Block-Based Approaches to Internet of Things in MIT App Inventor

Wen Xi
MIT CSAIL
Massachusetts Institute of Technology
Cambridge, MA, USA
wenxi@mit.edu
Department of Computing
Hong Kong Polytechnic University
Kowloon, Hong Kong, China
wen.cy.wen@connect.polyu.hk

Evan W. Patton
MIT CSAIL
Massachusetts Institute of Technology
Cambridge, MA, USA
ewpatton@mit.edu

Abstract

Internet of Things (IoT) integrates physical devices and creates opportunities for people to interact with the surrounding environment. While a number of blocks-based approaches exist for programming some hardware, such as Snap! for Arduino, Microblocks, and Scratch extensions, this is still an underexplored area. In this paper, we propose a block-based programming approach using MIT App Inventor to enable novices be able to build mobile apps integrated with IoT technology. We also review other block languages applied for IoT. We conclude with some thoughts on how blocks languages might inspire people to create with IoT.

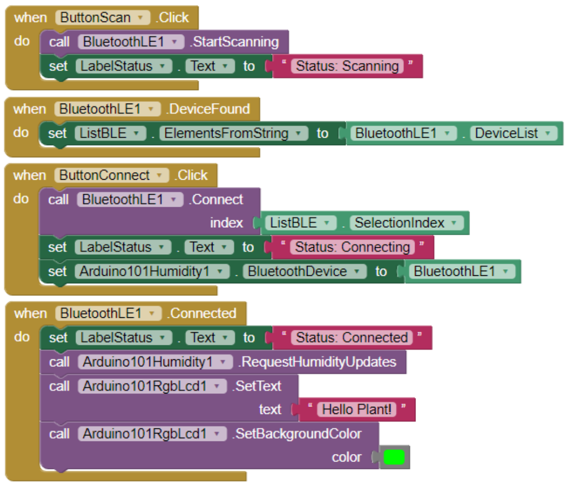
Keywords Internet of Things, computational action, block-based programming, MIT App Inventor

1 Introduction

MIT App Inventor is an intuitive, block-based programming environment that allows beginner programmers to build functional apps for smart phones and tablets. MIT App Inventor team has provided IoT extensions [2] to allow people to design and create apps to interact with physical devices. The IoT extensions are built on Bluetooth Low Energy (BLE) to exchange data like receiving temperature, humidity, heart rate from sensors, and sending out signals to control robots.

2 Use Cases

This Healthy Plant Monitoring App [1] is a use case for MIT App Inventor IoT that monitors plants' humidity, temperature, light and moisture, by using the IoT extension's provided blocks. After setting up Arduino 101 (a microcontroller) to work with App Inventor, users should design their app interface by dragging components to the designer screen. Some sample blocks are here for finding and connecting BLE devices by call-back functions, and requesting sensor data such as humidity (Figure 1). After that, user could export



```
when ButtonScan .Click
do  call BluetoothLE1 .StartScanning
    set LabelStatus . Text to "Status: Scanning"

when BluetoothLE1 .DeviceFound
do  set ListBLE . ElementsFromString to BluetoothLE1 . DeviceList

when ButtonConnect .Click
do  call BluetoothLE1 .Connect
                      index ListBLE . SelectionIndex
    set LabelStatus . Text to "Status: Connecting"
    set Arduino101Humidity1 . BluetoothDevice to BluetoothLE1

when BluetoothLE1 .Connected
do  set LabelStatus . Text to "Status: Connected"
    call Arduino101Humidity1 .RequestHumidityUpdates
    call Arduino101RgbLcd1 .SetText
                           text "Hello Plant!"
    call Arduino101RgbLcd1 .SetBackgroundColor
                           color 🟩
```

Figure 1. Blocks used to establish a BLE connection with an Arduino and request sensor updates

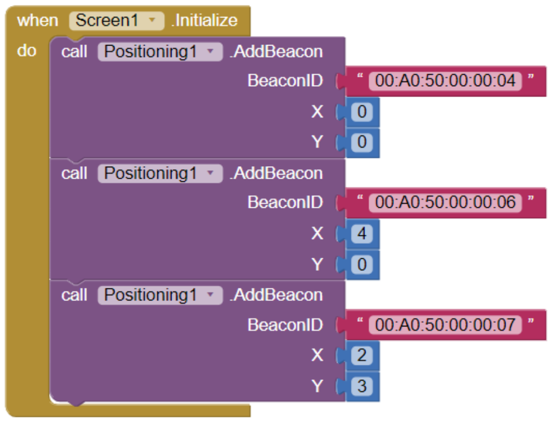
the .apk file and install it on their phone to track the overall health of the plant.

The Indoor Positioning App is another use case on MIT App Inventor that collects Received Signal Strength Indication (RSSI) and calculate the phone's indoor position by using IoT BLE extension together with programmer self-defined blocks. Programmers created a positioning extension with customized blocks to input beacon information (Figure 2), collect RSSI from BLE extension blocks and use their own algorithm to calculate the positioning (Figure 3).

3 Implementation

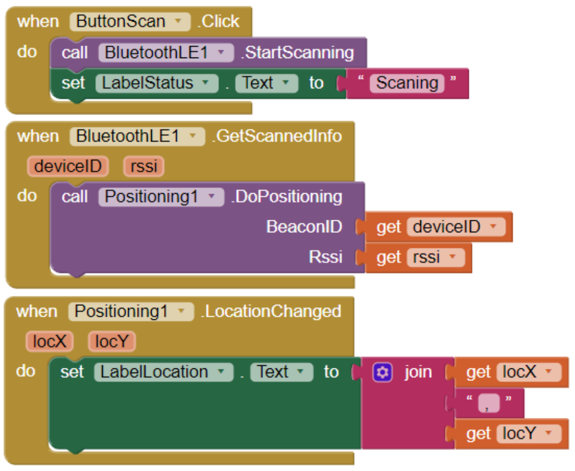
The MIT App Inventor Arduino 101 extension wraps functionality, including requesting sensor data and writing values to the device, to make it easy to understand and modify apps. The blocks for Indoor Positioning are also relatively high-level and abstract. For example, the `DoPositioning` block filters RSSI, converts RSSI to distance, and calculates distance from three beacons. This way, the user could just

BLOCKS+, November 04, 2018, Boston, MA, USA
2018.



```
when Screen1 .Initialize
do  call Positioning1 .AddBeacon
                      BeaconID  " 00:A0:50:00:00:04 "
                             X  0
                             Y  0
    call Positioning1 .AddBeacon
                      BeaconID  " 00:A0:50:00:00:06 "
                             X  4
                             Y  0
    call Positioning1 .AddBeacon
                      BeaconID  " 00:A0:50:00:00:07 "
                             X  2
                             Y  3
```

Figure 2. Blocks for beacons configuration



```
when ButtonScan .Click
do  call BluetoothLE1 .StartScanning
    set LabelStatus . Text to " Scaning "

when BluetoothLE1 .GetScannedInfo
  deviceID  rssi
do  call Positioning1 .DoPositioning
                      BeaconID  get deviceID
                          Rssi  get rssi

when Positioning1 .LocationChanged
  locX  locY
do  set LabelLocation . Text to  join  get locX
                                       " , "
                                       get locY
```

Figure 3. Blocks for receiving RSSI, calculate and update x,y location

drag and use the blocks `DoPositioning` without needing to understand the complex mathematics behind it.

4 Literature Review

Beside MIT App Inventor, there are other block-based programming environments supporting Arduino (e.g., Scratch, Snap4Arduino, Microblocks, BlocklyDuino) with different block design. The blocks in the Scratch Arduino 101 extension provides the very elementary functions of setting the pin mode, reading and writing pin value, and events on pin value comparison. Blocklyduino as well as Snao4Arduino program the device and aim to abstract connected hardware into different categories so that users don't necessarily need to recall the matching between pins and devices, visualizing the code to be more directly perceived. Microblocks allows for developing a program using a virtual machine running on the Arduino.

Blocks languages target a breadth of platforms. MIT App Inventor particularly targets mobile apps interacting with IoT. Scratch supports Arduino and micro:bit by extensions, but targets programs running on web browser. Snap4Arduino as a visual programming language version modified from the Snap!, supports different Arduino boards as well while like Snap!, users in Snap4Arduino could only see the programs running on web browsers. BlocklyDuino is a web-based visual programming Arduino editor, generates the source code in textual format and needs the user manually copying it to Arduino to get the result [3]. Microblocks targets a virtual machine running on the target device. These many options provide users flexibility in their choice of blocks language for IoT.

5 Discussion

Dragging and combining blocks is a low barrier to entry to allow people to enjoy the experience of learning and playing with IoT. Blocks allow one to code without paying excessive attention on coding syntax. Including pictures or text to represent different devices could make the code more closely represents the objects, reducing cognitive load, which is especially beneficial for beginners and children. Ultimately, we expect blocks-based languages to make IoT approachable to general population.

MIT App Inventor is a relatively mature platform in terms of creating Android apps and supports lots of functionality by extensions. Users can implement creative ideas with IoT and easily carry to use and show to friends on smart phones. Furthermore, by allowing customized blocks, it's efficient for experienced programmers by accelerating IoT application prototyping.

6 Conclusions

There are several block-based programming environments which build apps for different environment and have perspective features. As one of them, and intuitive, visual programming platform, MIT App Inventor has a distinguishing feature focusing on mobile apps and expanding more functions related to IoT to enlarge its usability, while still keeps its simplicity for novice programmers and even children. By deploying code to mobile devices, MIT App Inventor makes access to IoT portable for anyone to carry and show to friends.

References

[1] MIT App Inventor. 2017. App Inventor + IoT: Building a Healthy Plant Monitoring App. http://iot.appinventor.mit.edu/assets/tutorials/MIT_App_Inventor_IoT_Healthy_Plant.pdf. Accessed on 2018-08-18.

[2] MIT App Inventor. 2017. Internet of Things. http://iot.appinventor.mit.edu/#/. Accessed on 2018-08-18.

[3] J. Judvaitis, A. Elsts, and L. Selavo. 2013. Demo Abstract: SEAL-Blockly: Sensor Network Visual Programming Using a Web Browser. In *10th European Conference of Wireless Sensor Networks.*