Supporting Co-adaptive Human-Agent Relationship through Programming by Demonstration using Existing GUIs

Toby Jia-Jun Li
HCI Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
tobyli@cs.cmu.edu

Tom M. Mitchell
Machine Learning Dept.
Carnegie Mellon University
Pittsburgh, PA 15213, USA
tom.mitchell@cs.cmu.edu

Igor Labutov
Machine Learning Dept.
Carnegie Mellon University
Pittsburgh, PA 15213, USA
ilabutov@cs.cmu.edu

Brad A. Myers
HCI Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
bam@cs.cmu.edu

Xiaohan Nancy Li
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213, USA
nancylxh14@gmail.com

Rethinking Interaction CHI 2018 Workshop
April 22, 2018, Montreal, QC, Canada.

Abstract

Intelligent agents have become an important new type of interface in the post-WIMP era. However, end users lack the capability to customize, appropriate and extend current agents. In this position paper, we describe our programming by demonstration (PBD) approach, which leverages the end users' familiarity with existing apps' GUIs by allowing them to demonstrate the desired new behaviors of the agent using GUI objects in existing apps. We also outline challenges in providing the users with more expressive power and greater flexibility for this approach, and propose a solution of using a multi-modal interface that combines verbal instructions with the demonstrations using the GUI.

Author Keywords

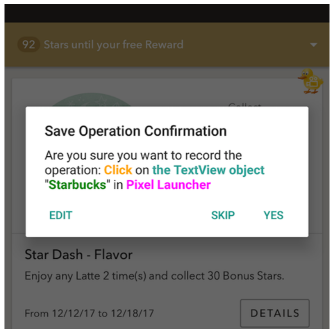
Programming by demonstration; end user development; multi-modal interface.

ACM Classification Keywords

H.5.2. User interfaces (Interaction styles).

Introduction

Intelligent agents have rapidly gained popularity in recent years – they can be found in devices from wearables and phones to smart home speakers and cars. They usually interact with users through a conversa-


92 Stars until your free Reward

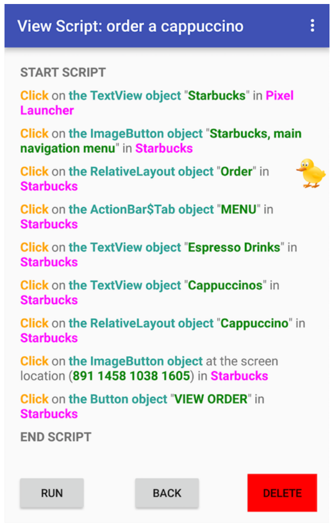
Save Operation Confirmation

Are you sure you want to record the operation: Click on the TextView object "Starbucks" in Pixel Launcher

Star Dash - Flavor

Enjoy any Latte 2 time(s) and collect 30 Bonus Stars.

From 12/12/17 to 12/18/17


View Script: order a cappuccino

START SCRIPT

Click on the TextView object "Starbucks" in Pixel Launcher

Click on the ImageButton object "Starbucks, main navigation menu" in Starbucks

Click on the RelativeLayout object "Order" in Starbucks

Click on the ActionBar$Tab object "MENU" in Starbucks

Click on the TextView object "Espresso Drinks" in Starbucks

Click on the TextView object "Cappuccinos" in Starbucks

Click on the RelativeLayout object "Cappuccino" in Starbucks

Click on the ImageButton object at the screen location (891 1458 1038 1605) in Starbucks

Click on the Button object "VIEW ORDER" in Starbucks

END SCRIPT

Figure 1: SUGILITE's recording confirmation pops up when the user demonstrates using the GUI of the Starbucks app. (top)

SUGILITE's interface for viewing a script. (bottom)

tional user interface (CUI) and can act on behalf of users to perform computing tasks and to interact with external services. Compared with GUIs, CUIs are often more efficient (especially for tasks that require interacting with multi-level menus or multiple apps), and are usable in contexts where direct manipulation is not convenient or possible (e.g., while driving).

However, current agents provide little support for users to customize their behaviors and to extend their capabilities. Many agents support some degree of personalization in a few popular domains (e.g., music, news) by learning the user's past behaviors and predicting the user's preferences. Besides this, users have no way to teach the agent a new task, and have little control on how a task should be performed by the agent when they perceive a need for customization and automation such as in the case of cognitive overload [12].

In our motivating study [9], we observed that since the users lack the capability to *appropriate* the agents, they often just adapt to the agents' constraints by learning what tasks work reliably well, figuring out effective utterances for triggering these tasks, and limiting their use of agents to only this small set of tasks (e.g., checking weather, setting alarms), instead of having a co-adaptive relationship with the agent.

We envision intelligent agents playing an increasingly important role as the interfaces for computing tasks and services in the post-WIMP era. Therefore, our work seeks to empower end users with no programming expertise to customize how tasks should be performed by agents, and to teach agents new tasks.

Our system uses a programming by demonstration approach that allows end user to create task automations by demonstrating using GUIs of existing mobile apps. This approach leverages the available app GUIs and more importantly, the *user knowledge* about how to manipulate these GUI objects to accomplish tasks.

Instead of being invoked directly as interaction instruments, GUI objects can form a vocabulary through which users can communicate their procedural knowledge about tasks, so the objects become building blocks for constructing a meta-instrument (i.e., the agent) that mediates the interactions between the user and the GUI of the app by automating UI actions to perform a task based on the user's verbal input through the CUI.

SUGILITE

We have already designed and implemented a prototype system named SUGILITE [9] (Figure 1) that empowers users to program a virtual assistant by demonstration on GUIs of existing third-party Android mobile apps. Through the EPIDOSITE [10] extension, SUGILITE also supports programming for smart home devices via their corresponding apps, and supports triggering task automation by notifications, app launches and events from external web services in addition to commands from the virtual assistant. In our study [9], SUGILITE was shown to be an efficient way to automate repetitive tasks so that they can be invoked using a voice command rather than directly manipulating the GUI.

SUGILITE can generalize the recorded UI actions by identifying parameters in the task, and then associating parameters specified in the utterance with their corresponding UI actions. For example, for a task with the triggering utterance "order a venti cappuccino", SUGILITE finds two parameters (venti and cappuccino), and matches them to the actions of choosing the cup size

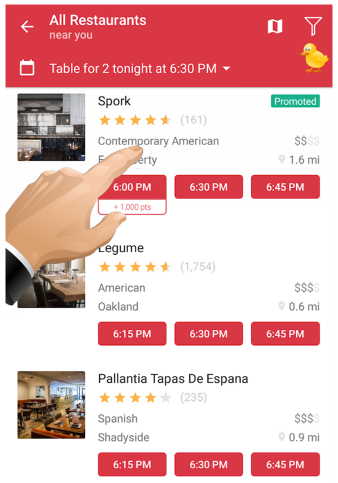


Figure 2: An example of ambiguous demonstration. This demonstrated action can represent many possible commands. Such as "choose 'Spork'", "choose the first entry", "choose the cheapest restaurant from the list", "choose the promoted restaurant" or "choose the restaurant with bonus points available".

Additional information is needed to record the true rationale of the user so that the agent can perform the desired action when encountering a different list later.

from a dropdown menu, and choosing a product from the available coffee types. It also scrapes the alternative possible values for each parameter from the GUI. This generalization mechanism allows SUGILITE to learn not only the exact demonstrated task, but also similar tasks with different variations of parameter values.

Demonstrational agents like SUGILITE are also useful for breaking down the "information silos" between apps by supporting interoperability in the demonstration. Users can first demonstrate locating the information of interest by navigating through the GUI of one app, selecting the GUI object containing the information using a SUGILITE gesture, and then using the scraped information later. Since information is extracted from the presentation layer, it does not require the availability of APIs in the involved apps. Our approach enables end users with no programming skills to automate their personalized cross-app tasks, since they do not have to learn any programming language, or how to use any APIs.

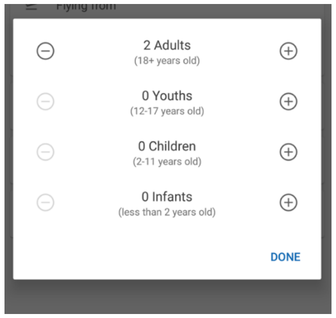
Beyond Record and Replay

Although SUGILITE has supported some generalizations of the demonstrated actions, its current underlying workflow is still largely record-and-replay, where the user first demonstrates an instance of performing the task, SUGILITE then infers parameters in the task, and tweaks the actions accordingly when replaying if the parameter values have changed. Our current research focuses on providing the users with more expressive power and greater flexibility when creating the automation, enabling them to easily construct control structures such as conditionals, iterations and triggers, to embed reasoning and computations to handle situations that are different than when demonstrated, to forage reusable components in existing demonstrations, and to edit existing automations at runtime.

However, many challenges remain. A major one is the lack of information about the user's *rationale* behind actions. In other words, the system needs to infer *why* the user did something from *what* the user did. So, when the scenario changes, the agent can still perform the correct action that matches the user's intention. Figure 2 shows an ambiguous demonstration where the user's rationale cannot be inferred from the action alone.

Understanding user rationale, in many cases, requires the system to know about the semantics of GUI objects. It needs to understand the meaning of the information that each UI element communicates, and the command that will be performed when each interactive GUI object is operated on. While some great progress has been made in this area (e.g., [4–8]), we are still far from having specific and accurate semantic representations for GUI objects that are agnostic to the low-level platform and app-specific details. The semantics of a GUI object is also often not only represented by the object alone, but also in the layout of the GUI and the object's relationship with other objects (see Figure 3 for examples). We are developing an architecture for extracting some semantic information, but future work involves extracting more semantic information from the GUI, or providing it as meta-information with the app.

Furthermore, designing the interactions for the demonstrational interface itself is challenging. To provide users adequate expressive power when they program with GUI objects in existing app interfaces, this "meta-interface" needs to support at least the following types of interactions: (1) *invoke* an object in the app as a part of the demonstration; (2) *extract* the content or a property from an object, (3) *select* an object to use it



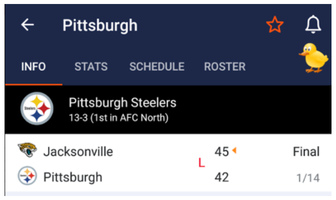


Figure 3: Two examples where the semantics of a GUI object is inferred from its relationship with other objects.

Top: The plus sign is an instrument for incrementing the number of people next to it. Demonstration of clicking on it can be inferred as commands such as "click on the plus sign next to the number of adults until the number of adults reaches 2".

Bottom: SUGILITE needs to understand that the numbers next to the team names represent their scores, so that the user can express implicit reasoning, such as "choose the winning team".

as the target or as a parameter for an operation, etc. Additional commands are also needed for specifying programming logic in automation scripts.

As outlined in the instrumental interaction model [2], due to the mismatch between the small vocabulary of actions and the large vocabulary of commands, additional interface elements are required to specify commands. Those elements often play the role of instruments that can mediate interaction between a user and a domain object. However, adding these elements is difficult on touch-based phones, as techniques like modifier keys or large tool palettes are not feasible due to the limitations of the input devices and screen sizes. It is challenging to design the interaction for the user to specify what command to perform on a GUI object.

These added interactions for controlling the demonstration can also potentially be confusing since most of the screen space is taken up with the GUI of the underlying app. Consequently, it is difficult for users to distinguish between modes of using an app directly and programming by demonstration on the same app, especially if the user needs to refer to a GUI element without invoking any action (e.g., to copy the value of the high score in Figure 3 to use later).

Verbal Instruction + Demonstration on GUIs

To address some of the above challenges, we are investigating the use of multi-modal interfaces, where verbal instructions are supported *at the same time* as direct manipulations on the GUI. This has long been proposed as a more natural way of interaction that provides users with greater expressive power dating back to early pioneer systems like Put-that-there [3] and DreamSpace [11]. Recent systems such as PLOW [1] have also demonstrated the effectiveness of verbal instructions in assisting the agent to learn new tasks from demonstrations by providing extra semantic information for demonstrated actions.

The "speak-and-point" pattern [13] in multi-modal interaction can naturally separate choosing an interaction instrument (by verbally talking about the command to perform) from specifying the target for the interaction (by pointing at the target). As we can often observe from human-human interaction, one would often use this pattern (e.g., "move *this* chair *there*", "paint *this* fence in *that* color") when giving instructions. We also hope that supporting verbal instructions in our PBD system can enable more natural programming by allowing users to express their intentions intuitively in a way that closely matches their conceptual model of the task.

Based on results from our preliminary Wizard-of-Oz studies, we make two hypotheses on how users would perform when asked to provide verbal instructions simultaneously while demonstrating: (1) They would refer to GUI objects available on the screen in verbal instructions (e.g., "if the number *here* is greater than the one *next to it, …*"); (2) They would naturally use the feature that reflects their rationale for referring to GUI objects (e.g., for the action in Figure 2, they would say "choose the first one / "Spork" (the text) / the cheapest one" depending on their intentions). If both hypotheses turn out to hold, the instructions will be very useful in inferring user rationale for the ambiguous demonstration problem illustrated in Figure 2. On the other hand, the information about the GUI can also be used for grounding the verbal instructions, helping to improve the performance of speech recognition and semantic parsing.

Acknowledgement

This work was supported by Oath/Yahoo! through CMU's InMind project.

References

1. James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. 2007. PLOW: A Collaborative Task Learning Agent. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2* (AAAI'07), 1514–1519.
2. Michel Beaudouin-Lafon. 2000. Instrumental Interaction: An Interaction Model for Designing post-WIMP User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '00), 446–453. https://doi.org/10.1145/332040.332473
3. Richard A. Bolt. 1980. "Put-that-there": Voice and Gesture at the Graphics Interface. In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques* (SIGGRAPH '80), 262–270. https://doi.org/10.1145/800250.807503
4. Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST '17), 845–854. https://doi.org/10.1145/3126594.3126651
5. Biplab Deka, Zifeng Huang, and Ranjitha Kumar. 2016. ERICA: Interaction Mining Mobile Apps. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16), 767–776. https://doi.org/10.1145/2984511.2984581
6. 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), 1525–1534. https://doi.org/10.1145/1753326.1753554
7. 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), 225–234. https://doi.org/10.1145/2047196.2047226
8. Björn Hartmann, Leslie Wu, Kevin Collins, and Scott R. Klemmer. 2007. Programming by a Sample: Rapidly Creating Web Applications with D.Mix. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology* (UIST '07), 241–250.
9. 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), 6038–6049. https://doi.org/10.1145/3025453.3025483
10. Toby Jia-Jun Li, Yuanchun Li, Fanglin Chen, and Brad A. Myers. 2017. Programming IoT Devices by Demonstration Using Mobile Apps. In *End-User Development* (Lecture Notes in Computer Science), 3–17. https://doi.org/10.1007/978-3-319-58735-6_1
11. Mark Lucente, Gert-Jan Zwart, and Andrew D. George. 1998. Visualization space: A testbed for deviceless multimodal user interface. In *Intelligent Environments Symposium*.
12. Wendy E. Mackay. 2000. Responding to cognitive overload: Co-adaptation between users and technology. *Intellectica* 30, 1: 177–193.
13. Sharon Oviatt. 1999. Ten Myths of Multimodal Interaction. *Commun. ACM* 42, 11: 74–81. https://doi.org/10.1145/319382.319398