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. An open research challenge in the deployment of IoT devices is the interaction with humans. We introduce a block-based visual programming language and execution environment called BLAST. BLAST allows for visually creating programs involving devices with a Web of Things interface. We demonstrate that BLAST can be used to create programs that interact with a variety of devices in a geofencing scenario.

Keywords: Block-based Programming · Web of Things.

1 Introduction

IoT devices form the basis for a variety of applications, such as condition monitoring, energy consumption analysis, simulation and optimisation [2]. However, there are still some research challenges that need to be overcome. Interaction capabilities refer to the fact that devices will not be involved in any important decisions without humans. In this paper, we focus on the challenge of humans interacting with devices, a challenge that needs to be resolved to achieve a widespread deployment of IoT devices in industry [5]. Thus, humans should find it easy to interact with devices and the devices should be able to interact smoothly with humans.

We assume that the interoperability challenge, that is, a devices' ability to capture a multi-stage product lifecycle, which requires collecting and integrating data from multiple devices and sources [11], has been addressed by the use of the Web of Things abstraction.

We present BLAST (Block Applications for Things), an easy-to-use interaction interface for humans with devices following the Web of Things interface based on Google's Blockly[1]. Blockly is a pure JavaScript library for creating block-based languages and editors. BLAST implements the Web of Things abstraction to devices with Bluetooth Low Energy (BLE) and USB Human Interface Devices (HID) interface and to various browser APIs that allow for accessing HTTP APIs. These devices and APIs are supported by modern web browsers. The Web of Thing abstraction involves Things and their Properties that can be read and written, their Actions that can be invoked and their Events that can be observed.

[1] https://developers.google.com/blockly/

The currently best known tool for programming IoT devices using visual programming is Node-RED[2]. In contrast to BLAST, which focuses on control flow constructs, Node-RED's programming paradigm is based on dataflows. The use cases 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 [10], an Android app development system based on the MIT App Inventor[3]. 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 only and communicates with the device's internal hardware, in addition to working as a LDP-CoAP client to publish and access data on CoAP servers. BLAST, on the other hand, runs in modern web browsers and supports devices based on a Web of Things abstraction.

We first overview block-based visual programming approaches, next introduce BLAST with a scenario involving multiple devices, list the currently supported devices and services and conclude with a summary and outlook.

2 Block-based Visual Programming

Visual programming refers to any system that allows the user to specify a program in a two (or more) dimensional fashion [8]. Conventional textual languages are not considered two dimensional since the compiler or interpreter processes it as a long, one-dimensional stream. 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 [13] [7] [4] [6].

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 [14] [3] [12] or executing SPARQL queries [1], both areas which normally require advanced programming skills.

3 BLAST Language and Editor

Due to the success of block-based languages in different challenging areas, we have developed our own block language. Our block-based language simplifies the interaction with IoT devices, digital twins or other web resources without

[2] https://nodered.org/
[3] https://appinventor.mit.edu/

the need for a deeper understanding of different programming languages or network protocols such as Bluetooth or HTTP. The browser-based BLAST 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.

When choosing the vocabulary for our block-based language BLAST, we resorted to a mixture of natural language and computer language [9]. This means that 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". For an example for a BLAST program see figure 1. We have also implemented a more complex logistics scenario involving an automated picking line to prepare orders.

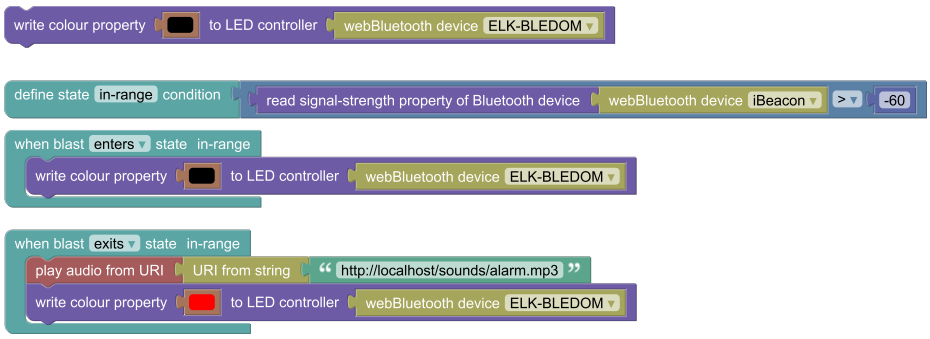

Fig. 1. Example BLAST program that implements a geofencing scenario: when the BLE beacon gets close, the LED light is turned off; when the BLE beacon leaves the close range, the LED light is turned red.

BLAST includes standards from other programming languages such as loops, functions, and variables, but also WoT and IoT related blocks that can be grouped under the three interaction affordances Properties, Actions, and Events. Properties of connected IoT devices can be read and written, actions refer to long-running processes that can be initiated and events are asynchronous and based on the "Event Conditon Action" (ECA) rule. If an event occurs, the condition is checked and if necessary the corresponding action is executed. Furthermore, it is possible to send HTTP requests with arbitrary header and body or load knowledge graphs from a URI and execute SPARQL queries on the records, which allows interaction with Read-Write Linked Data APIs or other REST APIs. If the available blocks are not sufficient to implement a desired functionality, there is a block in which native JavaScript code can be entered. This block makes almost everything that is possible in JavaScript also possible in BLAST.

4 BLAST Execution Environment

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

5 Conclusion

As IoT devices become more and more relevant in the industrial environment, it is essential to investigate how the interaction between humans and the IoT devices can be seamlessly implemented. We have explored the use of an easy-to-use block-based programming environment to interact with devices. 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. Kuehner, K.J., Scheer, R., Strassburger, S.: Digital twin: Finding common ground – a meta-review. Procedia CIRP **104**, 1227–1232 (2021). https://doi.org/10.1016/j.procir.2021.11.206, 54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0
6. Lye, S.Y., Koh, J.H.L.: Review on teaching and learning of computational thinking through programming: What is next for k-12? Computers in Human Behavior **41**, 51–61 (2014)
7. 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)
8. Myers, B.A.: Taxonomies of visual programming and program visualization. Journal of Visual Languages & Computing **1**(1), 97–123 (1990)
9. 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
10. 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)
11. Semeraro, C., Lezoche, M., Panetto, H., Dassisti, M.: Digital twin paradigm: A systematic literature review. Computers in Industry **130**, 103469 (2021). https://doi.org/10.1016/j.compind.2021.103469
12. 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)
13. Weintrop, D.: Block-based programming in computer science education. Commun. ACM **62**(8), 22–25 (jul 2019). https://doi.org/10.1145/3341221
14. 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