BLAST: Block Applications for Things

Michael Freund[1][0000−0003−1601−9331], Thomas Wehr[1][0000−0002−0678−5019], and Andreas Harth[1,2][0000−0002−0702−510X]

[1] Fraunhofer Institute for Integrated Circuits IIS, Nürnberg, Germany
firstname.lastname@iis.fraunhofer.de
[2] Friedrich–Alexander University Erlangen-Nürnberg, Nürnberg, Germany

Abstract. We introduce a block-based visual programming language called BLAST to write programs involving connected devices with a Web of Things abstraction. We developed an editor and an execution environment for BLAST programs that runs in a web browser. We demonstrate that BLAST can be used to create programs that interact with a variety of devices. In particular we show the use of connected devices in a geofencing scenario.

Keywords: Block-based Programming · Web of Things.

1 Introduction

Connected devices form the basis for a wide variety of applications in industry settings, such as condition monitoring, energy consumption analysis, simulation and optimisation [2]. The Web of Things (WoT)[1] aims at providing a set of standardised technologies to help simplify the creating of applications involving connected devices. At the same time, casual users should be able to quickly program applications involving connected devices. The programming languages currently in use for accessing devices are primarily text-based and are difficult to access for casual users. One possible approach to improve accessibility and simplify the creation of computer programs is the visual programming paradigm.

Visual programming does not require the user to enter any text; instead, programs are based on graphical elements like blocks, graphs, tables or diagrams. Programs are created by arranging predefined blocks with images or text are arranged into programs via a drag-and-drop interface. Blocks can be geometrically aligned with other compatible blocks, like in a jigsaw puzzle, to form complex programs. Since blocks can only be arranged in the correct way, no syntax errors can occur [9] [5] [4].

The created block programs can be translated into any predefined programming language, resulting in a normal source code. The block-based approach can be used not only for learning programming concepts, but also for more complex problems. For example, block languages have been successfully used in programming industrial robots [10] [3] [8] or executing SPARQL queries [1], both areas which normally require advanced programming skills.

[1] https://www.w3.org/WoT/

We present BLAST, a visual programming environment that works with devices following the Web of Things interface. BLAST programs can be created in a web browser, based on Google's Blockly[2], a JavaScript library for creating block-based languages and editors. BLAST programs can be translated to JavaScript and executed in the web browser. BLAST implements the Web of Things abstraction to devices with Bluetooth Low Energy (BLE) and USB Human Interface Devices (HID) interface via APIs of modern web browsers.

The currently best known tool for programming IoT devices using visual programming is Node-RED[3]. In contrast to BLAST, which focuses on control flow constructs, Node-RED's programming paradigm is based on dataflows. The use cases in industrial settings we envision for BLAST involve mostly control flow, and in such scenarios a dataflow abstraction can lead to programs that are difficult to understand and maintain.

Another system similiar to BLAST is Punya [7], an Android app development system based on the MIT App Inventor[4]. Like BLAST, Punya offers a block-based programming environment able to run SPARQL queries and include online services. Punya focuses on creating applications for Android devices and communicates with the mobile phones's internal hardware, in addition to working as a LDP-CoAP client to publish and access data on CoAP servers. BLAST programs, on the other hand, run in modern web browsers and support devices based on a Web of Things abstraction.

In the following we first introduce the BLAST language using a geofencing scenario, next describe the browser-based editor and execution environment for BLAST programs and then conclude.

2 BLAST Language

When choosing the vocabulary for our block-based language BLAST, we decided on a mixture of natural language and computer language [6]. In BLAST, the blocks are arranged into readable sentences, as in a natural language, but the blocks also contain a few technical programming terms, such as "repeat while true" instead of "forever".

The BLAST language includes standard elements from other programming languages such as loops, functions, and variables. In addition, the BLAST language provides blocks related to interacting with WoT devices. These WoT-related blocks can be grouped under the three interaction affordances Properties (that can be read and written), Actions (that can be invoked), and Events (that can be observed). Further, the BLAST language includes blocks to send HTTP requests with arbitrary headers, request method and request body. Using HTTP requests, one can load RDF graphs from a URI and execute SPARQL queries, which allows for the interaction with Read-Write Linked Data APIs or other REST APIs. Finally, if the available blocks are not sufficient to implement

[2] https://developers.google.com/blockly/
[3] https://nodered.org/
[4] https://appinventor.mit.edu/

a desired functionality, there is a block in which native JavaScript code can be entered, and thus almost everything that is possible in JavaScript is also possible in BLAST.

For an example for a BLAST program see figure 1. The program involves a LED light with a BLE interface and a small battery-powered BLE beacon. The BLE controller (the "BLE central") is implicitly used when reading the signal-strength property. The functionality of these devices ("Things") can be accessed via reading and writing properties. The LED light, for example, has a property "colour" that can be written. To enable an event abstraction, the program includes a block that defines a state with a condition on the signal strength of the BLE beacon. On entering or exiting the defined state, the event blocks, which can contain arbitrary blocks, are triggered.

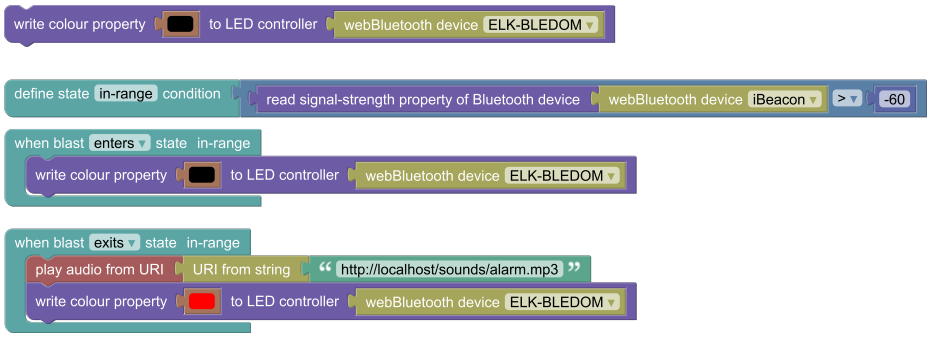

Fig. 1. Example BLAST program that implements a geofencing scenario: when the beacon gets close, the light is turned off; when the beacon leaves the close range, the light is turned red.

3 BLAST Editor and Execution Environment

We have implemented an editor and an execution environment for BLAST programs that runs in a modern web browser.

uses Web Bluetooth to communicate with IoT devices, custom control and command blocks, and generates JavaScript code that can be used by any JavaScript interpreter. To interact with the IoT devices, BLAST requires custom drivers for each device, which represent an extended form of the WoT TD and contain the metadata and interaction affordances as well as the JavaScript code needed for communication.

Supported Devices and Services

To demonstrate how BLAST can interact with WoT devices, we provide a Web of Things abstraction for devices communicating with Bluetooth Low Energy and USB Human Interface Devices (HID). We also provide access to

online APIs related to speech input and output and services like playing audio from a URI or invoking SPARQL queries on an online ressource.

Device	Communication	WoT interactions
Eddystone Devices	BLE GATT	read/write Eddystone properties
HuskyLens[5]	BLE GATT	read sensor properties, invoke forget action
RGB LED Controller[6]	BLE GATT	write color properties
RuuviTag[7]	BLE GAP	read sensor properties, read battery property, read/write txPower property, read movement counter propert,y read measurement sequence counter property
StreamDeck[8]	USB HID	write display properties, subscribe to button push events
Tulogic BlinkStick[9]	USB HID	write color properties
Nintendo JoyCon[10]	USB HID & BLE GATT	read sensor properties, subscribe to button push events
Xiaomi Thermometer[11]	BLE GAP	read sensor properties

Service	WoT interactions
Camera	invoke capture image action
Audio	invoke play audio action
WebSpeech API	invoke text to speech action, invoke speech to text action
HTTP Requests	invoke send HTTP request action
SPARQL Query	invoke execute SPARQL query action
SOLID	invoke upload to solid container action

4 Conclusion

Our block-based language simplifies the interaction with IoT devices, digital twins or other web resources without the need for a deeper understanding of different programming languages or network protocols such as Bluetooth or HTTP. As IoT devices become increasingly relevant in the industrial environment, investigating how the interaction between humans and the IoT devices can be seamlessly implemented. We have explored the use of block-based programming environment to interact with devices. We have also implemented a more complex logistics scenario involving an automated picking line to prepare orders.

We are currently working on a way to execute the created BLAST program independently of browsers. To this end, we are developing an execution environment that can load BLAST programs, convert them to JavaScript, and execute them as a server.

References

1. Bottoni, P., Ceriani, M.: Using blocks to get more blocks: Exploring linked data through integration of queries and result sets in block programming. In: 2015

IEEE Blocks and Beyond Workshop (Blocks and Beyond). pp. 99–101 (2015). https://doi.org/10.1109/BLOCKS.2015.7369012
2. Cimino, C., Negri, E., Fumagalli, L.: Review of digital twin applications in manufacturing. Computers in Industry **113**, 103130 (2019). https://doi.org/10.1016/j.compind.2019.103130
3. Ghazal, M., Haneefa, F., Ali, S., Al Khalil, Y., Sweleh, A.: A framework for teaching robotic control using a novel visual programming language. In: 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS). pp. 1–4. IEEE (2016)
4. Kelleher, C., Pausch, R.: Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. **37**(2), 83–137 (jun 2005). https://doi.org/10.1145/1089733.1089734, https://doi.org/10.1145/1089733.1089734
5. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The scratch programming language and environment. ACM Transactions on Computing Education (TOCE) **10**(4), 1–15 (2010)
6. Pasternak, E., Fenichel, R., Marshall, A.N.: Tips for creating a block language with blockly. In: 2017 IEEE Blocks and Beyond Workshop (B B). pp. 21–24 (2017). https://doi.org/10.1109/BLOCKS.2017.8120404
7. Patton, E.W., Woensel, W.V., Seneviratne, O., Loseto, G., Scioscia, F., Kagal, L.: The punya platform: Building mobile research apps with linked data and semantic features. In: International Semantic Web Conference. pp. 563–579. Springer (2021)
8. Tomlein, M., Grønbæk, K.: A visual programming approach based on domain ontologies for configuring industrial iot installations. In: Proceedings of the seventh international conference on the internet of things. pp. 1–9 (2017)
9. Weintrop, D.: Block-based programming in computer science education. Commun. ACM **62**(8), 22–25 (jul 2019). https://doi.org/10.1145/3341221
10. Weintrop, D., Shepherd, D.C., Francis, P., Franklin, D.: Blockly goes to work: Block-based programming for industrial robots. In: 2017 IEEE Blocks and Beyond Workshop (B B). pp. 29–36 (2017). https://doi.org/10.1109/BLOCKS.2017.8120406