Rataplan: Resilient Automation of User Interface Actions with Multi-modal Proxies

TOM VEUSKENS, KRIS LUYTEN, RAF RAMAKERS, Hasselt University - tUL - Flanders Make Expertise Centre for Digital Media

We present Rataplan, a robust and resilient pixel-based approach for linking multi-modal proxies to automated sequences of actions in graphical user interfaces (GUIs). With Rataplan, users demonstrate a sequence of actions and answer human-readable follow-up questions to clarify their desire for automation. After demonstrating a sequence, the user can link a proxy input control to the action which can then be used as a shortcut for automating a sequence. Alternatively, output proxies use a notification model in which content is pushed when it becomes available. As an example use case, Rataplan uses keyboard shortcuts and tangible user interfaces (TUIs) as input proxies, and TUIs as output proxies. Instead of relying on available APIs, Rataplan automates GUIs using pixel-based reverse engineering. This ensures our approach can be used with all applications that offer a GUI, including web applications. We implemented a set of important strategies to support robust automation of modern interfaces that have a flat and minimal style, have frequent data and state changes, and have dynamic viewports.

CCS Concepts: • **Human-centered computing → User interface programming**; Accessibility systems and tools.

Additional Key Words and Phrases: tangible user interfaces, pixel-based reverse engineering, UI automation, programming-by-demonstration

ACM Reference Format:
Tom Veuskens, Kris Luyten, Raf Ramakers. 2020. Rataplan: Resilient Automation of User Interface Actions with Multi-modal Proxies. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 2, Article 60 (June 2020), 23 pages. https://doi.org/10.1145/3397329

1 INTRODUCTION

Over the past decade, various mobile and desktop applications became available to support activities throughout our daily lives, including reading, music, cooking, sports, and working. To speed up interactions, these applications often offer visual and keyboard shortcuts to frequent actions. For automating custom or personalized action sequences, macro recorders, such as Autohotkey [21] and JitBit [17], became available that simply record and replay mouse interactions using absolute coordinates. In contrast, Sikuli [31] and Prefab [7] support automating more dynamic interfaces as they record and match visual elements, an approach referred to as pixel-based reverse engineering [7]. This approach is often combined with "programming-by-demonstration (PbD)" techniques to allow users to automate actions by simply demonstrating them [1, 16].

Traditional pixel-based reverse engineering approaches work well for interfaces that conform to well-known UI standards and consist of basic UI elements. Examples include skeuomorphic interfaces, and standardized

Author's address: Tom Veuskens, Kris Luyten, Raf Ramakers, Hasselt University - tUL - Flanders Make Expertise Centre for Digital Media, Diepenbeek, Belgium, firstname.lastname@uhasselt.be.

2474-9567/2020/6-ART60 $15.00
https://doi.org/10.1145/3397329

UI themes, such as the Windows Areo style[1]. Over the past decade however, Graphical User Interfaces (GUIs) have evolved from being skeuomorphic to having a flat and minimal design [5, 27]. Examples include the latest versions of Slack, Gmail, Skype, and Spotify. This shift makes it challenging for state-of-the-art pixel based UI automation tools [1, 7, 16, 31] to reliably automate behavior. For example, it is not possible to automate the archive, delete, unread, and snooze mail buttons in Gmail using only visual inspection as these buttons only become visible when hovering an email. Similarly, state-of-the-art UI automation tools like Prefab [7] extract UI elements and interface hierarchies using borders of UI elements, frames, and dividers. However, these borders are not prominent or sometimes not present at all in flat interface styles. Finally, application data is updated frequently or even continuously retrieved from the cloud in modern interfaces. This makes it very hard for state-of-the-art UI automation tools to reliably automate actions as the content, used as identifiers, changes. For example, automating the action sequence to play a song in the top charts of Spotify is challenging as that song might become less popular and move outside the view port. While automating all these interactions should be possible in advanced programming tools like Sikuli [31], this would require users to write advanced automation scripts to handle all possible cases and interruptions.

In this paper we present Rataplan, a robust and resilient pixel-based approach for automating advanced action sequences in modern user interfaces. With our approach, users demonstrate sequences of actions and answer human-readable follow-up questions to refine the intent of demonstrations (Figure 1a). In the most basic configuration, demonstrated sequences are executed according to the demonstration. Going further, Rataplan's follow-up questions aim to capture more detailed specifications of the goals and intended behavior of the UI automation, such as parameters the user want to use and the intended number of repetitions of an action sequence. Rataplan advances state-of-the-art approaches in UI automation [1, 15, 16] in two ways: (1) Rataplan contributes a *contingency plan to recover from unexpected scenarios*, such as application data and state changes often present in modern user interfaces, and (2) Rataplan avoids requiring up-front complex specifications, by performing a detailed graphical UI analysis and asking accessible and human-readable follow-up questions. We believe these aspects are crucial for automating advanced sequences over extended periods of time without frequent user intervention. Using Rataplan, a wide variety of people without programming expertise, including caregivers, lab assistants, and designers, are empowered to specify and execute GUI automation sequences in a resilient way, thus facilitating user-driven task automations. In this paper, we use Tangible User Interfaces (TUIs) and keyboard shortcuts as possible ways of interacting with automated behaviour in Rataplan. However, Rataplan can be extended to include support for other interaction techniques, such as gestures, speech, and sensor-based triggering. We showcase the robustness of our approach and demonstrate the utility and usability of Rataplan through several use cases, example designs, and a user evaluation.

2 WALKTHROUGH

In this walkthrough, we automate several actions in the Spotify application using Rataplan. As we show later, Rataplan can be used to automate any application offering a GUI, including web applications. To execute automated behaviour from ubiquitous locations, Rataplan offers a set of tangible controls which can be linked to a demonstrated action sequence. While we use these tangible controls in this walkthrough to showcase our approach, we would like to emphasize Rataplan can be extended with additional interaction techniques.

John has a physical disorder, which makes it hard for him to precisely control Spotify through a regular computing device, such as a desktop. As John would like to enjoy the convenience of his favorite music player Spotify, he asks his caregiver Emma to make a tangible interface for some of the functions in Spotify.

[1] https://docs.microsoft.com/en-us/windows/win32/controls/visual-styles-overview

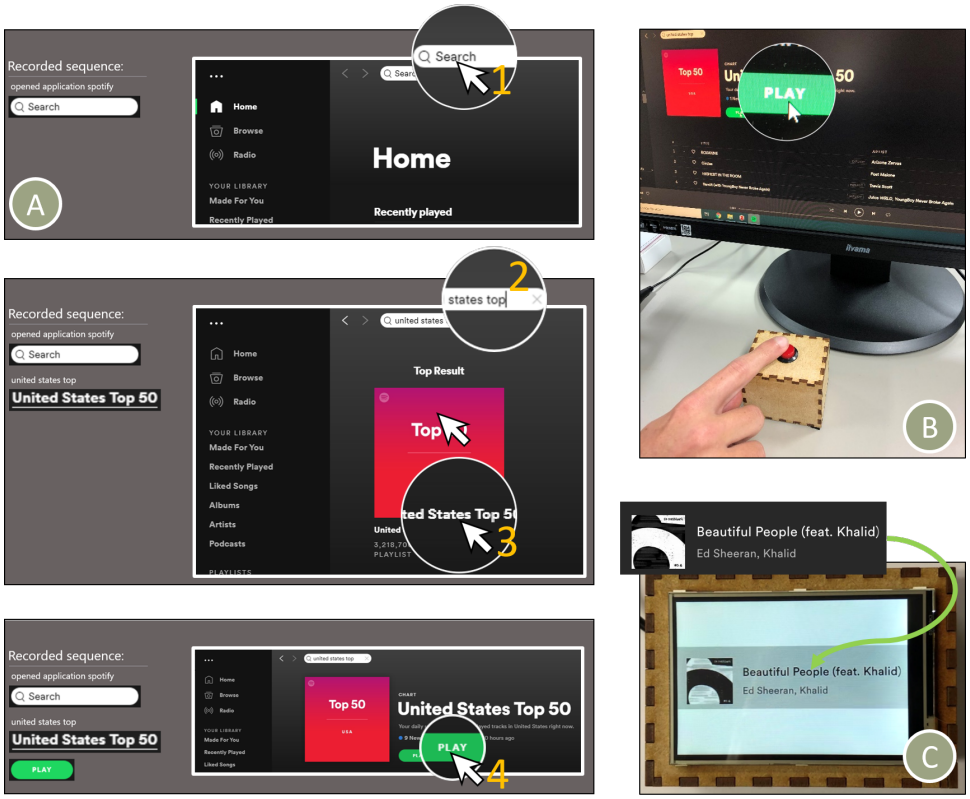

Fig. 1. Using Rataplan to couple tangible controls to Spotify. (a) The user demonstrates to Rataplan how to navigate to and play the "United States Top 50" playlist. (b) When interacting with Rataplan controls, the demonstrated action is replayed by Rataplan, regardless of which state the computer is in. (c) Rataplan also provides output, here the "Currently playing" region is being monitored by Rataplan.

2.1 Basic UI Automation

John requests a tangible control to play his favorite playlist "United States Top 50". Emma starts the Rataplan system, starts a new recording, and demonstrates all mouse and keyboard actions to start the playlist after opening Spotify through Rataplan's homescreen, as shown in Figure 1a. When Emma picks up a Rataplan push-button, the system automatically recognizes the tangible being manipulated. Emma clicks the play button in the Rataplan interface while holding the physical control to link them together. Next, a new recording is started and the play/pause button in Spotify is pressed and linked to another Rataplan push-button. Rataplan detects that the visual state of the play button changes into pause after being pressed, and suggests configuring the Rataplan push-button as a toggle switch. John also wants to control the volume in Spotify, but because of his impairment he is unable to precisely manipulate a physical slider. Emma therefore decides to link three Rataplan push-buttons to the volume slider. After demonstrating in Rataplan how to navigate to the slider, the slider is enlarged by Rataplan. This allows Emma to precisely link Rataplan push-buttons to specific locations on the slider. She links one button to 0%, one to 50%, and one to 100% of the slider, so John can choose between three volume levels with

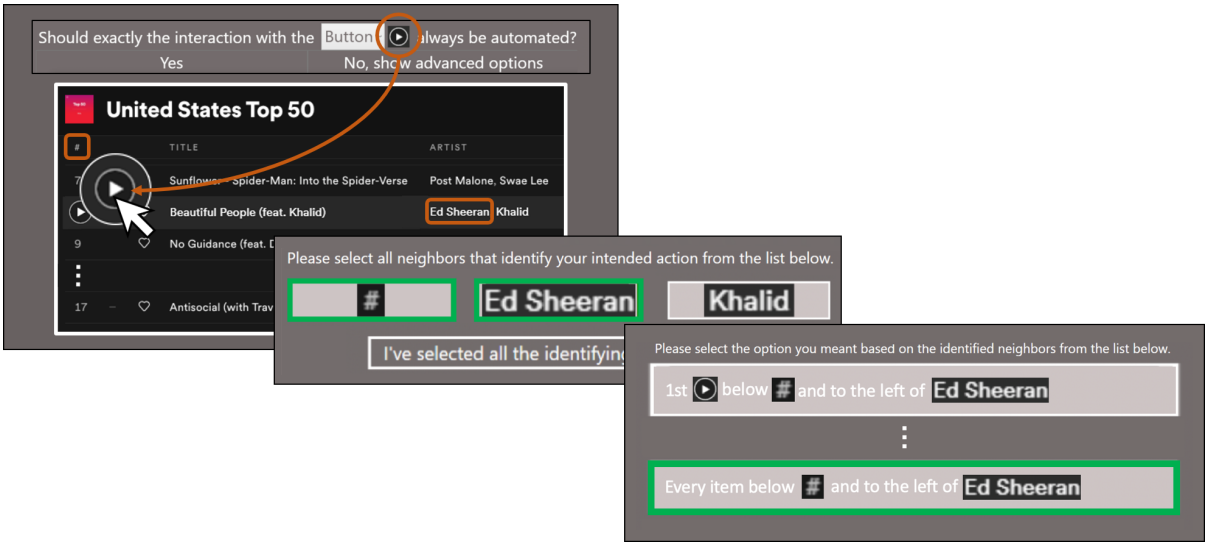

Fig. 2. Using Rataplan's advanced options to play every song of "Ed Sheeran" in the top 50 playlist.

the click of a button. The Rataplan tangibles now offer control over a few essential features in Spotify without forcing John to use a device that requires precise input, such as a mouse, keyboard, or touch (Figure 1b).

2.2 Parametric UI Automation

After a while, John wants to explore more music from his favorite artist "Ed Sheeran". He therefore asks Emma to reconfigure the button that currently starts the playlist "United States Top 50". Emma decides to sequentially play all songs of "Ed Sheeran" in the top charts. Emma again demonstrates the actions to navigate to the playlist, but instead of clicking the "Play" button for the playlist, she clicks the play button next to a song of "Ed Sheeran" (Figure 2). Emma uses Rataplan's advanced options to specify that she does not want to automate playing this specific song, but every song matching "Ed Sheeran" in the playlist. As shown in Figure 2, Rataplan therefore extracts nearby UI elements and compiles possible filters from which Emma selects the desired one. Through follow-up questions, she configures the button to go to the next song when pressed again. As the button is configured to play different songs from "Ed Sheeran", Emma thinks it is convenient to allow John to see which song is currently playing. Therefore, Emma starts a last recording and selects the "Currently Playing" region in the bottom region of Spotify's UI while physically touching a Rataplan display. Rataplan now continuously monitors this region and forwards updates to the remote Rataplan display (Figure 1c).

After completing these steps, John can take the tangibles anywhere in the house to control Spotify. Even when Spotify is not active or is in a different state, Rataplan will initiate strategies to change the state of the application and perform the desired action as we will discuss in the next sections.

3 RELATED WORK

This work draws from, and builds upon prior work on techniques for automating and enhancing user interfaces. We differentiate between two classes of UI-automation techniques: (1) automation techniques that require an understanding of the implementation and workings of the applications, and (2) automation techniques that

extract and analyze the visual pixels of the interfaces, oftentimes referred to as pixel-based reverse engineering approaches [7].

3.1 Implementation-Specific Automation Methods

Over the years, multiple new approaches have been proposed for automating user interfaces. Stuerzlinger et al. [28] shows how to automate and reverse engineer interfaces that implement accessibility APIs. However, Hurst et al. [14] found that on average 26% UI elements of an application are not available through these APIs. Therefore, hybrid approaches have been presented to augment missing information from accessibility APIs with information extracted from analyzing the pixels using computer vision techniques (pixel-based reverse engineering) [6, 14]. In contrast to these approaches, Rataplan is primarily a pixel-based reverse engineering technique that does not use accessibility APIs since these are often not implemented consistently across platforms and applications [14].

A number of approaches have been presented which allow for run-time modifications in applications by injecting code and libraries into third party applications to retrieve hierarchy, track user interactions, and adapt behavior [10, 11, 22, 23]. These approaches, however, only work when specific UI toolkits are used, and all relevant UI elements are required to be visible when opening the application as only this initial state is analyzed.

Besides desktop UI automation, several approaches are proposed to automate mobile GUIs [20] and actions in browsers [3, 18, 19]. While the first approach relies on Android's accessibility API, the latter approaches use web-specific APIs. While SUGILITE [20] does not assume the system is in exactly the same state during demonstration and automation phases, as for example automated actions can be interrupted by external events such as a phone call, they require user intervention to manually resolve the ambiguity. In contrast, Rataplan automatically deals with changes in the state of the GUI.

3.2 Pixel-Based Reverse Engineering

The methods discussed in the previous section depend on applications implemented in a specific toolkit, or application developers implementing specific (accessibility) APIs. This, however, poses limitations as users cannot automate arbitrary applications. Applications are oftentimes implemented in a wide variety of toolkits, and as identified by Hurst et al. [14], on average 26% of UI elements are missing in accessibility APIs. Instead of depending on underlying implementations of applications to augment or automate them, another approach is to operate solely on the GUI of an application. The main advantage of operating on the GUI directly is that arbitrary applications can be automated and enhanced. Commercially available macro recorders [17, 21, 24], offer basic UI automation features by repeatedly executing recorded mouse and keyboard input on the GUI. However, the automation sequence fails when interface elements, or their position, change as these tools do not analyze and interpret the state of the interface. By analyzing and interpreting the GUI at pixel-level, more robust automation sequences can be specified. This technique is oftentimes referred to as pixel-based reverse engineering [7].

Several pixel-based reverse engineering techniques have been presented to automate and enhance GUIs, independent of the underlying platform or applications. Prefab [7–9] uses computer vision and machine learning to recognize widget types and retrieve the UI hierarchy. However, these techniques require training the system for every UI theme, and do not work reliably when UI elements have no clear borders. This is often the case in flat and minimal designs, oftentimes used in modern interfaces. In contrast, Sikuli [31] does not require training the system and offers a visual scripting language to empower users to automate GUIs by combining programming constructs and screenshots. In Sikuli, advanced automation sequences, such as loops, parametric actions, and GUI variations can be specified. However, programming knowledge is required, as well as a careful analysis of the various GUI states, to write robust automation scripts. Sikuli Slides [1] offers an alternative to programming by visually annotating screenshots. This approach is, however, limited to basic UI automation without support for advanced automation sequences, such as loops, parameters, and GUI variations. Similar to our approach,

Fig. 3. Using supporters in HILC. (a) The Wi-Fi toggle can be identified by identifying both labels to say it is in between. (b) HILC asks the user to differentiate between the "Previous" and "Next" buttons, which is intricate to pick the correct supporter for.

"Help, it looks confusing" (HILC) [16] offers a pixel-based automation tool and a programming-by-demonstration paradigm with follow-up questions to understand end-users' desires for automation. In contrast to our approach however, many questions in HILC are complex and not human-readable. Furthermore, this system does not support a contingency plan to recover from UI changes.

While Sikuli Slides [1] and HILC [16] provide novice end-users with tools to specify automation behaviour without relying on technical knowledge, they make several assumptions to facilitate this process. However, as we will show, these assumptions do unfortunately not always hold true in modern user interfaces, and therefore can fail when executing the resulting automation sequences. To successfully support resilient UI automation for modern interfaces by novice end-users, we did a thorough analysis of state-of-the-art visual UI automation approaches, including Sikuli Slides [1] and HILC [16]. Four issues present in current automation tools are identified that make state-of-the-art tools not resilient when dealing with modern user interfaces.

(1) **Ambiguities**: Oftentimes, similar or identical looking items are present in interfaces. For example, multiple identical toggle buttons can be present for controlling different settings (Figure 3a). HILC requires the user to define supporters that have clear geometric relations to these duplicate UI elements. In the example in Figure 3a, the Wi-Fi label and Bluetooth label can both be supporters for the first toggle as it surrounds the control. However, selecting appropriate supporters is not always straightforward: consider automating the "Next" button in Spotify. HILC asks to differentiate between the "Previous" and the "Next" button (Figure 3b). One would assume the "Play" button in between both matches can be used as an identifier. However, this button changes into pause frequently and therefore will not be sufficient for robust automation. As a result, the correct supporter to differentiate the "Next" button is the "Previous" button, which is intricate as it is already annotated as a false-positive match (red square in Figure 3b).
(2) **Hidden information**: In modern interfaces, information or UI elements can be partially or completely hidden. First of all, information can be positioned outside of the current viewport, such as mails becoming visible when scrolling the interface. Additionally, UI elements can be hidden until certain parts of the interface are hovered. For example, the archive function in Gmail becoming visible when hovering a mail (Figure 4a), or the thumb of the volume slider in Spotify only showing when hovering. HILC and Sikuli slides rely exclusively on elements that are always visible and thus do not support automating information that might be hidden during execution. Finally, since interfaces oftentimes have different panels, resizing them can cause certain elements to become partially invisible or cropped (Figure 4b). HILC and Sikuli Slides are oftentimes not successful in finding targets that are cropped.
(3) **Interface interruptions**: While demonstrating or recording an action, the action sequence can be interrupted by unrelated messages, such as OS or application notifications, advertisements, and update requests. During demonstration, closing these interruptions adds noise to specification, which are not supposed to be executed upon playback. Alternatively, the execution of an automated action sequence can be hindered when

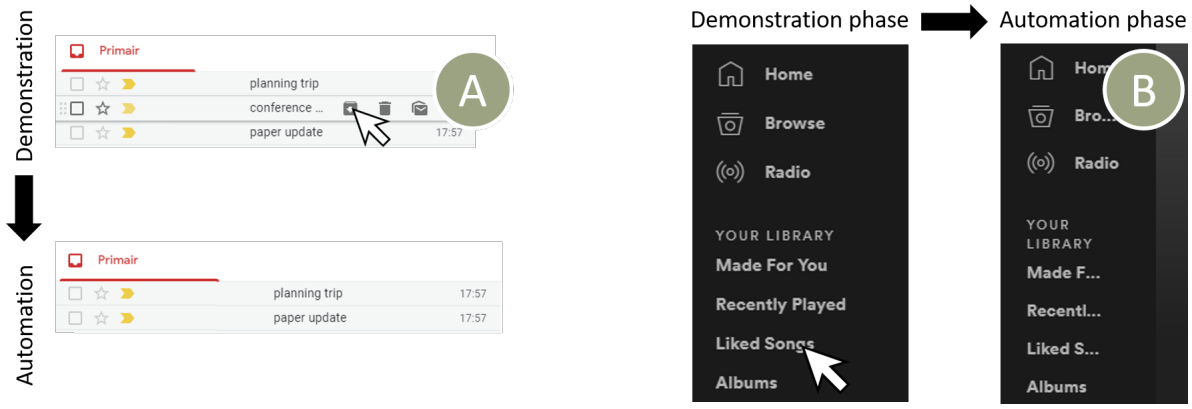

Fig. 4. Hidden information in modern interfaces. (a) Action buttons for list items only become visible when hovering a list item. (b) Visual appearance of UI elements changes as they get cropped due to resizing.

notifications show up while executing recorded sequences. While not supported in Sikuli Slides, in HILC one could manually specify a monitoring script for all notifications that can possibly halt the replay. In practice however, this is cumbersome and oftentimes unfeasible.

(4) **Application management**: In current desktop environments, users work across multiple applications which are frequently opened, closed, and minimized. As a result, applications can be closed, minimized, or in a different state when triggering the execution of a recorded action sequence. In HILC and Sikuli Slides, these situations are not supported and every application must be in the exact same state as during the demonstration of the action sequence.

Table 1 offers a summary and additional details on shortcomings of state-of-the-art approaches, including HILC and Sikuli slides. Additionally, we briefly highlight how Rataplan supports these important and frequently encountered cases to allow for robust and resilient automation of modern GUIs. Section 4 elaborates on these novel features.

4 RATAPLAN UI AUTOMATION

Rataplan's UI automation consists of three phases. First, the user demonstrates an action sequence in the *demonstration phase*. Next, our system asks follow-up questions during the *specification phase* to fine-tune the recorded sequence, retrieve missing details, and resolve ambiguities. Finally, the automated UI sequence can be executed repeatedly and triggered during the *automation phase*.

4.1 Demonstration Phase

Rataplan's home screen consists of all applications installed on the computer. Rataplan requires users to start programs from the Rataplan home screen. To enforce this, the Rataplan home screen runs in full-screen mode by default. Recording a new action sequence starts with selecting one or multiple applications. After Rataplan starts these applications, the user demonstrates the action sequence which requires automation by simply demonstrating the interactions. Rataplan records all keystrokes and all mouse events, such as clicking menu items and manipulating UI controls. Actions across multiple applications, such as copying data or files, are also supported within the Rataplan environment. After the demonstration is finished, the user links one or more triggers to the recorded action sequence, e.g. keyboard shortcuts or tangible controls (Section 5).

Table 1. Comparison between Sikuli slides [1], HILC [16], and Rataplan with respect to resilient automation for modern UI's.

	Sikuli Slides	HILC	**Rataplan**
Ambiguity resolution	Not supported - cannot differentiate between similar UI elements	Additional user input during demonstration (spatial supporter) or during automation (if ambiguity was not present during demonstration). Not always straightforward (Figure 3b)	Automatic Support - expanding search region to level of neighboring UI elements until ambiguity is resolved
Hidden information	- *Completely hidden elements*: not supported - unable to find UI element. - *Partially hidden elements*: not supported - triggers a false positive match or finds no match at all.	- *Completely hidden elements*: not supported - unable to find UI element. - *Partially hidden elements*: not supported - triggers a false positive match or finds no match at all.	- *Completely hidden elements*: supported by exploring all viewports and by looking for hidden information that appear on hover. - *Partially hidden elements*: supported by combining pixel-based recognition and OCR.
Handling interface interruptions	- *During recording:* supported, user can delete slides from recorded set. - *During execution:* not supported - unable to proceed execution.	- *During recording:* not supported - user has to demonstrate the action again. - *During execution*: partially supported with additional user input - users have to write monitoring scripts to handle all possible notifications that could halt the replay.	- *During recording:* supported by allowing users to remove some demonstrated actions. - *During execution*: supported by looking for UI elements to dismiss popups or in worst-case restart the application
Application state management	Partially supported - user has to make sure to demonstrate opening the application	Partially supported - user has to make sure to demonstrate opening the application	Supported by restarting the application for every demonstration

4.2 Specification Phase

Next, when an action sequence is demonstrated, Rataplan asks follow-up questions and additional specifications to better understand the user's intention for automation. During this process, Rataplan repeats and thus automates all demonstrated actions while requesting additional specifications. Before executing an action of the sequence, Rataplan asks whether the interaction with `<widgetType> <snapshot>` should be automated. As shown in Figure 5, the systems replaces `<widgetType>` with the recognized UI element and `<snapshot>` with a screenshot

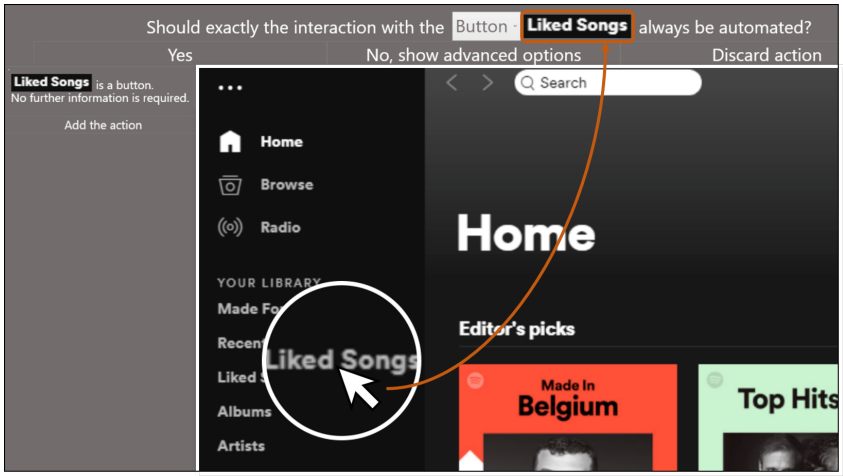

Fig. 5. A human-readable follow-up question asked by Rataplan during specification.

of the extracted target. Rataplan recognizes the basic UI control types, however users can correct these suggestions if necessary using a drop-down menu. As Rataplan understands the type of UI widget, it can immediately link the states of the trigger control to the states of the UI widget. In contrast, state-of-the-art approaches [1, 16] require repeating the entire automation sequence for every state of a widget. For the example of a toggle switch, state-of-the-art approaches require a demonstration of the full sequence, including navigating to the widget and controlling the widget, twice.

When actions cannot be successfully completed during the specification phase, Rataplan suggests discarding that action from the automation sequence. This sometimes occurs when the demonstration was interrupted by unrelated pop-up messages, such as advertisements or operating system update requests.

Requesting advanced options for a demonstrated action offers features to clarify user-intent. A situation that often occurs is the selection of a specific element from a list - does the user desire automating this precise action, execute the action always at this position in the list, or perform the action on every element in the list? We allow the user to generalize actions and define looping behavior to clarify their intent to Rataplan. We explain these features in the remainder of this section.

Generalizing Actions: Data presented through digital interfaces frequently changes. Examples include, the number one song of a top 50 chart in a music player or incoming mails. The visual information available at the time of recording a Rataplan automation can be insufficient and more semantics of the interaction is required to correctly automate the interaction on similar data in the future. In Rataplan, the process to generalize a demonstrated action is initiated by gathering additional information that define user intent.

Rataplan extracts all potentially relevant UI elements near the demonstrated action (Figure 6a). This consists of all elements on top, below, right, and left of the demonstrated action, and includes elements outside the current view port. Elements that have a diagonal relationship (e.g. top-right direction) or are part of another scrollable panel are not considered. When the user selects one or multiple relevant neighbors, Rataplan interprets all possible spatial relationships and suggests various parametric action specifications (Figure 6b).

Figure 6 shows how to use Rataplan to play the first song of “Ed Sheeran” in the top 50 chart, independent of the song title. The user first demonstrates the actions to navigate to the top 50 chart and starts the first song of the artist. Rataplan then extracts and presents GUI elements having a horizontal or vertical spatial relationship with

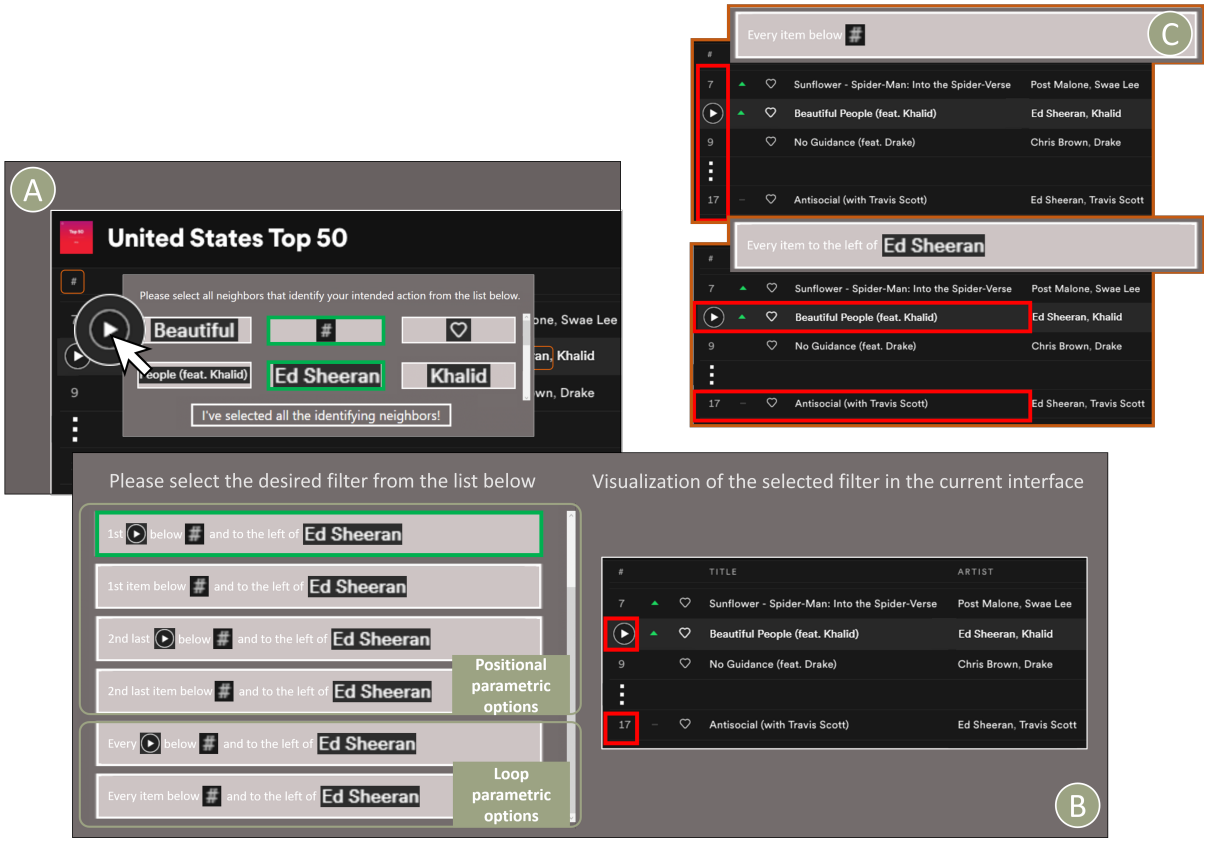

Fig. 6. The process for generalizing a demonstration to play the first song of a specific artist in the top 50 chart. (a) The user selects the relevant nearby UI elements. (b) Rataplan generates a list of possible intentions from which the user can choose and visualizes the effect on the current interface. (c) Selecting only one of the nearby elements would change the generated human-readable possibilities and thus the selection in the current interface. With the real-time interface offered by Rataplan, the user can explore options to accurately define intent.

the play button within the same scrollable panel (Figure 6a). In this case, selecting the header for the play button (the label "#") and the label containing the artist name triggers Rataplan to generate possible human-readable parametric user intents, depending on the selected elements and their spatial relations with respect to the play button, including "the first <play button> below # and to the left of Ed Sheeran". The proposed positions are based on the demonstration and the relation to the identified neighbors (i.e. demonstrating the second song of the same artist would change the generated option to always play the second song). In addition to parametric actions, selecting relevant neighboring GUI elements can also be used to further constrain demonstrated actions. For example, playing only a song by "Ed Sheeran" if it is a collaboration with "Khalid" in this case. Note that Rataplan considers all information in the viewport (automated scrolling) to find all possible parametric actions. While users need to select appropriate neighboring elements to reveal the desired parametric action sequences, Rataplan facilitates this by extracting possible neighboring elements, expressing actions in natural language, and

visualizing the effect of a selection immediately on the current interface. Users can always go back and explore parametric actions in relation to other GUI elements (Figure 6c).

Loop Actions: For every generalized action, Rataplan offers the option to replace the positional attribute of the action (e.g. 5th item in a list) with a variable. As shown in Figure 6, Rataplan generates looping intents in addition to positional parametric intents. For example, by selecting the last option in Figure 6b, every song of the artist in the playlist can be played in sequence. This allows users to define three types of looping features. In each of these loops, the variable is incremented every iteration.

- Repeat immediately: Finishing the action sequence immediately triggers the next iteration. This is applicable for action sequences that only have a momentary effect, such as archiving a mail and checking off items in a "To Do" application. In other words, no wait time is required between iterations.
- Repeat and wait for monitoring action: Rataplan waits for the previous iteration of the loop to finish before starting the next iteration. As our approach is entirely pixel-based, Rataplan is not notified when an action is completed (e.g. a song is finished). Therefore, the user has to specify a change in visual state to trigger the next iteration. This is similar to the monitoring functionality offered by "Help, it looks confusing" [16].
- Repeat and wait for user action: The action sequence is repeated when the user interacts with the trigger, such as pressing a Rataplan push-button or executing a keyboard shortcut. By default, a double press resets the variable.

When a single action sequence consists of multiple loops, the user is requested to specify the order. This is done by selecting the action to repeat first. For example, starting the X-th song of every playlist first before playing the X+1-th song. Alternatively, playlist X can be processed from start to finish before starting playlist X+1. In contrast to Rataplan's human readable questions, earlier UI automation systems that support loops only allow automation of simple loops (i.e. only one variable and no nested loops), and require users to specify positive and negative example of looping objects, together with "supporters", which was reported to be a foreign concept in previous user studies [16].

4.3 Robust Automation Phase

When interacting with Rataplan controls, the user-specified action sequences are executed without requiring additional user intervention. We believe this is crucial to ensure users do not have to go back to the desktop or tablet computer to retrain the system. Rataplan achieves a robust automation phase by performing a thorough graphical analysis of the interface and by contributing a strong contingency plan designed to automate the current generation of user interfaces (Section 3.2). Our contingency plan supports the following four major contingency strategies to successfully automate modern UIs over extended periods of time without user intervention. Unlike current state-of-the-art UI automation techniques [1, 15, 16], these strategies do not require the system to be in the correct state before starting the automation.

(1) **Ambiguity Resolution**: User interfaces often consist of elements that look very similar or even exactly the same. Figure 3a shows an example of a toggle switch that is present twice, each having a different functionality. Duplicate UI elements can already be present during the demonstration phase, or only later during the automation phase as a result of user actions or updates. Handling these ambiguities previously required user input by identifying supporting elements or additional training during the automation [16]. Rataplan automatically resolves ambiguities considering nearby UI elements while analyzing matches during the automation phase. For the example in Figure 3a, menu items on top and below matches are compared to elements present during the demonstration phase. Depending on these results, a penalty is given to both matches to eliminate false-positives as we explain in Section 6.2.

(2) **Hidden Information**: Automated interface elements might not be visible during the automation phase. Rataplan supports several features for finding UI elements.
First, we match text using Optical Character Recognition (OCR) in addition to matching images. OCR is oftentimes more robust for matching UI elements that are only partially visible. Figure 4b shows an example of menu items being partially invisible in the automation phase as a result of changes in the viewport. In these situations, matching images will not result in accurate matches.
Additionally, interface elements that users interact with are sometimes only visible when hovering a region. Figure 4a shows an example in Gmail in which hovering mails reveals the archive button for that mail. Rataplan supports a strategy to automate such hidden actions by automatically hovering nearby UI elements. Finally, a demonstrated action can be outside of the current view port. Rataplan therefore navigates through scroll panels while matching and automating actions.
(3) **Handling Interface Interruptions**: Occasionally, operating systems and applications present users with messages that require action for the system to continue. Examples include, application updates, OS updates, and advertisements. As these interrupts happen occasionally, they are not likely to be present during both the demonstration and automation phase. Rataplan is therefore configured to automatically cancel such messages during the automation phase by finding a close button or buttons that include predefined text, such as “close”, “cancel”, “postpone”, or “later”.
(4) **Application State Management**: In the automation phase, application states can differ from the demonstration and specification phase as a result of user interactivity or previous actions. Rataplan mitigates this concern by always running applications in full-screen mode (taskbar cannot be used to switch applications) and forcing the user to open applications through Rataplan’s home screen. Rataplan can therefore manage these applications and restarts applications when initiating the demonstration phase. Therefore, the user always demonstrates the entire action sequence from the application startup. In the automation phase, Rataplan first tries to execute an automated action sequence from the current state. If not successful, the application restarts. While these features simulate a sandbox mode, Microsoft Windows’ sandbox features could not be used for Rataplan as it starts applications without any user data or preferences.

5 RATAPLAN CONTROLS

Automated action sequences in Rataplan can be triggered locally or over the network. Rataplan’s most basic feature is to trigger automated actions using keyboard shortcuts. However, with more and more mobile and desktop applications becoming available, supporting users in a wide variety of tasks, it is not always convenient to physically move to the computer to trigger an action. Many situations exist in which it is cumbersome, not possible, or not allowed to use mobile or desktop computing devices. Examples include checking a digital manual with dirty hands during cooking or construction work, contamination or distraction risks in laboratories, and impaired users physically not able to operate digital devices [26]. To support these use-cases, Rataplan provides the option to link tangible controls to automated sequences [4, 12]. This allow actions automated with Rataplan to be triggered from various locations as well as to monitor information on graphical interfaces remotely through the use of Rataplan remote displays. Rataplan supports five controls (three input and two output), including push-buttons, a toggle-switch, a rotary potentiometer, a monochrome OLED screen, and a LCD screen (Figure 7a). More tangible controls can be supported in future versions, such as actuated tangibles [2]. Moreover, as we will show in this section, Rataplan can be easily extended to support a wide variety of input methods to trigger actions, such as speech, gesture, and sensor based interactions.

To link a physical input control to a UI element, the user first demonstrates all actions to navigate to the UI control and holds the control while simultaneously touching the UI element. Rataplan recognises the tangible control being manipulated using a capacitive sensor embedded in every Rataplan control (Figure 7c). Linking a

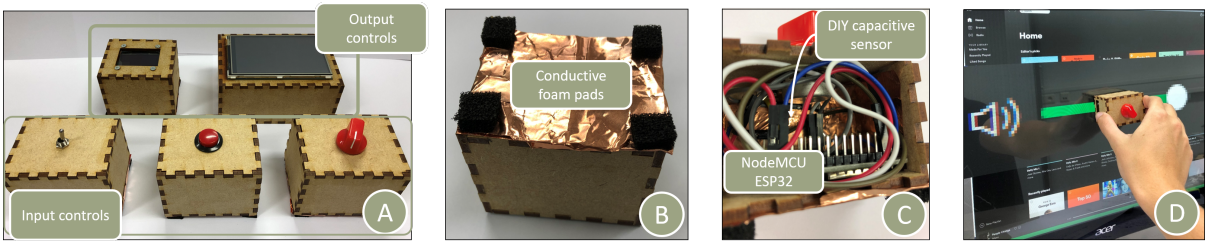

Fig. 7. Rataplan Controls. (a) 3 types for input: a switch, a push-button, and a rotary potentiometer; 2 types for output: a monochome OLED screen and a 3.5" LCD display. (b) Four 1x1 cm conductive foam pads on the bottom of a control, interconnected with copper tape. (c) The inside of a Rataplan control where the microcontroller and a capacitive sensor are visible. (d) The Rataplan control recognised on a capacitive touch screen.

Rataplan control to a UI element of the same type (e.g. Rataplan push-button to UI button control) creates a 1:1 mapping between the physical and the digital widgets. However, Rataplan supports more complex mappings by linking Rataplan controls to a specific state of a widget. For example, linking three Rataplan buttons to a specific position on a slider widget. To allow precise positional specification, Rataplan enlarges the final widget of the demonstrated sequence making it more convenient to specify specific states (Figure 7d).

To monitor UI elements via a Rataplan output control (OLED or LCD screen), users demonstrate the action sequence to navigate to the UI element and link the Rataplan output control to that UI region using a selection. Rataplan will continuously stream that region to the output control. For textual information, the user is presented the option to use OCR and stream only the characters. For monitoring operations, Rataplan launches the target application in a different workspace to ensure users can continue using other applications on their computer. When monitoring multiple parts of an application that are not visible at the same time, Rataplan continuously navigates and thus switches between the UI regions. In those situations, the information on a Rataplan output control is frozen while Rataplan navigates to another screen region.

Users working on a capacitive touch screen are also offered the possibility to link Rataplan controls to GUI widgets by positioning the tangible control on the touch screen (Figure 7d). Rataplan then recognizes the position of the control and automatically links it to the respective position/state of the enlarged UI element below. To sense Rataplan controls on the touch screen, every tangible is equipped with a conductive pattern at the bottom (Figure 7b) [29, 30].

Rataplan controls are convenient to configure as they are immediately recognized when touched and intercommunicate with the Rataplan system. However, as automated action sequences in Rataplan can be triggered over the network, one could also link other commercial devices, such as gestures from the Microsoft Kinect, speech commands from Amazon Alexa, or Amazon Dash buttons to recorded actions in Rataplan.

6 SYSTEM ARCHITECTURE AND IMPLEMENTATION

The current version of Rataplan is developed for Microsoft Windows platforms, but the concepts can be transferred to other operating systems. Rataplan is implemented in C#.NET and uses the Win32 API for tracking and triggering operating system wide mouse events, keystrokes, and for capturing screenshots. We use Emgu CV[2], a .NET wrapper for OpenCV, for computer vision processing.

[2]http://www.emgu.com

Fig. 8. Rataplan's graphic pipeline to extract UI elements. (a) Conversion to gray-scale image. (b) Sobel operator in X and Y direction for edge detection, binary threshold, and morphological open and close operations. (c) The bounding box of every contour is considered a potential target.

6.1 Recognizing UI Elements

Rataplan recognizes UI elements by analyzing screenshots and user interactions during the demonstration and specification phases. In addition, Rataplan further analyzes the UI's dynamic behavior during the specification and automation phase as explained in Section 6.3.

The analysis of a screenshot captured during the demonstration phase starts with recognizing UI elements to retrieve the target control and all surrounding elements. State-of-the-art approaches consider fixed-sized regions around the mouse pointer as target controls [1, 16] which not always accurately enclose the true target. Rataplan takes a more scalable approach by recognizing and extracting targets using computer vision. During this process, a sequence of computer vision filters are applied to the screenshot, including conversion to gray-scale (Figure 8a), a Sobel operator in the X and Y direction for edge detection, binary thresholding, and morphological open and close operations to filter spacing between letters and words belonging together (Figure 8b). Finally, we consider the bounding boxes of every contour as individual UI elements (Figure 8c). Using this approach, Rataplan extracts UI elements users interact with and finds matches in the interface during the specification and automation phase.

6.2 Matching UI Elements

Rataplan matches UI elements during the automation phase using template matching and SIFT, for matching respectively small and large patterns. This approach is similar to the FindAll() implementation in Sikuli [31]. Regions that match more than 70% are considered a match. This threshold is intentionally low to allow for changes in the interface design over time and match controls, such as sliders, when they are in a different state. False positives are filtered by our ambiguity resolution strategy. When no match is found, Rataplan looks for matches using Optical Character Recognition (OCR). Our implementation uses tesseract-ocr[3] for recognizing text in captured screenshots. In contrast to matching images, OCR is more robust for matching UI elements that are cropped as a result of changes in the viewport (Figure 4b).

When no matches are found, Rataplan initiates a strategy to hover nearby UI elements while taking new screenshots. The matching procedure then starts again to find the target element. When no matches are found, we change the viewport of the interface (i.e. we automatically scroll the interface), and repeat the strategy.

When matching target controls during automation, multiple exact matches (i.e. ambiguities) and false positives (due to relative low matching threshold) are extracted from the interface. Therefore, Rataplan always considers the neighboring UI elements of these matches. This process starts with matching the direct neighbors of the

[3] https://github.com/tesseract-ocr/tesseract/wiki

target control. When scrollable panels are present during the demonstration or automation phase, only neighbors located in the same panel as the matching region are considered (Section 6.3). When a neighbor element is not found during the automation phase, or when its spatial relationship changed, a penalty of respectively 75 and 50 points is assigned to the match. For every pixel that a neighbor moved from its original relative position to the target, a penalty of 0.5 applies. Matches for which the penalty exceeds our threshold value of 300 are eliminated. At least five neighbors are considered and more neighbors are processed in the surrounding area until one match remains with an acceptable penalty (below threshold). When all matching target elements are eliminated, we change the viewport of the interface (i.e. we automatically scroll the interface), and repeat the entire strategy.

6.3 Analyzing Dynamic UI Behavior

To automate viewport changes and only consider neighbor elements within a specific scroll view, scrollable panels have to be recognized. This is challenging for pixel-based analysis as current generation user interfaces often only visualize scroll-bars when the user starts scrolling. Rataplan recognizes scrollable panels during the specification and automation phase by analyzing the dynamic behavior of the interface. This is done by triggering scroll-events in different regions of the interface and calculating the visual changes in the captured screenshots before and after this event. Rataplan looks for scrollable panels efficiently by dividing the interface in a quadtree structure and triggers short events that go unnoticed to the user.

During the specification phase, Rataplan recognizes the types of UI controls to correctly automate behavior (Figure 5). Rataplan identifies the type of UI control by analyzing user interactions with this element and corresponding UI changes. For example, a UI element that visually changes after being pressed is considered a toggle switch. Alternatively, when the mouse drags on top of a UI element, it is recognized as a slider. As this technique is not always fool-proof, users can adjust the recognized widget type while answering follow-up questions (Figure 5).

6.4 Observing and Interacting with Pixels in the Background

Rataplan provides the option to deploy action and monitoring sequences on a separate workspace on the Windows operating system. This allows for using the computer while working with Rataplan controls as all automated interactivity takes place in another workspace. We implemented this by taking screenshots of multiple workspaces using Microsoft's Screen Capture API[4] and programatically controlling virtual desktops[5].

6.5 Sensing Rataplan Controls

Every Rataplan control is a stand-alone module containing a battery-powered NodeMCU ESP32 development board[6]. Rataplan controls use the ESP32 WiFi chips embedded in the NodeMCU to communicate to the Rataplan software over TCP/IP sockets. Events are transferred from the Rataplan controls to the Rataplan software in JSON format, and embed a unique identifier for each control.

To sense when a Rataplan control is grasped by the user and thus linked to the recorded action sequence (see section 5), every control is equipped with a capacitive sensor.

To extract the position of a Rataplan control on a capacitive touch screen, we use a technique similar to PUCs [30]. The bottom region of Rataplan controls is equipped with a geometric pattern of conductive pads, interconnected with copper tape (Figure 7b). As the number of patterns, recognisable on a touchscreen is limited [30], we use this technique only for sensing the position of Rataplan controls placed on the screen. To retrieve the ID of the Rataplan control positioned, we use the embedded capacitive sensor in the control and

[4] https://docs.microsoft.com/en-us/windows/uwp/audio-video-camera/screen-capture
[5] https://msdn.microsoft.com/en-us/library/windows/desktop/mt186440%28v%3Dvs.85%29.aspx
[6] https://joy-it.net/en/products/SBC-NodeMCU-ESP32

Table 2. An overview of use cases which have been resiliently automated by Rataplan

Application	**Examples of automated behaviour**
Spotify, Groove Music	Controlling the play/pause button, controlling the volume slider, playing a specific song/playlist, playing all songs from a specific artist in playlist
Skype (standalone and website through Google Chrome)	Calling a specific contact (e.g. as emergency contact)
Windows Photos	Navigating through photos
Windows File explorer & Adobe Acrobat Reader	Opening all files with PDF icon, and printing them through the PDF program
Gmail (through Google Chrome), Windows Mail, Outlook	Sorting mails in folders, archiving mails, monitoring the number of unread mails (output)
Todoist, Microsoft Todo	Checking all todo items in a list or all todo items between boundaries (e.g. all items for today only, leaving others unchecked)
Windows Settings	Turning settings on and off (e.g. WiFi toggle)
Gimp, Inkscape	Changing settings from tools (e.g. brush or line thickness)
Windows Alarms & Clocks	Turning a specific alarm on or off, (re)starting the timer or the stopwatch

assume this is the only control being manipulated at the time (Figure 7c). This limitation can be mitigated in the future by using more advanced sensing approaches [29] in Rataplan.

7 VALIDATION, EXAMPLE DESIGNS, AND USE CASES

Rataplan is designed to automate a wide variety of graphical interfaces on desktop and tablet computers, including web browsers and thereby numerous websites. The possibilities of Rataplan go beyond the examples given in this paper. To validate our approach scales to a wide variety of settings and interfaces, we successfully automated a number of scenarios with Rataplan, as shown in Table 2. To illustrate the potential of Rataplan, we discuss several examples and use cases in more detail below.

- **Helping people with disabilities**: Rataplan can help people with disabilities regain their independence. Examples include controlling advanced desktop interfaces, such as Spotify (Section 2), from around the house using input modalities, such as speech or a physical remote control. In this context, we also experimented with configuring a physical button that triggers calling an emergency contact over Skype.
- **Productivity and professional use**: Rataplan offers many opportunities to increase one's productivity in professional or personal settings. For example, configuring Rataplan with an external trigger to navigate through a photo album in Windows Photos avoids having a computer on the lap when scrolling through vacation photos during a family gathering. In a printing shop, Rataplan can automate the file browser and simplify the procedure to trigger the printing procedure for all files in a folder. While testing this example, we noticed that Rataplan's contingency plan makes our automation sequence even resilient when switching the layout (i.e. list, thumbnail, details mode) of the file explorer. As shown in Figure 9, Rataplan also offers convenience for mail clients, including web based and standalone clients. We successfully used Rataplan to continuously map the number of unread emails in Gmail to a Rataplan display (Figure 9a). Rataplan achieves this by recording and replaying the keystrokes to navigate to the Gmail website. As this is a monitoring task,

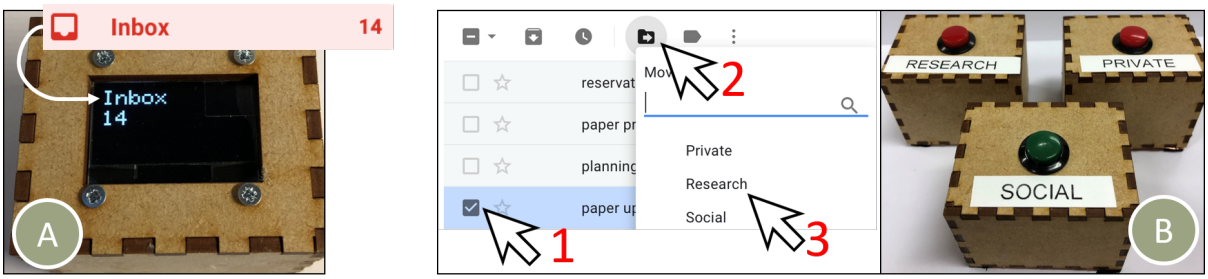

Fig. 9. Using Rataplan to increase productivity in Gmail. (a) a Rataplan screen used to monitor the number of unread mails. (b) Dedicated Rataplan push-buttons to organise mails into folders.

Rataplan opens the web browser in another workspace to continue monitoring the region while the computer is being used for other activities (see Section 6.4). Additionally, we used Rataplan to make an automation sequence for organizing mails in Gmail using physical buttons. Pressing one of the buttons, shown in Figure 9b, assigns the corresponding label to the selected mail.

- **Alternative interaction modalities**: As Rataplan actions can be triggered over the network, similar to the presented Rataplan controls (Section 5), they can also be triggered using custom designed controls, commercially available devices, or online events (e.g. IFTTT[7]). For example, one can build a custom foot control [25] to trigger actions on their desktop computer while carrying a large box. Similarly, a voice command via Amazon Alexa could turn on an alarm clock on a tablet computer while laying in bed.

8 USER EVALUATION

Although Section 3.2 includes a detailed analysis and comparison of features available in Rataplan and state-of-the-art visual techniques for UI automation, this section reports on a user study comparing the usability and utility of features, Rataplan has in common with existing techniques. Our study compares Rataplan to "Help, it looks confusing" (HILC) [16] as this tool is open source and supports basic automation as well as advanced looping actions, also available in Rataplan. As Rataplan includes several features not available in state-of-the-art visual UI automation approaches, such as automatic view port changes (Section 3.2), we did not include those features in the study tasks presented to our participants. The user study was designed to reveal desires for automating GUI tasks, to compare the usability between Rataplan and HILC, and to see which features and aspects of Rataplan are desirable to have.

8.1 Study Procedure and Tasks

Our user study involved 16 participants (15 male, 1 female) who were recruited from computer science labs outside our research unit. None of them were involved in, or aware of this project. Participants had an average age of 28 years (SD 6.8, min 22, max 42). All participants reported a formal computer science education or background, with various degrees of knowledge of software development. Only half of the participants had any experience with automating GUI tasks before the study. All participants spent roughly one hour to complete the entire study. The study used a within-subjects design and consisted of two tasks: a simple linear task and an advanced looping task in both Rataplan and HILC, for a total of four conditions. For every task, both conditions were counterbalanced across participants. After completing a task in one system, a questionnaire was filled in to

[7]https://ifttt.com

gather feedback on their experience with using either Rataplan or HILC. The five-point Likert scale questioned the participant's confidence, the level of feedback from the tool, the understandability, and several task specific questions. After completing a questionnaire, we conducted a short follow-up discussion to get more insights in participants' ratings. After all tasks were completed, a final questionnaire was filled in by the participants to get insight in their demographics, their background in GUI automation, and to compare Rataplan directly with HILC (e.g. how straightforward the system was, which system they prefer for linear and advanced tasks).

At the start of the study, instruction videos were presented, demonstrating features in both Rataplan and HILC. Participants could pause and rewind these videos, and if necessary watch parts of the videos again while executing the tasks. One of those instruction videos demonstrated automating the action sequence in both HILC and Rataplan for turning MS Windows in airplane mode. This requires automating the following menus: Settings →Network →Airplane mode →WiFi Toggle. To get participants familiar with both systems, they first replicated this procedure using both HILC and Rataplan.

Afterwards, the following two automation tasks were given to the participants:

- The **simple linear task** requires participants to automate actions for opening Spotify, navigate to the "Liked songs" menu, and trigger the "Play" button (Figure 10a). When automating this simple sequence in HILC, supporting UI elements are requested twice: for starting Spotify and for pressing the "Play" button (Figure 10b). Although in the latter case, both identified targets are located on the "Play" button and thus valid, HILC frequently requests such supporters as it matches fixed-sized regions in the UI.
- The **advanced looping task** requires automation for opening the "Todoist" application, navigate to the "Next 7 days" window, and mark all items in the category "Today" as completed (Figure 11a). We explicitly instructed participants that all todo items for other days should not be marked as completed by the automation procedure. In both Rataplan and HILC, this is done by identifying the UI elements containing the labels "Today" and "Tomorrow" as boundaries of the automation task (Figure 11b).

Participants were instructed to specify and test a UI automation procedure for both tasks in Rataplan and HILC. During specifications, we continuously encouraged participants to think aloud. This helped us to get insights in how the tools were used and why errors were made. For each task, participants were given 30 minutes and could try multiple times when the specification turned out to be incorrect after testing. This time limit is deliberately high as in HILC, machine learning approaches need to train multiple times for several minutes to compile an automation procedure based on the identified supporting elements, and positive and negative examples. As participants might not be experienced with Spotify and Todoist, we offered them a printout of screenshots, detailing the sequence of interactions that required automation in Spotify and Todoist. To allow for fair comparison with HILC, both Spotify and Todoist were made available in the Windows taskbar to allow for quick access, similar to the list of applications available in Rataplan.

8.2 Study Results

For the linear tasks, all participants were able to correctly specify automation behaviour in Rataplan in a single attempt. In HILC, all participants successfully identified supporting elements for the Spotify taskbar icon (Figure 10b). However, only 7 participants recognized that no supporters are needed for the play button and correctly dismissed the follow-up request for supporting elements. The other 9 participants positioned supporters inside or around the play button (black background of Spotify's interface). While in most cases the specifications still worked, HILC sometimes got confused by these supporting elements. As such, 3 participants needed a second attempt in HILC to complete the linear task. All participants were able to produce a working specification within the time limit.

To compare the responses on the five-point likert-scale questions between both systems, we ran Mann-Whitney's U tests. Participants reported feeling more confident when specifying the automation sequence in

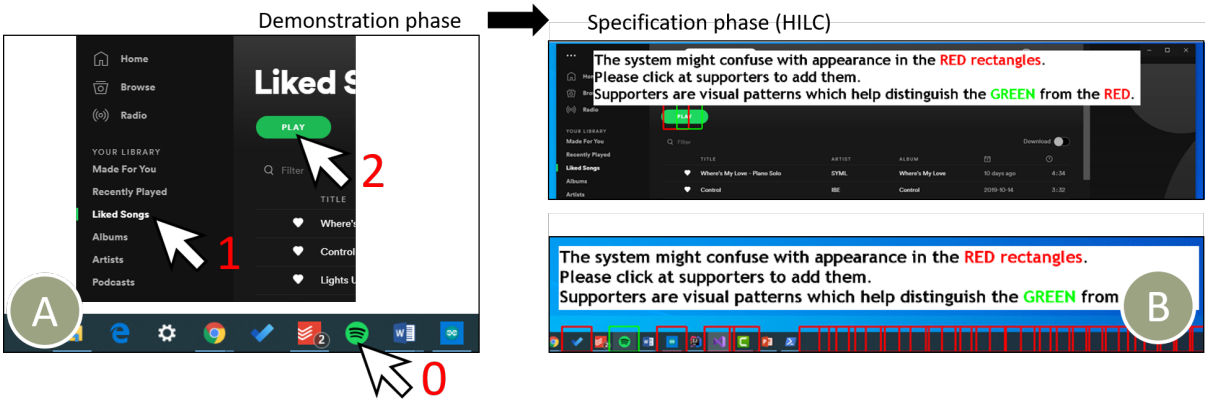

Fig. 10. Linear task presented in the user study. (a) The task the user should demonstrate. (b) Follow-up questions presented by HILC after demonstrating the actions on the left.

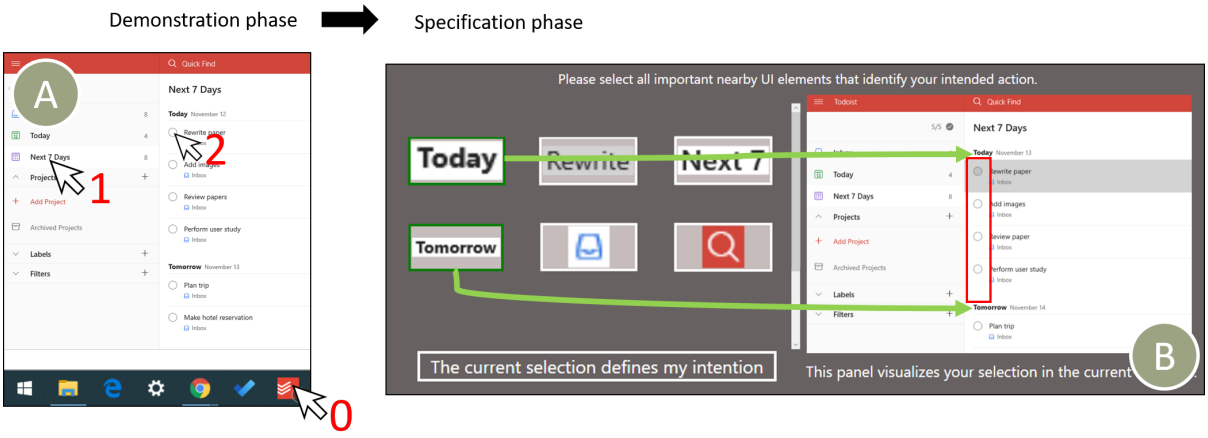

Fig. 11. Loop task presented in the user study. (a) The task the user should demonstrate. While all todo items between today and tomorrow should be checked, the user only needs to demonstrate how to check one item (step 2). (b) Defining user intention in Rataplan. Identifying the labels "Today" and "Tommorow" as important nearby elements visualizes the bounding area in which Rataplan will look for matches in the current interface (cropped and zoomed for clarity).

Rataplan (Mdn=4.0) compared to HILC (Mdn=4.0, Mann-Whitney U=53.0, $p<0.05$). Participants also reported that they received more feedback during specifications in Rataplan (Mdn=5.0) compared to HILC (Mdn=2.5, Mann-Whitney U=21.5, $p<0.05$). Our follow-up discussion revealed that participants felt more guided and understood by Rataplan as follow-up questions are asked in context and participants have a clear visual overview of recorded actions. In contrast, HILC presents feedback and asks follow-up questions through a command-line interface. Therefore feedback is only textual, not in context, and recorded snapshots of interface elements can only be seen when explicitly opening bitmap images. As one participant commented "HILC offers a tool to automate GUIs

but continuous interaction with a command-line interface is required". Participants did not report significant differences between the clarity and understandability of follow-up questions in Rataplan (Mdn=4.0) and HILC (Mdn=3.0). We think this is because the linear task is fairly simple and not many follow-up questions were presented to participants.

For the looping task, for Rataplan we define an attempt as going back to previous screens, and for HILC we define an attempt as retraining the system to get feedback about the current specification. In Rataplan, 13 users successfully specified the action on the first attempt. For the other three participants, one clicked on the label "Today" instead of a todo item during demonstration, one selected 2 todo items during demonstration, and one identified the "Today" label from the left panel instead of the label above the todo items. In the second attempt, these participants were successful as well. For Rataplan, 3 participants quit during or after their third attempt. The other 13 participants required on average 2.2 (SD 1.1) retraining attempts to successfully automate the desired action. Keeping in mind that HILC takes on average 5 minutes of processing time per retraining attempt, specifications took on average longer in HILC than in Rataplan. All participants either made a working specification or quit within the time limit. Participants felt more confident specifying the advanced task in Rataplan (Mdn=4.0) vs. HILC (Mdn=2.5, Mann-Whitney U=39.0, $p<0.05$). Our follow-up discussion revealed that Rataplan really guided participants through the specification. Being able to go back and explore different options without having to wait really helped, as they did not have to think everything through before committing to an action. Rataplan (Mdn=4.0) also provides significantly more feedback compared to HILC (Mdn=2.0) when specifying an advanced task (Mann-Whitney U=26.0, $p<0.05$). Participants appreciated the fact that Rataplan offers real-time feedback when selecting neighbors, which allowed them to reason about which neighboring elements they want to identify. During discussions, some participant mentioned this feature helped them to correctly reason about, and specify, the desired behaviour on the first attempt. In fact, the usefulness of this feature was rated on average 4.7 (SD 0.5) on a five point scale. The follow-up questions asked by Rataplan (Mdn=4.0) are perceived to be more clear and understandable compared to those asked by HILC (Mdn=3.0) for the advanced task (Mann-Whitney U=44.0, $p<0.05$). During the follow-up discussions, the graphical and human-readable questions, and automation queries, offered by Rataplan were identified to be really helpful. In contrast, two participants described the follow-up questions asked by HILC as "cryptic". The usefulness of extracting and visualizing a list of possible important neighbors to choose from (Figure 6a) was ranked on average 4.3 (SD 0.9) out of 5. During discussions, some concerns regarding this feature were expressed, as the overview contained duplicate items (e.g. multiple todo items, and 2 labels containing the text "Today"). Participants suggested visualizing the widget in the interface when hovering it, to allow exploration before selecting one. However, as mentioned by one of the participants, "the overview restricted my choice so I can only select things which are definitely relevant for the tool, so I'm unable to identify wrong things".

In a final questionnaire, participants in our study group reported they found Rataplan (Mdn=4.0) more straightforward to use than HILC (Mdn=2.0, Mann-Whitney U=31.5, $p<0.05$). During follow-up discussion, the guidance of Rataplan throughout the automation process and the real-time feedback for every action were often mentioned. 14 participants would choose Rataplan to automate linear tasks (1 expresses a preference for HILC as they like the efficiency of a command-line interface offered by HILC, and 1 is not interested in automating linear tasks). For advanced tasks (such as looping), 10 participants prefer Rataplan, one prefers HILC and the other 5 are not interested in automating advanced tasks by demonstration. One participant raised an interesting concern about this: "if I would want to automate an advanced task by demonstration, I would either have to do it on real data, or make a test environment where I can specify it. This means I first have to wait until I have my data available before I can make a specification, but maybe at that time I do not feel like doing it, or I have no time to specify the behaviour". On the other hand, other participants appreciated the demonstration aspect, and liked the fact they could just show their intentions to the system. Additionally, multiple participants commented on the time required to specify automation sequences in HILC. Participants mentioned that "this is a long waiting

time especially when you realize afterwards you selected incorrect supporting items" and "if it takes this long to automate a task, it's hard to save time while saving time is what I want to do when automating actions".

9 LIMITATIONS AND FUTURE WORK

Rataplan has several limitations, which open opportunities for future work:

First, additional contingency strategies will further increase the robustness and applicability of our approach. For example, to automate interfaces when a user is not logged in or his computer is locked requires additional research. Simply recording and replaying keystrokes to enter passwords would result in major security flaws. Likewise, interfaces that start in an unknown state or the same state as it was closed, have to be treated differently. For example, users could be asked to demonstrate how to navigate to the home screen in any application. We also noticed that pixel-based GUI automation is challenging when the appearance of a UI element is almost entirely defined by user content. This includes, for example, recognizing a new message in a chat applications like Skype in which the text balloon resizes to fit the message. One powerful feature of Rataplan is the automated scrolling to find content outside the current view port. However, timeouts are required for infinite scrolling lists, often present on social media overview pages. Our algorithms could be extended to support all these cases. However, it would also be interesting to involve remote workers, such as crowd workers, trusted workers, or care givers to occasionally disambiguate or confirm actions during the automation phase. For example, an unknown popup that cannot be canceled or ignored could be forwarded to a trusted worker [13].

Second, Rataplan's GUI automation can be extended with additional features, such as automation of drag-and-drop operations. As the visual state of targets often change during drag-and-drop operations, Rataplan needs to extract a clean image of the target before dropping. This is not trivial, as the target is already occluded when the mouse-release event occurs. Additionally, more advanced automation sequences can be supported in the future by presenting more advanced parametric actions to users. Examples include parametric actions that consider the lexicographic order of lists, such as all items starting with the letter X or all files ending in ".pdf". It can also be desirable to support disjunctions (OR-relations) besides conjunctions (AND-relations) in future versions of Rataplan. This would, for example, allow the same action sequence to run over all ".pdf" and ".png" files, or to play all the songs from two different artists in the same list.

Finally, one of the keys to our resilient automation approach is that applications are started and thus managed by Rataplan. The current version of Rataplan therefore does not support features to UI operations for which no application has to launch. Examples include, OS features that have shortcuts in the taskbar or on the desktop, such as controlling the volume of the speakers. Until the time that we implement a separate strategy for these features, OS actions can be automated with Rataplan by navigating to the respective OS applications that embed these settings. For example, controlling the volume of the speaker through the "Setting →System →Sound" screen.

10 CONCLUSION

As more everyday tasks will continue to be augmented or replaced by digital counterparts, we expect people to have an increasing desire to make alternative input methods to GUI actions. To enable novice end-users to be successful, Rataplan lowers the barier and allows the creation of custom input methods with minimal efforts. The contingency strategies presented in Rataplan allow for resilient automation of dynamic user interfaces, which change over time or during usage, due to variations in the viewport, changing application state, and changing application data. Additionally, Rataplan's parametric and loop features allow non-programmers to automate advanced action sequences that traditionally require complex scripts. While Rataplan empowers new groups of users to program alternative input controls, we hope our insights contribute new knowledge to the broader field of GUI automation.

ACKNOWLEDGMENTS

This research was supported by the Special Research Fund (BOF) of Hasselt University and by the Research Foundation - Flanders (FWO), project G0E7317N End-User Development of Intelligible Internet-of-Things Objects and Applications.

REFERENCES

[1] Khalid Alharbi and Tom Yeh. 2019. Sikuli Slides. http://slides.sikuli.org. Accessed: 2019-07-30.

[2] Mark S. Baldwin, Gillian R. Hayes, Oliver L. Haimson, Jennifer Mankoff, and Scott E. Hudson. 2017. The Tangible Desktop: A Multimodal Approach to Nonvisual Computing. *ACM Trans. Access. Comput.* 10, 3, Article 9 (Aug. 2017), 28 pages. https://doi.org/10.1145/3075222

[3] Shaon Barman, Sarah Chasins, Rastislav Bodik, and Sumit Gulwani. 2016. Ringer: Web Automation by Demonstration. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016)*. ACM, New York, NY, USA, 748–764. https://doi.org/10.1145/2983990.2984020

[4] Florian Block, Michael Haller, Hans Gellersen, Carl Gutwin, and Mark Billinghurst. 2008. VoodooSketch: Extending Interactive Surfaces with Adaptable Interface Palettes. In *Proceedings of the 2Nd International Conference on Tangible and Embedded Interaction (TEI '08)*. ACM, New York, NY, USA, 55–58. https://doi.org/10.1145/1347390.1347404

[5] Ivan Burmistrov, Tatiana Zlokazova, Anna Izmalkova, and Anna Leonova. 2015. Flat Design vs Traditional Design: Comparative Experimental Study. In *Human-Computer Interaction – INTERACT 2015*, Julio Abascal, Simone Barbosa, Mirko Fetter, Tom Gross, Philippe Palanque, and Marco Winckler (Eds.). Springer International Publishing, Cham, 106–114.

[6] Tsung-Hsiang Chang, Tom Yeh, and Rob Miller. 2011. Associating the Visual Representation of User Interfaces with Their Internal Structures and Metadata. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 245–256. https://doi.org/10.1145/2047196.2047228

[7] Morgan Dixon and James Fogarty. 2010. Prefab: Implementing Advanced Behaviors Using Pixel-based Reverse Engineering of Interface Structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 1525–1534. https://doi.org/10.1145/1753326.1753554

[8] Morgan Dixon, Daniel Leventhal, and James Fogarty. 2011. Content and Hierarchy in Pixel-based Methods for Reverse Engineering Interface Structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 969–978. https://doi.org/10.1145/1978942.1979086

[9] Morgan Dixon, Alexander Nied, and James Fogarty. 2014. Prefab Layers and Prefab Annotations: Extensible Pixel-based Interpretation of Graphical Interfaces. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 221–230. https://doi.org/10.1145/2642918.2647412

[10] James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2011. Cracking the Cocoa Nut: User Interface Programming at Runtime. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 225–234. https://doi.org/10.1145/2047196.2047226

[11] W. Keith Edwards, Scott E. Hudson, Joshua Marinacci, Roy Rodenstein, Thomas Rodriguez, and Ian Smith. 1997. Systematic Output Modification in a 2D User Interface Toolkit. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology (UIST '97)*. ACM, New York, NY, USA, 151–158. https://doi.org/10.1145/263407.263537

[12] Saul Greenberg and Michael Boyle. 2002. Customizable Physical Interfaces for Interacting with Conventional Applications. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST '02)*. ACM, New York, NY, USA, 31–40. https://doi.org/10.1145/571985.571991

[13] Anhong Guo, Xiang âĂIJAnthonyâĂİ Chen, Haoran Qi, Samuel White, Suman Ghosh, Chieko Asakawa, and Jeffrey P. Bigham. 2016. VizLens: A Robust and Interactive Screen Reader for Interfaces in the Real World. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST âĂŹ16)*. Association for Computing Machinery, New York, NY, USA, 651âĂŞ664. https://doi.org/10.1145/2984511.2984518

[14] Amy Hurst, Scott E. Hudson, and Jennifer Mankoff. 2010. Automatically Identifying Targets Users Interact with During Real World Tasks. In *Proceedings of the 15th International Conference on Intelligent User Interfaces (IUI '10)*. ACM, New York, NY, USA, 11–20. https://doi.org/10.1145/1719970.1719973

[15] Thanapong Intharah, Michael Firman, and Gabriel J. Brostow. 2018. RecurBot: Learn to Auto-complete GUI Tasks From Human Demonstrations. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems (CHI EA '18)*. ACM, New York, NY, USA, Article LBW595, 6 pages. https://doi.org/10.1145/3170427.3188532

[16] Thanapong Intharah, Daniyar Turmukhambetov, and Gabriel J. Brostow. 2017. Help, It Looks Confusing: GUI Task Automation Through Demonstration and Follow-up Questions. In *Proceedings of the 22Nd International Conference on Intelligent User Interfaces (IUI '17)*. ACM, New York, NY, USA, 233–243. https://doi.org/10.1145/3025171.3025176

[17] JitBit. 2019. Macro Recorder, Macro Program, Keyboard Macros & Mouse Macros. https://www.jitbit.com/macro-recorder/. Accessed: 2019-07-01.

[18] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: Automating & Sharing How-to Knowledge in the Enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 1719–1728. https://doi.org/10.1145/1357054.1357323

[19] Ian Li, Jeffrey Nichols, Tessa Lau, Clemens Drews, and Allen Cypher. 2010. Here's What I Did: Sharing and Reusing Web Activity with ActionShot. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 723–732. https://doi.org/10.1145/1753326.1753432

[20] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 6038–6049. https://doi.org/10.1145/3025453.3025483

[21] AutoHotkey Foundation LLC. 2020. AutoHotKey. https://www.autohotkey.com. Accessed: 2020-02-13.

[22] Xiaojun Meng, Shengdong Zhao, Yongfeng Huang, Zhongyuan Zhang, James Eagan, and Ramanathan Subramanian. 2014. WADE: Simplified GUI Add-on Development for Third-party Software. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2221–2230. https://doi.org/10.1145/2556288.2557349

[23] Dan R. Olsen, Jr., Scott E. Hudson, Thom Verratti, Jeremy M. Heiner, and Matt Phelps. 1999. Implementing Interface Attachments Based on Surface Representations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. ACM, New York, NY, USA, 191–198. https://doi.org/10.1145/302979.303038

[24] Macro Recorder. 2019. Mouse and Keyboard Capture. https://www.macrorecorder.com. Accessed: 2019-07-01.

[25] Dominik Schmidt, Raf Ramakers, Esben W. Pedersen, Johannes Jasper, Sven Köhler, Aileen Pohl, Hannes Rantzsch, Andreas Rau, Patrick Schmidt, Christoph Sterz, Yanina Yurchenko, and Patrick Baudisch. 2014. Kickables: Tangibles for Feet. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 3143–3152. https://doi.org/10.1145/2556288.2557016

[26] P. Frazer Seymour, Justin Matejka, Geoff Foulds, Ihor Petelycky, and Fraser Anderson. 2017. AMI: An Adaptable Music Interface to Support the Varying Needs of People with Dementia. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '17)*. ACM, New York, NY, USA, 150–154. https://doi.org/10.1145/3132525.3132557

[27] Konstantinos Spiliotopoulos, Maria Rigou, and Spiros Sirmakessis. 2018. A Comparative Study of Skeuomorphic and Flat Design from a UX Perspective. *Multimodal Technologies and Interaction* 2, 2 (June 2018), 31. https://doi.org/10.3390/mti2020031

[28] Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips, and Nicolas Roussel. 2006. User Interface FaçAdes: Towards Fully Adaptable User Interfaces. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06)*. ACM, New York, NY, USA, 309–318. https://doi.org/10.1145/1166253.1166301

[29] Simon Voelker, Christian Cherek, Jan Thar, Thorsten Karrer, Christian Thoresen, Kjell Ivar Overgard, and Jan Borchers. 2015. PERCs: Persistently Trackable Tangibles on Capacitive Multi-Touch Displays. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 351–356. https://doi.org/10.1145/2807442.2807466

[30] Simon Voelker, Kosuke Nakajima, Christian Thoresen, Yuichi Itoh, Kjell Ivar Overgard, and Jan Borchers. 2013. PUCs: Detecting Transparent, Passive Untouched Capacitive Widgets on Unmodified Multi-touch Displays. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces (ITS '13)*. ACM, New York, NY, USA, 101–104. https://doi.org/10.1145/2512349.2512791

[31] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: Using GUI Screenshots for Search and Automation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology (UIST '09)*. ACM, New York, NY, USA, 183–192. https://doi.org/10.1145/1622176.1622213