Programming IoT Devices by Demonstration Using Mobile Apps

Toby Jia-Jun Li[1(✉)], Yuanchun Li[2], Fanglin Chen[1], and Brad A. Myers[1(✉)]

[1] Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, USA
{tobyli,bam}@cs.cmu.edu, fanglin@cmu.edu
[2] School of Electronics Engineering and Computer Science, Peking University, Beijing, China
yuanchun.li@pku.edu.cn

Abstract. The revolutionary advances of Internet of Things (IoT) devices and applications have helped IoT emerge as an increasingly important domain for end-user development (EUD). Past research has shown that end users desire to create various customized automations, which would often utilize multiple IoT devices. Many solutions exist to support EUD across multiple IoT devices, but they are limited to devices from the same manufacturer, within the same "eco-system" or supporting a common API. We present EPIDOSITE, a mobile programming-by-demonstration system that addresses this limitation by leveraging the smartphone as a hub for IoT automation. It enables the creation of automations for most consumer IoT devices on smartphones by demonstrating the desired behaviors through directly manipulating the corresponding smartphone app for each IoT device. EPIDOSITE also supports using the smartphone app usage context and external web services as triggers and data for automations, enabling the creation of highly context-aware IoT applications.

Keywords: Internet of Things · Programming by demonstration · End user development

1 Introduction

In the recent years, the rapid growth of Internet of Things (IoT) has surrounded users with various smart appliances, sensors and devices. Through their connections, these smart objects can understand and react to their environment, enabling novel computing applications [39]. A past study has shown that users have highly diverse and personalized desired behaviors for their smart home automation, and, as a result, they need end-user tools to enable them to program their environment [42]. Especially with the growing number of devices, the complexity of the systems, and their importance in everyday life, it is increasingly important to enable end users to create the programs themselves for those devices to achieve the desired user experience [33, 40].

Many manufacturers of smart devices have provided their customers with tools for creating their own automations. For example, LG has the SmartThinQ[1] app, where the

[1] https://us.smartthinq.com/.

© Springer International Publishing AG 2017
S. Barbosa et al. (Eds.): IS-EUD 2017, LNCS 10303, pp. 3–17, 2017.
DOI: 10.1007/978-3-319-58735-6_1

bam@cs.cmu.edu

user can author schedules and rules for their supported LG smart appliances such as fridges, washers and ovens. Similar software is also provided by companies like Samsung (SmartThings), Home Depot (Wink) and WeMo. However, a major limitation for all of these is the lack of interoperability and compatibility with devices from other manufacturers. They all only support the limited set of devices manufactured by their own companies or their partners. Therefore, users are restricted to creating automations using devices within the same "ecosystem" and are unable to, for instance, create an automation to adjust the setting of an LG air conditioner based on the reading from a Samsung sensor.

Some platforms partially address this problem. For example, IFTTT (ifttt.com) is a popular service that enables end users to program in the form of "if *trigger*, then *action*" for hundreds of popular web services, apps, and IoT devices. With the help of IFTTT, the user can create automations across supported devices like a GE dishwasher, a WeMo coffeemaker and a Fitbit fitness tracker. However, the applicability of IFTTT is still limited to devices and services offered by its partners, or those that provide open APIs which can connect to IFTTT. Even for the supported devices and services, often only a subset of the most commonly used functions is made available due to the required engineering effort. Other platforms like Apple HomeKit and Google Home also suffer from the same limitations. Because of the lack of a standard interface or a standard protocol, many existing tools and systems cannot support the heterogeneity of IoT devices [17, 21]. While generalized architectures for programming IoT devices and higher-level representations of IoT automation rules and scripts have been proposed (e.g., [15, 16, 20, 21, 37]), they have not yet been widely adopted in the most popular commercial IoT products, and there is some reason for pessimism that such an agreement will ever happen.

To solve these problems, we create a new system named EPIDOSITE,[2] which is an end-user development (EUD) tool that enables the creation of automations for IoT devices from different ecosystems by demonstration though manipulating their corresponding mobile apps. Our system particularly targets the development of automation scripts for consumer IoT devices in the smart home environment by end users with little or no programming expertise. Thanks to the ubiquity of smartphones, for the majority of consumer IoT devices, there are corresponding smartphone apps available for remote controlling them, and these apps often have access to the full capabilities of the devices. A smartphone loaded with these apps is an ideal universal interface for monitoring and controlling the smart home environment [46]. Thus, by leveraging the smartphone as a hub, we can both read the status of the IoT sensors by scraping information from the user interfaces of their apps, and control the IoT actuators by manipulating their apps. To our knowledge, ours is the first EUD system for IoT devices using this approach.

[2] EPIDOSITE is a type of rock. Here, the name stands for "**E**nabling **P**rogramming of **I**oT **D**evices **O**n **S**martphone **I**nterfaces for **T**he **E**nd-users".

bam@cs.cmu.edu

1.1 Advantages

Our approach has the following three major advantages:

Compatibility: Unlike other EUD tools for consumer IoT devices, which can only support programming devices from the same company, within the same ecosystem, or which provide open access to their APIs, EPIDOSITE can support programming for most of the available consumer IoT devices if they provide Android apps for remote control. For selected phones with IR blasters (e.g., LG G5, Samsung Galaxy S6, HTC One) and the corresponding IR remote apps, EPIDOSITE can even control "non-smart" appliances such as TVs, DVRs, home theaters and air conditioners that support IR remote control (but this obviously only works when the phone is aimed at the device).

Interoperability: EPIDOSITE can support creating automation across *multiple* IoT devices, even if they are from different manufacturers. Besides IoT devices, EPIDOSITE can also support the incorporation of arbitrary third-party mobile apps and hundreds of external web services into the scripts. The exchange of information between devices is also supported by demonstration. The user can extract values of the readings and status of IoT devices using gestures on one mobile app, and use the values later as input for other apps. This approach addresses the challenge of supporting *impromptu interoperability* [14], a vision that devices acquired at different times, from different vendors and created under different design constraints and considerations should interconnect with no advance planning or implementation.

Usability: We believe that EPIDOSITE should be easy to use, even for end users with little or no prior programming experience. Since the users are already familiar with how to use the mobile apps to monitor and to control IoT devices, EPIDOSITE minimizes the learning barrier by enabling users to use those familiar app interfaces to develop automations by demonstrating the procedures to control the IoT devices to perform the desired behaviors. A major advantage of PBD is that it can empower users while minimizing learning of programming concepts [30, 35]. The evaluation of the SUGILITE system [29], which shares the same demonstrational interface as EPIDOSITE, also suggests that most end users can successfully automate their tasks by demonstration using mobile apps on smartphones.

2 Related Work

Supporting end user development activities in the Internet of Things era has been identified as a particularly important and promising research direction [3]. Due to the dynamic nature of context, it is difficult or even impossible for a developer or a designer to enumerate all of the possible contextual states and the appropriate action to take from each state [19]. To solve this problem, the end users should play a central role in deciding how elements in the IoT ecosystem should interact with people and with other devices, and the end-users should be empowered to program the system behavior [3, 44]. The longitudinal study by Coutaz and Crowley [7] reported that families liked EUD for IoT

bam@cs.cmu.edu

devices in the smart home environment because it provides convenience and personalization. Much prior work has focused on enabling end users to configure, extend and customize the behaviors of their IoT devices with various approaches. However, we believe EPIDOSITE is the first to enable the users to create automations for IoT devices by demonstration using the devices' corresponding mobile apps.

In the specific area of EUD for developing home automation scripts across IoT devices, rule-based systems like HomeRules [10], IFTTT (ifttt.com) and Altooma (www.atooma.com) use a trigger-action model to allow users to create automation rules for supported web services and IoT devices. Compared with EPIDOSITE, those services can only support much a smaller set of devices. Smart home solution providers like Samsung, LG and Apple also provide software kits for end user development with their products, but unlike EPIDOSITE, they can only support devices within their own "ecosystems". Some generalized architectures and standardized models for programming IoT devices and higher-level representations of IoT automation rules and scripts have been proposed (e.g., [15, 16, 20, 21, 37]), but none have been widely adopted. In a field study for home automation systems [11], interoperability was shown to be important for the users. The same study showed that, although expert users reported that they facilitated communication between devices and services utilizing HTTP requests, this set-up would be difficult to configure for end users with limited technical background [11].

Visual programming tools such as Midgar [17], Jigsaw [22], Puzzle [9], Yahoo Pipes (http://pipes.yahoo.com/pipes) and Bipio (https://bip.io) enable the users to create programs, data pipelines and mashup applications between supported devices, services and APIs. AppsGate [7] provides a syntax-driven editor that empowers the users to program their smart home devices using a pseudo-natural language. Kubitza and Schmidt [25] proposed a web-based IDE that allows users to mash-up heterogeneous sets of devices using a common scripting language. However, the above approaches require the user to work with a visual programming language interface, a scripting language, or a domain-specific programming language (DSL), which is challenging for non-expert end users with little or no programming knowledge.

Some prior PBD systems such as motionEAP (www.motioneap.de) [40], Backpacks [38] and prior work in robot programming by demonstration (e.g., [1, 4]) enable users to program their desired behaviors by directly manipulating the physical objects (e.g., grabbing tools on an assembly line, or manipulating the arms of a robot). The *a CAPpella* system [12] empowers the user to specify a contextual situation by demonstration using data from multiple sensors in a smart environment. A major advantage of PBD is that it is easy to learn for end users with limited programming knowledge, as it allows users to program in the same environment in which they perform the actions [8, 30, 36]. However, these systems require the presence of extensive sensors on the devices and objects being programmed, or in the environment, which is costly and unavailable for most existing smart home devices and environments. As a comparison, EPIDOSITE shares these systems' advantages in usability and low learning barrier, but can work with most current IoT devices with available remote control Android apps without requiring extra sensors.

There are also PBD systems focusing on automating tasks on other domains, such as file management [34], photo manipulation [18] and web-based tasks [27]. The most

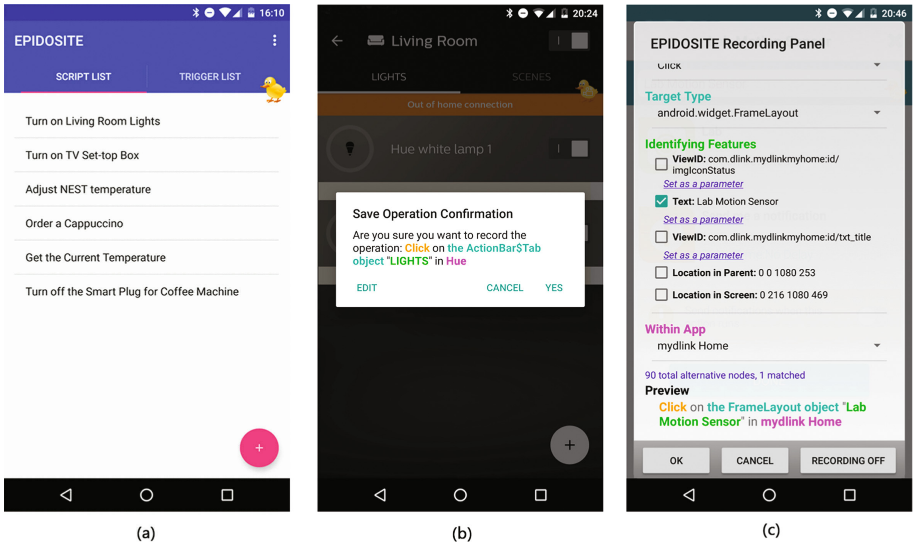
bam@cs.cmu.edu

relevant prior system to this work is SUGILITE [29], which is a PBD system for automating tasks on smartphones. Our EPIDOSITE system leverages the PBD engine from SUGILITE, extending its capabilities to support programming for IoT devices and provides new features specifically for smart home automation.

3 Example Usage Scenario

In this section, we use an example scenario to illustrate the procedure of creating an EPIDOSITE automation. For the scenario, we will create a script that turns on the TV set top box and turns off the light when someone enters the TV room. We will use a Verizon TV set-top box, a Philips Hue Go light and a D-Link DCH-S150 Wi-Fi motion sensor in this script. To our best knowledge, there exists no other EUD solution that can support all the above three devices.

First, the user starts EPIDOSITE (Fig. 1a), creates a new script and gives it a name. The phone then switches back to the home screen, and prompts the user to start demonstrating. The user now starts demonstrating how to turn off the light using the Philips Hue app – tapping on the Philips Hue icon on the home screen, choosing "Living Room", clicking on the "SCENES" tab, and selecting "OFF", which are the exactly same steps as when the user turns off the light manually using the same app. After each action, the user can see a confirmation dialog from EPIDOSITE (Fig. 1b). Running in the background as an Android accessibility service, the EPIDOSITE PBD engine can automatically detect the user's action and determine the features to use to identify the element using a set of heuristics (see [29] for more details), but the user can also manually edit the details of each recorded action in an editing panel (Fig. 1c). Figure 2a shows the textual



Turn on Living Room Lights
Turn on TV Set-top Box
Adjust NEST temperature
Order a Cappuccino
Get the Current Temperature
Turn off the Smart Plug for Coffee Machine

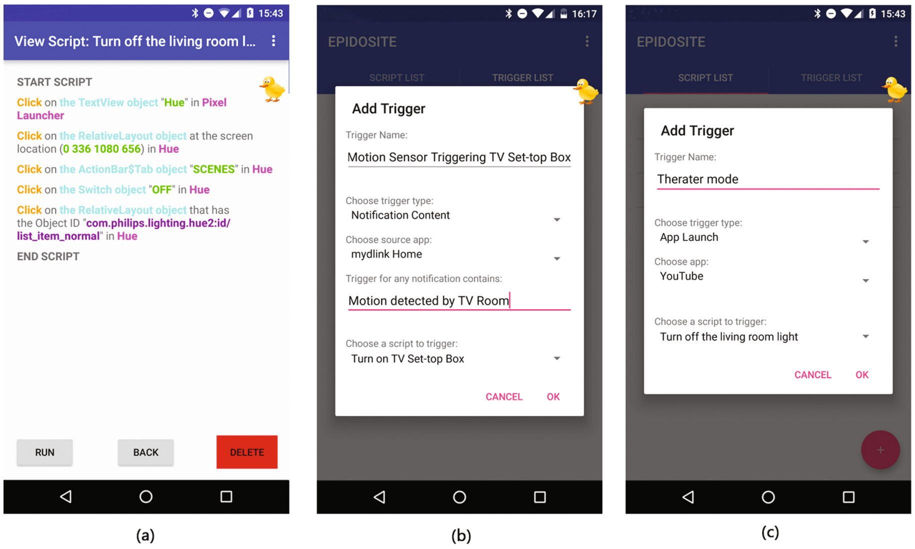
Save Operation Confirmation
Are you sure you want to record the operation: Click on the ActionBar$Tab object "LIGHTS" in Hue

Target Type
android.widget.FrameLayout
Identifying Features
☐ ViewID: com.dlink.mydlinkmyhome:id/imgIconStatus
Set as a parameter
✓ Text: Lab Motion Sensor
Set as a parameter
☐ ViewID: com.dlink.mydlinkmyhome:id/txt_title
Set as a parameter
☐ Location in Parent: 0 0 1080 253
☐ Location in Screen: 0 216 1080 469
Within App
mydlink Home
90 total alternative nodes, 1 matched
Preview
Click on the FrameLayout object "Lab Motion Sensor" in mydlink Home

Fig. 1. Screenshots of EPIDOSITE: (a) the main screen of EPIDOSITE showing a list of available scripts; (b) the confirmation dialog for an operation; (c) the editing panel for one operation.

bam@cs.cmu.edu

representation of the EPIDOSITE script created for turning off the light. Next, the user demonstrates turning on the TV set-top box using the SURE Universal Remote app on the phone (or other IR remote apps that support the Verizon TV set-top box), and EPIDOSITE records the procedure for that as well.



(a)

View Script: Turn off the living room l...

START SCRIPT

Click on the TextView object "Hue" in Pixel Launcher

Click on the RelativeLayout object at the screen location (0 336 1080 656) in Hue

Click on the ActionBar$Tab object "SCENES" in Hue

Click on the Switch object "OFF" in Hue

Click on the RelativeLayout object that has the Object ID "com.philips.lighting.hue2:id/list_item_normal" in Hue

END SCRIPT

(b)

Add Trigger

Trigger Name: Motion Sensor Triggering TV Set-top Box

Choose trigger type: Notification Content

Choose source app: mydlink Home

Trigger for any notification contains: Motion detected by TV Room

Choose a script to trigger: Turn on TV Set-top Box

(c)

Add Trigger

Trigger Name: Therater mode

Choose trigger type: App Launch

Choose app: YouTube

Choose a script to trigger: Turn off the living room light

Fig. 2. Screenshots of EPIDOSITE's script view and trigger creation interfaces: (a) the script view showing the script from the example usage scenario; (b) the window for creating an app notification trigger; (c) the window for creating an app launch trigger.

The user ends the demonstration recording and goes back to the EPIDOSITE app. She then clicks on the menu, and chooses "Add a Script Trigger". In the pop-up window (Fig. 2b), she gives the trigger a name, selects "Notification" as the type of the trigger, specifies "mydlink Home" (the mobile app for the D-Link motion sensor) as the source app, chooses the script she just created as the script to trigger, enters "Motion detected by TV room" as the activation string, and finally, clicks on "OK" to save the trigger. This trigger will execute the script every time that the "mydlink Home" app sends a notification that contains the string "Motion detected by TV room".

The steps shown above are the whole process to create this automation. Once the trigger is enabled, the background Android accessibility service waits for the activation of the trigger. When the motion sensor detects a motion, an Android notification is generated and displayed by the mydlink Home app. Then EPIDOSITE intercepts this notification, activates the trigger, executes the script, and manipulates the UI of the Philips Hue app and the SURE Universal Remote app to turn off the lights and to turn on the TV set-top box.

bam@cs.cmu.edu

4 System Design and Implementation

4.1 Implementation

The client component of Epidosite is an Android application written in Java. It does not require root access to the Android system, and should work on any smartphone with Android 4.4 or above. The client component is standalone. There is also an optional add-on server application available for supporting automation triggered by external web services through IFTTT. The server application is implemented in Java with Jersey[3] and Grizzly[4].

The mobile programming by demonstration engine used in Epidosite extends the prior mobile PBD system Sugilite [29]. Sugilite uses Android's accessibility API to support automating tasks in Android apps. During the demonstration, all of the user's interactions with the phone, together with the relevant UI elements on the screen, are recorded by a background recording handler through the accessibility API. Sugilite then processes the recording and generates a reusable script for performing the task. Sugilite also generalizes the script from the demonstrated example by analyzing the user interface structures of the apps used in the demonstration. For instance, if the user demonstrates how to turn on the living room light using the Philips Hue app, Sugilite will detect "living room" as a parameter of this script, and automatically generalizes the script so it can be used to turn on other available Philips Hue lights by providing a different string when the script is invoked. With Sugilite, the user can record an automation by demonstrating the procedure of performing the task using the user interfaces of any third-party Android app (with some exceptions noted in [29]), and then run the automation through a multi-modal interface, invoked through the GUI or using speech. Sugilite also supports the viewing and editing of the scripts. Details about Sugilite can be found in the earlier paper [29].

On top of Sugilite, Epidosite adds new features and mechanisms to support the programming of IoT devices in the smart home setting, including new ways for triggering scripts, new ways for scripts to trigger external services and devices, and new mechanisms for sharing information among devices. To better meet the needs of developing for IoT devices, Epidosite also supports programming for different devices in separated subscripts, and reusing the subscripts in new scripts. For example, the user can demonstrate the two scripts for "turning off the light" and "turning on the TV", and then create a new script of "if …, turn off the light and turn on the TV") without having to demonstrate the procedures for performing the two tasks again.

4.2 Key Features

Notification and App Launch Triggers

The most common context-aware applications in the smart home are naturally described using rule-based conditions in the model of trigger-action programming, where a *trigger*

[3] https://jersey.java.net/.
[4] https://grizzly.java.net/.

bam@cs.cmu.edu

describes a condition, an event or a scenario, and an *action* specifies the desired behavior when its associated trigger is activated [13, 42]. In Epidosite, scripts can be triggered by the content of Android notifications, or as a result of the launch of a specified app on the phone.

Epidosite keeps a background Android accessibility service running at all times. Through the Android accessibility API, the service intercepts system notifications and app launches. If the content of a new notification contains the activation string of a stored notification trigger (as shown in the example usage scenario), or an app associated with an app launch trigger has just launched, the corresponding automation script for the trigger will be executed. Figure 2c shows the interface with which the user can create an automation that turns off the light when the YouTube app launches, after first creating the "Turn off the living room light" script by demonstration.

These features allow scripts to be triggered not only by mobile apps for IoT devices, as shown in the example usage scenario, but also by other third-party Android apps. Prior research has shown that the usage of smartphone apps is highly contextual [6, 23] and also varies [45] for different groups of users. By allowing the launching of apps and the notifications from apps to trigger IoT scripts, the user can create highly context-aware automation with Epidosite, for example, to change the color of the ambient lighting when the Music Player is launched, to adjust the thermostat when the Sleep Monitor app is launched, or even to warm up the car when the Alarm app rings (and sends the notification) on winter mornings.

Using the above two types of triggers, along with the external service trigger introduced in the next section, user-demonstrated scripts for smartphone apps can also be triggered by readings and status of IoT sensors and devices. As shown in [29], many common tasks can be automated using PBD on smartphone apps. Some examples are ordering a coffee (using the Starbucks app), requesting cabs (Uber app), sending emails, etc. Epidosite empowers users to integrate IoT smart devices with available smartphone apps to create context-aware and responsive automations.

External Service Triggers

To expand the capabilities of Epidosite to leverage all of the available resources, we implemented a server application that allows Epidosite to integrate with the popular

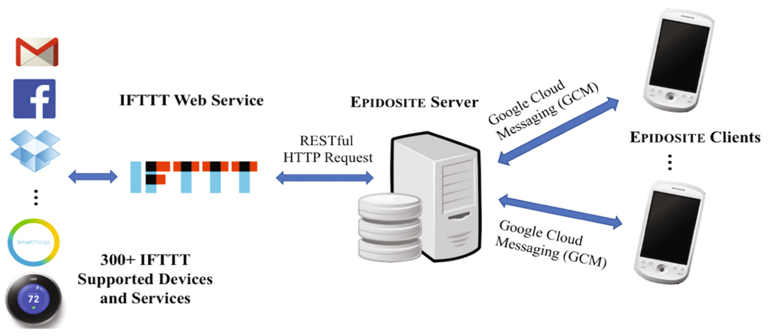

IFTTT Web Service
IFTTT
300+ IFTTT Supported Devices and Services
RESTful HTTP Request
Epidosite Server
Google Cloud Messaging (GCM)
Google Cloud Messaging (GCM)
Epidosite Clients
72

Fig. 3. The architecture of the Epidosite external service trigger mechanism.

bam@cs.cmu.edu

automation service IFTTT. Through this integration, an Epidosite script can be triggered by over 360 web services supported by IFTTT, including social networks (e.g., Facebook, Twitter), news (e.g., ESPN, Feedly), email, calendar management, weather and supported devices like smart hubs (e.g., Google Home, Amazon Echo), smart appliances, fitness trackers, home monitors, and smart speakers. An Epidosite script can also be used to trigger actions for IFTTT-supported services. Figure 3 shows the overall architecture for supporting external service triggers, consisting of the client side, the server side, the IFTTT service and how they communicate.

An IFTTT applet consists of two parts: a trigger and an action, in which either part can be an Epidosite script. If an Epidosite script is used as the trigger, then an HTTP request will be sent out to IFTTT via the Epidosite server to execute the corresponding IFTTT action when the trigger is activated. Similarly, if an Epidosite script is used as the action, then it will be executed on the corresponding client smartphone upon the client application receiving a Google Cloud Messaging (GCM) message sent by the Epidosite server when the associated IFTTT trigger is activated. The Epidosite server communicates with IFTTT through the IFTTT Maker channel, which can receive and make RESTful HTTP web requests. The Epidosite server side application is also highly scalable and can handle multiple clients at the same time.

To create an IFTTT triggered script, the user first creates an Epidosite script for the "action" part by demonstration, where the user records the procedure of performing the desired task by manipulating the phone apps. Then, the user goes to IFTTT, chooses "New Applet," and chooses a trigger for the script. After this, the user chooses the Maker channel as the action. For the address for the web request, the Epidosite app on the phone will automatically generate a URL which the user can just paste into this field. The auto-generated URL is in the format of:

http://[SERVER_ADDRESS]/client=[CLIENT_NAME]&scriptname=[SCRIPT_NAME]

where *[SERVER ADDRESS]* is the address of the Epidosite server, *[CLIENT_NAME]* is the name of the Epidosite client (which by default is the combination of the phone owner's name and the phone model. e.g., "Amy's Nexus 6") and *[SCRIPT_NAME]* is the name of the Epidosite script to trigger. The user can just paste this URL into the IFTTT field (see Fig. 4).

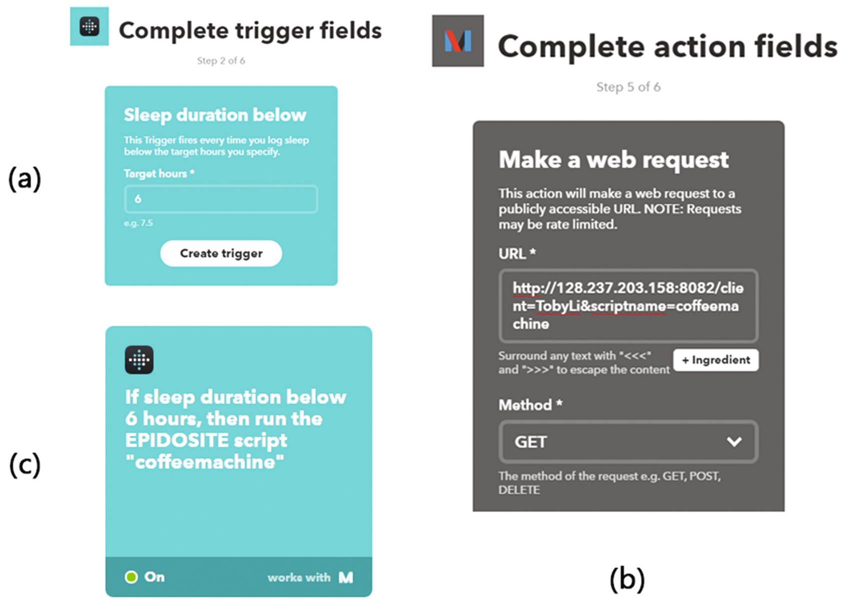
The procedure to create an Epidosite-triggered IFTTT applet is similar, except that the user needs to add "trigger an IFTTT applet" as an operation when demonstrating the Epidosite script, and then use the Maker channel as the trigger.

Cross-app Interoperability

Interoperability among IoT devices has been an long-time important challenge [14]. Sharing data across devices from different "ecosystems" often requires the user to manually setup the connection using techniques like HTTP requests, which require carefully planning and extensive technical expertise [11]. Middleware like Bezirk[5] and [5, 16, 24, 41] supports IoT interoperation and provides a high-level representation

[5] http://developer.bezirk.com/.

bam@cs.cmu.edu



Complete trigger fields

Sleep duration below

This Trigger fires every time you log sleep below the target hours you specify.

Target hours *

6

e.g. 7.5

If sleep duration below 6 hours, then run the EPIDOSITE script "coffeemachine"

Complete action fields

Make a web request

This action will make a web request to a publicly accessible URL. NOTE: Requests may be rate limited.

URL *

http://128.237.203.158:8082/client=TobyLi&scriptname=coffeemachine

Surround any text with "<<<" and ">>>" to escape the content

Method *

GET

The method of the request e.g. GET, POST, DELETE

Fig. 4. Creating an IFTTT applet that triggers an Epidosite script: (a) creating the trigger condition “sleep duration below 6 h” using the Fitbit activity tracker; (b) creating the action of running the Epidosite script “coffeemachine” using the URL generated by Epidosite; (c) the IFTTT applet created.

model for common IoT devices, but these also require the user to have sophisticated programming skills.

Epidosite supports the user in extracting the value of a TextView object in the user interface of the an app by using a gesture during the demonstration, storing the value in a variables and then using the values saved in these variables later in the script. All the user needs to do to save a value is to click on the “Get a Text Element on the Screen” option from the recording menu while demonstrating, circle the desired element on the screen using a finger gesture (the yellow stroke in Fig. 5), and select the element in a pop-up menu (see Fig. 5). Later when the user needs to enter a string in a script, the user can instead choose to use the value from a previously created variable.

When a script is executed that contains a value extraction operation, Epidosite will automatically navigate as demonstrated to the user interface where the desired value is located, and then will dynamically extract the value based on the resource ID and the location of the corresponding TextView object. This approach does not require the involved app to have any API or other data export mechanism. As long as the desired value is displayed as a text string in the app, the value can be extracted and used in other parts of the script.

Currently, Epidosite only supports using the extracted values in later operations exactly as they were displayed. As future work, we plan to support the reformatting and transformation of the variable values, as well as common arithmetic and string operations on the values.

bam@cs.cmu.edu

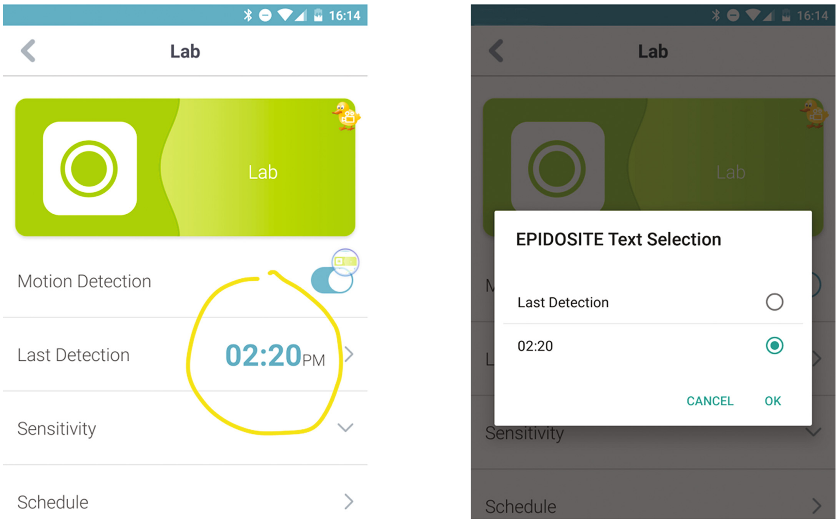


Fig. 5. Extracting the time of the last detection from a D-link motion sensor in the Mydlink Home app using a circle gesture in EPIDOSITE (Color figure online)

5 Limitations and Future Work

The current version of EPIDOSITE has several limitations. First, for executing an automation, the phone must be powered on and concurrently connected to all the devices involved in the automation. If the phone is powered off, or disconnected from the involved IoT devices, the automation will fail to execute. This limitation will particularly affect EPIDOSITE's applicability for devices that are connected to the phone via a local network or through a short-range wireless communication technology (e.g., Bluetooth, ZigBee, IR), since with these devices, the phone is restricted to be connected to the local network, or physically within range of the wireless connection for the automation to work. Second, EPIDOSITE automations need to run in the foreground on the main UI thread of the phone. Thus, if an automation is triggered when the user is actively operating the phone at the same time (e.g., if the user is on a phone call), then the user's interaction with the phone will be interrupted. The automation execution may also fail if it is interrupted by a phone event (e.g., an incoming phone call) or by the user's action.

For some of the above limitations, an approach is to use a separate phone as the hub for IoT automation, and to run EPIDOSITE on that phone instead of using the user's primary smartphone. By doing this, the separate phone can be consistently plugged in and stay connected with the IoT devices to ensure that the automations can be triggered and executed properly. Currently, a compatible Android phone can be purchased for less than $50, which makes this solution affordable.

Further limitations include that the current version of EPIDOSITE provides little assistance in testing and debugging. When an automation uses a trigger that cannot be activated manually (e.g., at a future time, or due to a weather condition), the user may not be able to demonstrate or test this automation. In the case that the controlling app of an

bam@cs.cmu.edu

IoT device is updated, the user may need to record the demonstration again if the user interface of the app has changed, or if the procedure to complete the task is now different. For the same reason, EPIDOSITE automation scripts may break if they are executed in an environment with different IoT devices available, with different versions of smartphone apps running, or on a smartphone with a different screen resolution. This limits the sharing of EPIDOSITE scripts, and may cause runtime problems for EPIDOSITE if the current software or hardware environment changes. To fix an error during recording, the user currently can only either record again from scratch, or manually edit the automation script using the editing panel (Fig. 1c), which is not easy to use for an end user. For future development, we plan to explore the designs of new end-user friendly testing and debugging interfaces and new error handling mechanisms that can automatically modify the script in case of a minor software update or when a different phone is used. The improvements will also facilitate the sharing of EPIDOSITE scripts among different users.

Due to the technical limitations in the current programming by demonstration engine in the system, tasks that involve the use of web applications are not supported by EPIDOSITE. EPIDOSITE also does not yet support recording gestures and sensory inputs (e.g., accelerometer, gyroscope) either, but these are planned for the future.

In this work, we enabled the user to trigger automations for IoT devices based on the usage context of smartphones by providing the notification and app launch triggers. For future work, we plan to generalize these capabilities, to design usable and easily understandable ways for the end users to create automations combining the contents displayed by various apps, the inputs from the available phone sensors, and the available personal data on the phone. For example, using the location sensor of the phone, one could enable different automation scripts for the same trigger depending on where the user was. Existing context-aware computing research have contributed many technical solutions for transforming the smartphone usage and sensor data into more meaningful behavioral-centric personal data [43]. EPIDOSITE offers opportunities to connect user-centric behaviors on smartphones and users' smart home environments. We hope this will empower the end users to create more intelligent and useful context-aware applications.

We also plan to explore how to make it easier for end users to create more complex automations with control structures such as compound conditionals and loops. AppsGate [7] supports creating control structures in a syntax-driven editor using a pseudo-natural language, which was shown to be easy for creating simple rules, but difficult for creating compound conditionals. SmartFit [2] introduces an interactive graphical editor with which end users can create conditionals using IoT sensor data for rule-based EUD systems. However, it still remains a major design challenge to make it easier for end users to express the logic and the control structures in programming by demonstration systems. Gamut [31, 32] is a PBD system that can infer programming logic from multiple examples from the user, but it has been shown to be hard for non-expert end users to provide meaningful and helpful additional examples for inferring programming logics [26, 31, 32]. We are currently investigating the approach of having the users talk about their intentions using speech during the demonstration, and inferring the programming logic using techniques from AI and natural language processing [28].

bam@cs.cmu.edu

6 Conclusion

In this work, we introduce EPIDOSITE, a new programming by demonstration system that makes it possible for end users to create automations for consumer IoT devices on their smartphones. It supports programming across multiple IoT devices and exchanging information among them without requiring the devices to be of the same brand or within the same "ecosystem". The programming by demonstration approach minimizes the necessity to learn programming concepts. EPIDOSITE also supports using arbitrary third-party Android mobile apps and hundreds of available web services in the scripts to create highly context-aware and responsive automations.

Acknowledgement. This work was supported in part by the Yahoo! InMind project.

References

1. Argall, B.D., et al.: A survey of robot learning from demonstration. Robot. Auton. Syst. **57**(5), 469–483 (2009)
2. Barricelli, B.R., Valtolina, S.: A visual language and interactive system for end-user development of internet of things ecosystems. J. Vis. Lang. Comput. (2017, in press)
3. Barricelli, B.R., Valtolina, S.: Designing for end-user development in the Internet of Things. In: Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. (eds.) IS-EUD 2015. LNCS, vol. 9083, pp. 9–24. Springer, Cham (2015). doi:10.1007/978-3-319-18425-8_2
4. Billard, A., et al.: Robot programming by demonstration. In: Siciliano, B., Khatib, O. (eds.) Springer Handbook of Robotics, pp. 1371–1394. Springer, Heidelberg (2008)
5. Blackstock, M., Lea, R.: IoT interoperability: a hub-based approach. In: 2014 International Conference on the Internet of Things (IOT), pp. 79–84. IEEE (2014)
6. Böhmer, M., et al.: What's in the apps for context? Extending a sensor for studying app usage to informing context-awareness. In: Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication, pp. 1423–1426. ACM, New York (2013)
7. Coutaz, J., Crowley, J.L.: A first-person experience with end-user development for smart homes. IEEE Pervasive Comput. **15**(2), 26–39 (2016)
8. Cypher, A., Halbert, D.C.: Watch What I Do: Programming by Demonstration. MIT Press, Cambridge (1993)
9. Danado, J., Paternò, F.: Puzzle: a visual-based environment for end user development in touch-based mobile phones. In: Winckler, M., Forbrig, P., Bernhaupt, R. (eds.) HCSE 2012. LNCS, vol. 7623, pp. 199–216. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34347-6_12
10. De Russis, L., Corno, F.: HomeRules: a tangible end-user programming interface for smart homes. In: Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems, pp. 2109–2114. ACM, New York (2015)
11. Demeure, A., Caffiau, S., Elias, E., Roux, C.: Building and using home automation systems: a field study. In: Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. (eds.) IS-EUD 2015. LNCS, vol. 9083, pp. 125–140. Springer, Cham (2015). doi: 10.1007/978-3-319-18425-8_9
12. Dey, A.K., et al.: A CAPpella: programming by demonstration of context-aware applications. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 33–40. ACM (2004)

bam@cs.cmu.edu

13. Dey, A.K., et al.: iCAP: interactive prototyping of context-aware applications. In: Fishkin, K.P., et al. (eds.) Pervasive Computing, pp. 254–271. Springer, Berlin Heidelberg (2006)
14. Edwards, W.K., Grinter, R.E.: At home with ubiquitous computing: seven challenges. In: Abowd, Gregory D., Brumitt, B., Shafer, S. (eds.) UbiComp 2001. LNCS, vol. 2201, pp. 256–272. Springer, Heidelberg (2001). doi:10.1007/3-540-45427-6_22
15. Fox, A., et al.: Integrating information appliances into an interactive workspace. IEEE Comput. Graph. Appl. **20**(3), 54–65 (2000)
16. Gama, K., et al.: Combining heterogeneous service technologies for building an Internet of Things middleware. Comput. Commun. **35**(4), 405–417 (2012)
17. González García, C., et al.: Midgar: generation of heterogeneous objects interconnecting applications. a domain specific language proposal for Internet of Things scenarios. Comput. Netw. **64**, 143–158 (2014)
18. Grabler, F., et al.: Generating photo manipulation tutorials by demonstration. In: ACM SIGGRAPH 2009 Papers. pp. 66:1–66:9. ACM, New York (2009)
19. Greenberg, S.: Context as a dynamic construct. Hum.-Comput. Interact. **16**(2), 257–268 (2001)
20. Guinard, D., et al.: Towards physical mashups in the web of things. In: 2009 Sixth International Conference on Networked Sensing Systems (INSS), pp. 1–4. IEEE (2009)
21. Guinard, D., Trifa, V.: Towards the web of things: web mashups for embedded devices. In: Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in Proceedings of WWW (International World Wide Web Conferences), Madrid, Spain (2009)
22. Humble, J., Crabtree, A., Hemmings, T., Åkesson, K.-P., Koleva, B., Rodden, T., Hansson, P.: "Playing with the bits" user-configuration of ubiquitous domestic environments. In: Dey, Anind K., Schmidt, A., McCarthy, Joseph F. (eds.) UbiComp 2003. LNCS, vol. 2864, pp. 256–263. Springer, Heidelberg (2003). doi:10.1007/978-3-540-39653-6_20
23. Jesdabodi, C., Maalej, W.: Understanding usage states on mobile devices. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 1221–1225. ACM, New York (2015)
24. Katasonov, A., et al.: Smart semantic middleware for the Internet of Things. ICINCO-ICSO **8**, 169–178 (2008)
25. Kubitza, T., Schmidt, A.: Towards a toolkit for the rapid creation of smart environments. In: Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. (eds.) IS-EUD 2015. LNCS, vol. 9083, pp. 230–235. Springer, Cham (2015). doi:10.1007/978-3-319-18425-8_21
26. Lee, T.Y., et al.: Towards understanding human mistakes of programming by example: an online user study. In: Proceedings of the 22nd International Conference on Intelligent User Interfaces, pp. 257–261. ACM, New York (2017)
27. Leshed, G., et al.: CoScripter: automating and sharing how-to knowledge in the enterprise. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1719–1728. ACM, New York (2008)
28. Li, T.J.-J., et al.: Designing a conversational interface for a multimodal smartphone programming-by-demonstration agent. In: Conversational UX Design CHI 2017 Workshop, Denver, CO (2017, in press)
29. Li, T.J.-J., et al.: SUGILITE: creating multimodal smartphone automation by demonstration. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. ACM, Denver (2017, in press)
30. Lieberman, H.: Your Wish is My Command: Programming by Example. Morgan Kaufmann, San Francisco (2001)

bam@cs.cmu.edu

31. McDaniel, R.G., Myers, B.A.: Gamut: demonstrating whole applications. In: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, pp. 81–82 ACM, New York (1997)
32. McDaniel, R.G., Myers, B.A.: Getting more out of programming-by-demonstration. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 442–449 ACM, New York (1999)
33. Mennicken, S., et al.: From today's augmented houses to tomorrow's smart homes: new directions for home automation research. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 105–115. ACM, New York (2014)
34. Modugno, F., Myers, B.A.: Pursuit: graphically representing programs in a demonstrational visual shell. In: Conference Companion on Human Factors in Computing Systems, pp. 455–456. ACM, New York (1994)
35. Myers, B.A.: Demonstrational interfaces: a step beyond direct manipulation. Computer **25**(8), 61–73 (1992)
36. Myers, B.A.: Visual programming, programming by example, and program visualization: a taxonomy. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 59–66. ACM, New York (1986)
37. Pintus, A., et al.: The anatomy of a large scale social web for internet enabled objects. In: Proceedings of the Second International Workshop on Web of Things. pp. 6:1–6:6. ACM, New York (2011)
38. Raffle, H., et al.: Beyond record and play: backpacks: tangible modulators for kinetic behavior. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 681–690. ACM, New York (2006)
39. Ricquebourg, V., et al.: The smart home concept: our immediate future. In: 2006 1st IEEE International Conference on E-Learning in Industrial Electronics, pp. 23–28 (2006)
40. Schmidt, A.: Programming ubiquitous computing environments. In: Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. (eds.) IS-EUD 2015. LNCS, vol. 9083, pp. 3–6. Springer, Cham (2015). doi:10.1007/978-3-319-18425-8_1
41. Song, Z., et al.: Semantic middleware for the Internet of Things. In: 2010 Internet of Things (IOT), pp. 1–8 (2010)
42. Ur, B., et al.: Practical trigger-action programming in the smart home. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 803–812. ACM, New York (2014)
43. Wiese, J.S.: Evolving the Ecosystem of Personal Behavioral Data (2015)
44. Zhang, T., Brügge, B.: Empowering the user to build smart home applications. In: Proceedings of 2nd International Conference on Smart Homes and Health Telematics (ICOST2004), Singapore (2004)
45. Zhao, S., et al.: Discovering different kinds of smartphone users through their application usage behaviors. In: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 498–509. ACM, New York (2016)
46. Zhong, Y., et al.: Smart home on smart phone. In: Proceedings of the 13th International Conference on Ubiquitous Computing, pp. 467–468. ACM, New York (2011)

bam@cs.cmu.edu

Simone Barbosa
Panos Markopoulos
Fabio Paternò
Simone Stumpf
Stefano Valtolina (Eds.)

LNCS 10303

End-User Development

6th International Symposium, IS-EUD 2017
Eindhoven, The Netherlands, June 13–15, 2017
Proceedings

Springer

Lecture Notes in Computer Science 10303

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, Lancaster, UK
Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA
Josef Kittler
University of Surrey, Guildford, UK
Jon M. Kleinberg
Cornell University, Ithaca, NY, USA
Friedemann Mattern
ETH Zurich, Zurich, Switzerland
John C. Mitchell
Stanford University, Stanford, CA, USA
Moni Naor
Weizmann Institute of Science, Rehovot, Israel
C. Pandu Rangan
Indian Institute of Technology, Madras, India
Bernhard Steffen
TU Dortmund University, Dortmund, Germany
Demetri Terzopoulos
University of California, Los Angeles, CA, USA
Doug Tygar
University of California, Berkeley, CA, USA
Gerhard Weikum
Max Planck Institute for Informatics, Saarbrücken, Germany

bam@cs.cmu.edu

More information about this series at http://www.springer.com/series/7408

bam@cs.cmu.edu

Simone Barbosa · Panos Markopoulos
Fabio Paternò · Simone Stumpf
Stefano Valtolina (Eds.)

End-User Development

6th International Symposium, IS-EUD 2017
Eindhoven, The Netherlands, June 13–15, 2017
Proceedings

Springer

bam@cs.cmu.edu

Editors
Simone Barbosa
Pontifical Catholic University of
Rio de Janeiro
Rio de Janeiro
Brazil

Panos Markopoulos
Eindhoven University of Technology
Eindhoven
The Netherlands

Fabio Paternò
C.N.R. - ISTI
Pisa
Italy

Simone Stumpf
City University of London
London
UK

Stefano Valtolina
Università degli Studi di Milano
Milan
Italy

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-58734-9 ISBN 978-3-319-58735-6 (eBook)
DOI 10.1007/978-3-319-58735-6

Library of Congress Control Number: 2017940385

LNCS Sublibrary: SL2 – Programming and Software Engineering

© Springer International Publishing AG 2017
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

bam@cs.cmu.edu