in [0, 1]$ (F-measure [20]): is the harmonic mean of fitness and precision: $F1(M, L) = 2 * \frac{Fi(L,M)*Pr(L,M)}{Fi(L,M)+Pr(L,M)}$.
- Generalization $Ge(M, L) \in [0, 1]$: evaluates how well the discovered model generalizes to new instances or variants of the process that were not present in the original event log. We adopt the same methodology as described in [6,21]. Specifically, we partition the log into $k = 3$ subsets: L is randomly divided into L_1, L_2, and L_3. Subsequently, $Ge(L, M) = \frac{1}{3}\sum_{1\leq i\leq 3} 2 * \frac{Fi(L_i,M_i)*Pr(L,M_i)}{Fi(L_i,M_i)+Pr(L,M_i)}$ with M_i corresponding to L_i.

The evaluation of understandability of a model is intricately linked to its complexity. For this study, following [6,21], we assess complexity using both the control-flow complexity $CFC(M)$ and the size (i.e., number of nodes) $Size(M)$ of a model as response variables.

For the proposals that generate hierarchical models, the values of the response variables are computed as the mean values of the individual measurements for each subprocess within the hierarchical model and the root process.

For the 4 aforementioned approaches, the Inductive Miner algorithm with a 0.2 path filtering threshold is employed for mining the log.

5.3 Dataset

The experiments involve a synthetic UI event log tailored to the selected case[5], created through a method supported by specialized tools [23]. This log encapsulates diverse variations witnessed during the execution of business activities in the running example [7] with corresponding mockups (see [7]). To be more specific, we generate an UI event log comprising 90 cases (30 for each process variant). Each case encompasses an average of 32.73 events (min=23, max=41), featuring 13 distinct screenIDs and 39 unique activityIDs.

5.4 Results

Table 1 shows the values that are obtained for each response variable when considering the aforementioned approaches.

Concerning RQ1 our results show that both newly proposed approaches, i.e., *ScreenHier* and *KnowScreenHier*, exhibit substantial improvement over one of the baselines (i.e., *FlatScreen*) when focusing on both quality and complexity metrics. This enhancement is attributed to hierarchical models usually delivering better results than flat models for these metrics.

Concerning RQ2, *ScreenHier* and *KnowScreenHier* outperform *RandomHier* in general when focusing on quality and complexity metrics. This superiority arises from the consideration of screens for grouping user actions, as proposed

[5] Available at: [7].

Table 1. Values that are obtained for the response variables.

Approach	$\overline{Fi}$	$\overline{Pr}$	$\overline{F1}$	$\overline{Ge}$	$\overline{CFC}$	$\overline{Size}$
FlatScreen	0.571	0.605	0.588	0.595	28.00	52.00
RandomHier	0.964	0.674	0.765	0.809	9.40	31.00
ScreenHier	0.967	**0.839**	**0.885**	**0.888**	5.14	**13.64**
KnowScreenHier	**0.997**	0.813	0.870	0.885	**4.21**	14.43

in this work, leading to two-level models that surpass random grouping. Metrics combining precision and fitness (i.e., F1-score and generalization) also favor *KnowScreenHier* over *RandomHier*.

Additionally, although not empirically demonstrated in this study, providing semantics to activities, as done in *ScreenHier* and *KnowScreenHier*, is likely to further improve the comprehension of the generated models.

5.5 Threats to Validity and Limitations

Concerning *construct validity*, we assert that the conducted experiments align well with the intended objective of assessing the efficacy of the proposed approach in enhancing the analysis phase of an RPA project. The chosen response variables and the generated UI event log are deemed appropriate for addressing research questions *RQ*1 and *RQ*2. Nevertheless, additional response variables and cases could be defined to broaden the analysis performed. Regarding *external validity*, we believe that the study findings can be generalized to scenarios adhering to the outlined assumptions in Sect. 3. Though the proposed approach is described along the running example, its applicability extends to other scenarios that present similar characteristics. The running example encapsulates typical characteristics frequently encountered in common legacy system scenarios. Additionally, the attributes considered for each event in the UI event log align seamlessly with other significant works in the RPA field [2,18,24].

The UI event log in our experiments was synthetically generated from realistic screenshots, introducing potential bias as it simulates expected behavior rather than precisely reflecting real-world scenarios. However, the tool used for screenshot generation has been validated in previous research, closely replicating events and screenshots from the original log. Additionally, the tool's configurability enables the consideration of various generated configurations, helping to mitigate this potential bias. Additionally, we acknowledge the limitation of analyzing results related to only one process scenario. Evaluating the proposed approach in diverse process scenarios of increasing complexity, preferably real ones, would contribute to a more comprehensive understanding of the proposed approach. Moreover, we consider partially evaluating the understandability of the generated models through analyzing their complexity. Nevertheless, we recognize the importance of conducting controlled experiments involving real users to achieve a more comprehensive evaluation of model understandability. Furthermore, to generate the two-level models, we have considered FlexHMiner [21], although there are other options that could be investigated.

6 Summary and Outlook

This paper presents a novel method for extracting two-level BP models from UI event logs. The approach works at two levels: for a comprehensive overview it captures activities at the screen level and delves into detailed user actions for deeper insights. This dual-level model not only support multi-granularity, but helps addressing ambiguities in inferred user intention and improves clarity with labeled activities. The method was validated using synthetically generated UI event logs, demonstrating that the proposed approach yields improved models in terms of both quality and complexity. For future work, we aim to evaluate our approach using real-world scenarios. Additionally, we plan to explore incorporating more levels of granularity in the discovered models and consider UI event logs with supplementary information, such as database logs.

Acknowledgement. This research was supported by the EQUAVEL project PID2022-137646OB-C31 funded by MICIU/AEI/10.13039/501100011033 and by FEDER, UE. ChatGPT was utilized to assist shortening parts of the paper.

References

1. van der Aalst, W.M.: On the pareto principle in process mining, task mining, and robotic process automation. In: DATA, pp. 5–12 (2020)
2. Abb, L., Rehse, J.R.: A reference data model for process-related user interaction logs. In: Di Ciccio, C., Dijkman, R., del Rio Ortega, A., Rinderle-Ma, S. (eds.) Business Process Management. BPM 2022. LNCS, vol. 13420, pp. 57–74. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-16103-2_7
3. Agostinelli, S., Lupia, M., Marrella, A., Mecella, M.: Reactive synthesis of software robots in RPA from user interface logs. Comput. Ind. **142**, 103721 (2022)
4. 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
5. Agostinelli, S., Marrella, A., Mecella, M.: Exploring the challenge of automated segmentation in robotic process automation. In: Cherfi, S., Perini, A., Nurcan, S. (eds.) RCIS 2021. LNBIP, vol. 415, pp. 38–54. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75018-3_3
6. Augusto, A., et al.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2018)
7. Barba, I., Del Valle, C., Jimenez-Ramirez, A., Weber, B., Reichert, M.: Example of a business process with three process variants: mockups and log (2024). https://doi.org/10.5281/zenodo.10730799
8. Beerepoot, I., et al.: The biggest business process management problems to solve before we die. Comput. Ind. **146**, 103837 (2023)

9. Choi, D., R'bigui, H., Cho, C.: Candidate digital tasks selection methodology for automation with robotic process automation. Sustainability **13**(16), 8980 (2021)
10. Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: BPMN miner: automated discovery of BPMN process models with hierarchical structure. Inf. Syst. **56**, 284–303 (2016)
11. Dogan, O., de Leoni, M.: Parallelism-based session creation to identify high-level activities in event log abstraction. In: De Smedt, J., Soffer, P. (eds.) Process Mining Workshops. ICPM 2023. LNBIP, vol. 503, pp. 58–69. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-56107-8_5
12. Dumas, M., La Rosa, M., Leno, V., Polyvyanyy, A., Maggi, F.M.: Robotic process mining. Process Min. Handb. 468–491 (2022)
13. Folino, F., Guarascio, M., Pontieri, L.: Mining multi-variant process models from low-level logs. In: Abramowicz, W. (ed.) BIS 2015. LNBIP, vol. 208, pp. 165–177. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19027-3_14
14. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 128–139. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12186-9_13
15. Gusfield, D.: Algorithms on stings, trees, and sequences: computer science and computational biology. ACM SIGACT News **28**(4), 41–60 (1997)
16. 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
17. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
18. Leno, V., Augusto, A., Dumas, M., La Rosa, M., Maggi, F.M., Polyvyanyy, A.: Identifying candidate routines for robotic process automation from unsegmented UI logs. In: ICPM, pp. 153–160 (2020)
19. Leno, V., Polyvyanyy, A., Dumas, M., La Rosa, M., Maggi, F.M.: Robotic process mining: vision and challenges. Bus. Inf. Syst. Eng. **63**, 301–314 (2021)
20. Liu, C., Cheng, L., Zeng, Q., Wen, L.: Formal modeling and discovery of hierarchical business processes: a petri net-based approach. IEEE Trans. Syst. Man Cybern. Syst. (2023)
21. Lu, X., Gal, A., Reijers, H.A.: Discovering hierarchical processes using flexible activity trees for event abstraction. In: Proceedings of the ICPM, pp. 145–152. IEEE (2020)
22. Macías, J.A.: Intelligent assistance in authoring dynamically generated web interfaces. World Wide Web **11**, 253–286 (2008)
23. Martínez Rojas, A., Jiménez Ramírez, A., González Enríquez, J., Reijers, H.A.: A tool-supported method to generate user interface logs. In: Proceedings of the HICSS, pp. 5472–5481 (2023)
24. Rebmann, A., van der Aa, H.: Unsupervised task recognition from user interaction streams. In: Indulska, M., Reinhartz-Berger, I., Cetina, C., Pastor, O. (eds.) Advanced Information Systems Engineering. CAiSE 2023. LNCS, vol. 13901, pp. 141–157. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-34560-9_9
25. Rehse, J.-R., Fettke, P.: Clustering business process activities for identifying reference model components. In: Daniel, F., Sheng, Q.Z., Motahari, H. (eds.) BPM

2018. LNBIP, vol. 342, pp. 5–17. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11641-5_1
26. Sani, M.F., Sroka, M., Burattin, A.: LLMS and process mining: challenges in RPA task grouping, labelling and connector recommendation. In: De Smedt, J., Soffer, P. (eds.) Process Mining Workshops. ICPM 2023. LNBIP, vol. 503, pp. 379–391. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-56107-8_29
27. Seiger, R., Franceschetti, M., Weber, B.: An interactive method for detection of process activity executions from IoT data. Futur. Internet **15**(2), 77 (2023)
28. Wang, Y., Wen, L., Yan, Z., Sun, B., Wang, J.: Discovering BPMN models with sub-processes and multi-instance markers. In: Debruyne, C., et al. (eds.) OTM 2015. LNCS, vol. 9415, pp. 185–201. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26148-5_11
29. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granul. Comput. **6**(3), 719–736 (2021)
30. Zerbato, F., Seiger, R., Di Federico, G., Burattin, A., Weber, B.: Granularity in process mining: can we fix it? In: CEUR Workshop Proceedings, vol. 2938, pp. 40–44 (2021)