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-Universität Erlangen-Nürnberg, Nürnberg, Germany

Abstract. We introduce a block-based visual programming language called BLAST for 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 · Graphical Programming.

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 [3]. The Web of Things (WoT)[1] aims at providing a set of standardised technologies to help simplify creating 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 ease 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 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 [5] [6] [10].

The block-based approach can be used not only for learning programming concepts, but also for more complex problems. Block languages have been successfully used in constructing SPARQL queries [2] or programming industrial robots [9] [11], both areas which normally require advanced programming skills.

We present BLAST, a visual programming environment that works with devices following the Web of Things interface. BLAST programs can be created

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

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 into 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]. Node-RED's programming paradigm is based on dataflows; in contrast, BLAST focuses on control flow constructs. 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 similar to BLAST is Punya [8], an Android app development system based on the MIT App Inventor[4]. Like BLAST, Punya offers a block-based programming environment for writing programs that can include SPARQL queries and access online services. Punya focuses on creating applications for Android devices that can access the mobile phone's internal hardware. Punya also acts as a client to publish and access data on servers supporting the Constrained Application Protocol (CoAP). BLAST programs, on the other hand, run in modern web browsers and support devices based on a WoT 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 [7]. 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. The BLAST language also includes blocks to 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 a desired functionality, there is a block in which native JavaScript

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

code can be entered, and thus almost everything that is possible in JavaScript is also possible in BLAST.

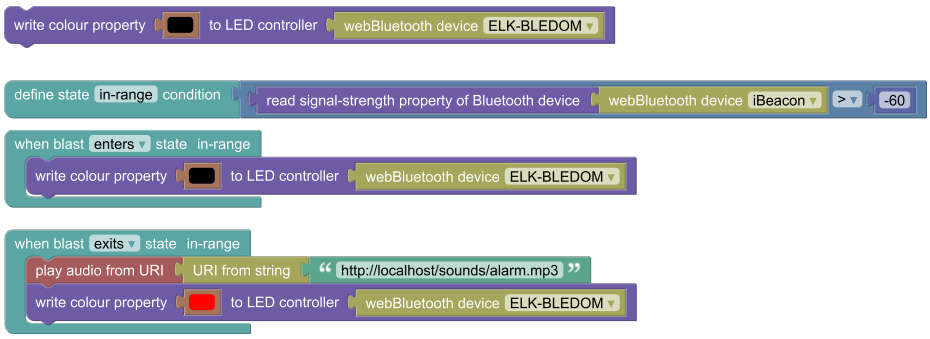

Fig. 1. Example BLAST program that implements a geofencing scenario: when the beacon gets close, the light is turned off; when the beacon leaves the close range, a sound is played and the light is turned red.

For an example of 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 itself can contain arbitrary blocks, are triggered.

3 BLAST Editor and Execution Environment

We have implemented an editor and an execution environment for BLAST programs that run in modern web browsers.

The editor uses the Blockly library and provides custom control and command blocks that make up the BLAST language. The custom blocks and the BLAST environment follow all the important design principles introduced by Fraser [4], such as a clear border style or the avoidance of transitive connections. Based on the user's arrangement of these blocks, the editor generates JavaScript code that can then be executed in the browser. The system builds on the Web Bluetooth API[5] and the WebHID API[6] to interact with devices. BLAST uses the Web of Things abstraction as interface; we have implemented the WoT interface for eight devices that use the BLE GAP and GATT profiles as well as the

[5] https://webbluetoothcg.github.io/web-bluetooth/
[6] https://wicg.github.io/webhid/

USB and Bluetooth HID interface. The WoT interface contains the metadata and interaction affordances as well as the JavaScript code needed for communication. In addition, the system includes access to the camera and sound output via browser APIs using the WoT interface. Further, the system includes access to the Web Speech API[7]. The functionality to issue HTTP requests is based on the Fetch API[8], Social Linked Data (Solid) authentication is based on the Inrupt client libraries[9], and SPARQL query support is based on µRDF.js[10]. Next to the presented small scenario around geofences, we have also developed BLAST programs to accomplish five use cases[11] with varying complexity. Each use case introduces a different challenge that requires the use of programming patterns in combination with the built-in blocks for properties, actions, events, requests, and queries.

4 Usability Evaluation

There is a large number of standardized questionnaires in the scientific literature that allow test users to evaluate the usability of a software product. We use the System Usability Scale (SUS) questionnaire to evaluate the user-friendliness of BLAST. According to Assila et al. [1], SUS provides reliable results over different sample sizes and converges fast to the correct conclusion.

We conducted an evaluation with nine participants. None of the participants had background knowledge regarding the Web of Things, RDF, or SPARQL. Participants were first asked to solve four tasks using BLAST, these included a simple "Hello World!" program, periodic polling of RSSI data by using a loop, reading data from a connected IoT device, and as a final task, reading data and processing it using logical operators. After the successful completion of these tasks, the participants filled out the SUS questionnaires. The achieved SUS score across all questionnaires is 81.4 out of 100 possible points, which corresponds to good usability.

It should be noted that the survey was conducted with an older version of BLAST. However, the user interface has changed only minimally since then, which is why the results are still relevant.

5 Conclusion

We have introduced BLAST, a block-based language that 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. We are currently working on an execution environment that runs on Node.js as

[7] https://wicg.github.io/speech-api/
[8] https://fetch.spec.whatwg.org/
[9] https://docs.inrupt.com/developer-tools/javascript/client-libraries/
[10] https://github.com/vcharpenay/uRDF.js/
[11] TODO: URI to videos

a way to execute BLAST programs independently of browsers, which impose restrictions on accessing devices to provide security and privacy protection.

References

1. Assila, A., Ezzedine, H., et al.: Standardized usability questionnaires: Features and quality focus. Electronic Journal of Computer Science and Information Technology **6**(1) (2016)
2. Bottoni, P., Ceriani, M.: Using blocks to get more blocks: Exploring linked data through integration of queries and result sets in block programming. In: IEEE Blocks and Beyond Workshop. pp. 99–101 (2015)
3. Cimino, C., Negri, E., Fumagalli, L.: Review of digital twin applications in manufacturing. Computers in Industry **113**, 103130 (2019)
4. Fraser, N.: Ten things we've learned from blockly. In: 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond). pp. 49–50 (2015). https://doi.org/10.1109/BLOCKS.2015.7369000
5. Kelleher, C., Pausch, R.: Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys **37**(2), 83–137 (2005)
6. 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)
7. Pasternak, E., Fenichel, R., Marshall, A.N.: Tips for creating a block language with blockly. In: IEEE Blocks and Beyond Workshop. pp. 21–24 (2017)
8. 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)
9. Tomlein, M., Grønbæk, K.: A visual programming approach based on domain ontologies for configuring industrial iot installations. In: Seventh International Conference on the Internet of Things. pp. 1–9 (2017)
10. WEINTROP, D.: Block-based programming in computer science education. COMMUNICATIONS OF THE ACM **62**(8), 22–25 (2019)
11. Weintrop, D., Shepherd, D.C., Francis, P., Franklin, D.: Blockly goes to work: Block-based programming for industrial robots. In: IEEE Blocks and Beyond Workshop. pp. 29–36 (2017)