A Javascript Web Framework for Rapid Development of Applications in IoT Systems for eHealth

Jonas Wåhslén
Department of Biomedical Engineering
and Health Systems, KTH
Stockholm, Sweden
jwa@kth.se

Thomas Lindh
Department of Biomedical Engineering
and Health Systems, KTH
Stockholm, Sweden
thlindh@kth.se

Abstract—Bluetooth Low Energy (BLE) is currently the dominating wireless network solution for eHealth and sports. However, most BLE sensors require dedicated applications with limited development capabilities. This paper presents a method for rapid development of applications in distributed BLE IoT systems for eHealth and sports. The method is implemented as a JavaScript web framework based on HTML5 canvas, WebSocket and Web Bluetooth APIs. This paper demonstrates how the framework can be applied to develop an application for monitoring physical activity and heart rate. The framework enables software and service operators to iteratively create, tune and deploy filter algorithms in distributed BLE IoT systems, without rebooting nodes or restarting programs using dynamic software updating.

Keywords—JavaScript, Web Bluetooth API, Internet of Things, WebSocket, Bluetooth low energy, dynamic software updating, eHealth.

I. INTRODUCTION AND MOTIVATION

According to Gartner, 8.4 billion devices were connected to the Internet in 2017\(^1\). The growth of IoT is one of the driving forces behind this increased number of connected devices. However, currently IoT control, such as OneM2M [1], mainly turns sensors on or off and sometimes sets thresholds. These solutions limit real-time changes and tuning to just basic commands. For more complex changes of local processing on sensors, the approach is often to make firmware over-the-air (OTA) upgrades. The development of sensors with OTA is cumbersome, requires rebooting, and there are security problems with sending binaries as shown in [2]. Moreover, some processing is better suited to being performed by a gateway and not on sensors or back-end servers.

We have identified two problems that need to be addressed: first, a general method for rapid development of applications to apply incremental changes in functionality, and second, a method using dynamic software updates to distribute algorithms OTA in real-time. An application developer or operator can iteratively develop an application, e.g., configure and distribute algorithms in a distributed IoT system. Currently the dominant web browsers have implemented HTML5, which includes new capabilities to visualise, process and transmit data. The hypothesis in this paper is that HTML5 and JavaScript can be applied to develop and distribute algorithms in a Bluetooth Low Energy (BLE) IoT system.

The outline of this paper is as follows. Related works on Bluetooth, HTML5, JavaScript and other dynamic software update solutions are found in Section II. The framework, a use case and functional requirements are described in Section III. Examples of rapid development and distribution of algorithms are presented in Section IV. The results are discussed in Section V and summarised in Section VI.

II. RELATED WORK

A. Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a personal wireless network solution, mainly aimed at short-lived connections, specified by the Bluetooth SIG (Special Interest Group). According to ABI Research, there will be 16 billion Bluetooth devices by 2021\(^2\). The possibilities and challenges for BLE in an IoT environment are well described in [3]. One major problem is the restricted access to the lower level of BLE stacks, limiting the usage of built-in BLE functionality, e.g., synchronisation and network management for application developers. The application layer has been standardised as GATT profiles, many based on the Bluetooth Health Device Profile (HDP) that implements the IEEE 11073-20601 personal health device standard [4]. These GATT profiles have successfully been implemented by applying the ISO standards from sensors to gateways [5]. Methods to monitor and control traffic in BLE networks on the application layer have also been successfully implemented [6]. Currently, BLE sensors are frequently used in sports to measure pulse, position (GPS), activity, cadence, and force. In home automation, BLE sensors are used to control and monitor light, temperature, humidity, noise level, air quality and in surveillance. Moreover, in health, BLE sensors measure among other things, weight, body temperature, blood pressure, sleep quality, activity and UV light. However, a problem with all these BLE sensors is that they have dedicated applications. Even though there are approximately 60 publicly adopted GATT profiles, there still exist many more proprietary profiles designed by the hardware manufacturers of different BLE sensors. There is not currently any generic way to

---

1. www.gartner.com/newsroom/id/3598917
retrieve data from all these sensors. The Physical Web is a solution to interact with physical objects using Bluetooth. An object includes a Bluetooth sensor that uses the Eddystone Bluetooth beacon protocol to broadcast a Uniform Resource Locator (URL). This URL points to a web page with necessary information about the physical object, as described in [7]. Connecting Bluetooth beacons from an HTML page has also been suggested [8]. Bluetooth beacons do not solve the problem of each sensor requiring a dedicated application. However, if a web browser could directly connect to a GATT profile, then the URL from a Bluetooth beacon could point to an HTML page that can connect to the specific BLE sensor. Thus, there would be no need to download a new application.

B. HTML5 and JavaScript

JavaScript was originally created to check HTML forms before sending them to servers. However, already in 2010, the Open Mobile Alliance defined a JavaScript-callable API for dynamic content delivery that could access APIs on the client side [9]. Moreover, there are JavaScript solutions for distribution and development of algorithms for computable heavy environments [10]. Currently JavaScript is an essential part of the Internet, and there are many libraries, plugins, and APIs available. Dynamic software updates of web applications are done routinely by transmitting JavaScript code. HTML5 is the latest version of HTML. It includes new features such as HTML5 canvas and WebSocket. HTML5 canvas can be used to draw graphics on a webpage, without the need for a third party plugin like Flash. WebSocket enables a two-way full duplex communication with a TCP server. Another feature is the Web Bluetooth API\(^1\), still in draft as of April 2018, but supported by 59% of today’s browsers\(^4\). The Web Bluetooth API enables connection to BLE sensors directly from an HTML5 webpage. Combining these three features – HTML5 canvas, WebSocket and Web Bluetooth – it is possible to process, send and visualise data from a connected BLE sensor, directly from a supported HTML5 browser. Since Bluetooth Web API is available for node.js servers, JavaScript on a BLE sensor also works on gateways running node.js. Tenson 2, Raspberry Pi and Intel’s Edison are examples of sensor nodes using JavaScript and JavaScript can also be installed with a minimal footprint, e.g. Duktape can run on platforms with 192 kB flash and 64 kB RAM. These sensors or Duktape would enable distribution of JavaScript algorithm down to sensors.

C. Other Solutions for Dynamic Software Update

This paper focuses on BLE sensors in an IoT environment being the dominant wireless sensor technology. Other alternatives such as Wi-Fi would be possible as seen in [11]. However, there are currently limited numbers of sensors with Wi-Fi capability. This is also the case for ZigBee sensors. Dynamic software updating for IoT systems is well-researched. One example is Contiki [12], which is an operating system for IoT sensors that provides the ability to dynamically update runnable modules during in-run-time. The programming language Erlang [13] is another solution for development and distribution of algorithms in an IoT environment. Erlang provides the possibility to in a fault-tolerant way during run-time dynamically update code. Dynamic TinyOS [14] is an extension of the well-known module-based operating system for embedded hardware that enables software changes without interrupting the sensor node operation. Contiki, Erlang and TinyOS are attractive solutions for dynamic software update. However, currently, none of the above-mentioned operating systems have support for Bluetooth GATT profiles. Hence, this paper focuses on a JavaScript solution.

Network communication for IoT is also well explored. Constrained Application Protocol (CoAP), and Message Queuing Telemetry Transport (MQTT) are the two dominant transport protocols. MQTT is a publish subscribe message protocol that requires a message broker. Interoperability between MQTT and BLE GATT sensors is supported by MQTT bridges\(^5\). CoAP is a RESTful application protocol that transmits data over UDP. CoAP is suitable for sensors with limited capacity. For BLE GATT sensors, a gateway is required to bridge sensors to an IP network. It is essential to support both CoAP and MQTT for deployed IoT systems.

III. METHOD AND FRAMEWORK

The method used to verify the hypothesis was to build a framework and evaluate the feasibility of using HTML5 in combination with JavaScript for the development and distribution of filter algorithms. By creating a framework that takes advantage of the new features introduced in HTML5, the hypothesis could also be tested against functional requirements. These were identified from Ambient Assisted Living (AAL) use cases related to eHealth.

A. The Framework

The framework description is divided into three parts: an overview of the components, functions and workflow for rapid development of algorithms as an example of application, and finally an explanation of the security features.

1) Framework Components

Our framework (Fig. 1) consists of three components, an HTML5 gateway, and a node.js server and an HTML5 client front-end (below 800 total lines of code)\(^6\). The node.js server acts as a web server. It also handles WebSocket connections to the HTML5 gateway and the client front-end.

\(1\) webbluetoothcgh.github.io
\(2\) caniuse.com/#feat=web-bluetooth
\(3\) github.com/shmuelzon/ble2mqtt
\(4\) github.com/JonasWahlsen/BLWebFramework

\(5\) github.com/shmuelzon/ble2mqtt
\(6\) github.com/JonasWahlsen/BLWebFramework
The HTML5 gateway connects and retrieves data from one or more BLE sensors utilising the Bluetooth Web API. Data from sensors are then processed locally on the gateway, using JavaScript and visualised continuously with HTML5 canvas. The testbed hardware consists of a MacBook (early 2015) running the entire framework, micro:bit sensors and a pulse oximeter\(^1\). The left part of Fig. 2 shows the gateway running on a Chrome web browser. Before the sensor is connected, the user receives a pop-up request to allow communication with the BLE sensors. In Fig. 2, a micro:bit sensor samples and transmits accelerometer data. The black line in the top left part of Fig. 2 shows raw data from an accelerometer sensor. The red line shows the raw data with a smoothing EWMA (Exponential Weighted Moving Average) filter. The programming code for the EWMA filter algorithm is displayed in the text field at the bottom left of Fig. 2.

2) Function and Workflow

In order to enable rapid development in the framework, it is possible to write or change the filter algorithm in the text field. The filter algorithm is saved as a variable that can change without restarting the program. For each packet from a BLE sensor, a function that includes the filter algorithm is executed. For security reasons, a small set of selectors limits the coding of filter algorithms. These selectors were identified by analysis of conventional digital filter for streaming data. These selectors consist of seven variables (Table I). The immutable variables are read-only. The variables that are mutable, that is, possible to modify in a filter algorithm, are the `FilterValue` and the `infoText`. `DataValue` contains the last sample received from the BLE sensor. The time is updated with the `updateTime()` function. The selectors with the prefix `Old` hold the previous time, filter and sample values. With these selectors, it is feasible to construct an extended number of filter algorithms. This limitation of a small set of selectors only affects the ability to create new variables or change variables on the page other than the selectors. A more detailed description of rapid development of filter algorithms is found in Section IV.A

![Fig. 2](image-url)

**Fig. 2.** A BLE sensor connected to an HTML5 gateway on the left (running in Chrome) and a client front-end on the right (running in Safari).

<table>
<thead>
<tr>
<th>Table I. Selectors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Immutable</strong></td>
</tr>
<tr>
<td><code>DataValue</code></td>
</tr>
<tr>
<td><code>OldDataValue</code></td>
</tr>
<tr>
<td><code>OldFilterValue</code></td>
</tr>
<tr>
<td><code>Time</code></td>
</tr>
<tr>
<td><code>OldTime</code></td>
</tr>
<tr>
<td><strong>Mutable</strong></td>
</tr>
<tr>
<td><code>FilterValue</code></td>
</tr>
<tr>
<td><code>infoTextmat</code></td>
</tr>
<tr>
<td><code>(updateTime())</code></td>
</tr>
</tbody>
</table>

Data from the sensor is transmitted over a WebSocket from the gateway to the node.js server. The server then forwards all data to the HTML5 client front-end. Initially, data from the gateway is a stream of samples from the connected BLE sensor. The right part of Fig. 2 shows a front-end, running in a Safari browser, receiving data from a node.js server. The filter algorithm runs locally on the front-end when pressing the Program filter button. The sensor can also be controlled remotely by activating the Send button, which transmits the filter algorithm to the gateway. The gateway will consequently run the filter algorithm and start transmitting filtered data. The ToggleRaw button starts or stops the gateway sending samples from the BLE sensor. The development environment for the framework also supports recording samples (Record button) that can be saved to file (Save button). Data can also be loaded from a file (Choose file button) to evaluate different filter algorithms offline. Loaded data will loop indefinitely, making it easy to compare different filter algorithms. Every data transmission between front-end, server and gateway include an identifying byte. This byte identifies a request to stop or start sending raw data, update filter algorithms (ex-

---

\(^1\) microbit.org

\(^2\) pulsesensor.com
amply code is in Fig. 3), and so forth. Currently, there are four different identifying bytes (Table II). However, this can be expanded to cover more choices, e.g., updating the filter algorithm on the sensor node.

### TABLE II. IDENTIFYING BYTES

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stop sending raw data</td>
</tr>
<tr>
<td>2</td>
<td>Start sending raw data</td>
</tr>
<tr>
<td>3</td>
<td>Update filter code algorithms on server</td>
</tr>
<tr>
<td>4</td>
<td>Update filter code algorithms on GW</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
</tr>
</tbody>
</table>

3) Security

The framework has two security mechanisms. The first option is to use HTTPS and WSS to secure the communication with TLS. Second, the framework protects against malicious or poorly written JavaScript code by sandboxing execution of filter algorithms. This can be done in different ways [15].

Implemented in the framework is a general solution. The execution of filter algorithms is wrapped inside a function that ensures that only the filter algorithm affect the selectors.

```
// Exponentially Weighted Moving Average (EWMA)
alfa = 0.9
FilterValue = FilterValue*alfa+DataValue*(1-alfa)

// Adaptive Sampling
diff = 100
if(Math.abs(OldDataValue - DataValue) > diff){
    FilterValue = DataValue
    updateTime()
}

// Zero-Crossing Rate
zero = 100
if((OldDataValue < zero && DataValue > zero) ||
   (OldDataValue > zero && DataValue < zero)){
    FilterValue = DataValue
    updateTime()
}

// Threshold
threshold = 100
if(DataValue > threshold){
    FilterValue = DataValue
    updateTime()
}

// Simple Step counter
alfa = 0.65, threshold = 17
FilterValue = FilterValue*alfa+DataValue*(1-alfa)
if(FilterValue > threshold){
    infoText = infoText + 2
}

// Heart rate
alfa = 0.3, beta = 0.95
if(OldDataValue > DataValue){
    FilterValue = FilterValue*alfa+DataValue*(1-alfa)
} else{
    FilterValue = DataValue
}
if(OldDataValue < OldDataValue &&
   DataValue > FilterValue){
    updateTime()
    infoText = "Heart rate: " + Math.round(60000/  
        (newTime-oldTime))
}
```

Fig. 3. Examples of filters using selectors.

B. **AAL Use Case Challenges in eHealth**

The functional requirements in Section III.C were derived from the seven representative use cases found in the AAL (Ambient Assisted Living) Use Cases and Integration Profiles [16]. These use cases can be summarised as an eHealth application where an older woman has a mobile unit to monitor her physical activity and heart rate. Data from the mobile unit is transmitted to the local healthcare provider to provide progress feedback and suggest appropriate physical activity. These recommendations motivate and help her to live a healthy life. Moreover, the mobile unit connects to external sensors in the home for behaviour tracking, for example detecting if the person watching TV has forgotten the boiling water on the stove. With a drug dispenser connected to the mobile unit, monitoring and reminding of medicine intake are possible. The system can also detect emergencies. For these eHealth-related use cases, an application developer is responsible for configuring the entire system from the external sensors via the mobile unit to the local healthcare provider. A challenge for the application developer is the wide diversity of sensors and mobile units that are available. The triaxial accelerometer on a micro:bit and an attached pulse oximeter measured physical and heart activity in our testbed. However, that is only one of several different types of hardware equipment to track a person’s physical activity and heart rate. For an application developer, manual configuration to connect new hardware will be needed. Consequently, a solution should support adaptation to hardware diversity, configuration and tuning without the need for extensive reprogramming and rebooting of the sensor device. The desired solution translates well to other eHealth and sports applications.

C. **Functional Requirements**

Physicians, athletes and other end-users are not interested in raw data from a monitoring system. Instead, users want processed data, e.g., physical activity; heart rate; cadence; number of steps or how long time a patient has been sitting, standing or walking during the day. Thus, there is a need for an algorithm-based translation from sampled to processed data. Application developers are required to create and tune a suitable algorithm to obtain relevant information when introducing new sensors. Development directly on the sensor or from the gateway is required for the initial setup and iterative development with the user e.g., elite athletes. For remote configuration or fine-tuning it is essential to be able to continue the development from other parts of the system, e.g., to calibrate the accelerometer-based measurement of physical activity from the front-end computer.

Four functional requirements were identified.

- **FR01** – Retrieve and visualise data from a BLE sensor on a gateway and front-end.
- **FR02** – Configure filter algorithms using incremental development.
- **FR03** – Distribute sensor data from gateway to server, front-end or other intermediate nodes.
- **FR04** – Support dynamic software update to distribute filter algorithms from a client front end to other nodes in an IoT system.
IV. RESULT

This section demonstrates how the framework can be applied to monitor physical activity and heart rate based on the AAL use case in Section III.B. A tool for tuning filter algorithms is presented as an example of rapid application development. The result is divided into two subsections: rapid development and distribution of algorithms, and a throughput comparison between a native and a Web Bluetooth solution.

A. Rapid Development and Distribution of Algorithms

The result of the rapid development and distribution of algorithms is divided into three parts: examples of iterative development of filter algorithms, how to use the framework for rapid development, and finally connection of new sensors to the web gateway.

1) Iterative Development of Filter Algorithms

Fig. 3 shows examples of four common digital filters for streaming data: Exponentially Weighted Moving Average (EWMA), adaptive sampling, zero-crossing rate, and thresholds. Each filter in Fig. 3 was created with the selectors described in Section III.A. Combinations of these filters are also feasible for construction, as exemplified in the simple step counter and the heart rate algorithms shown at the bottom of Fig. 3. The step counter and heart rate algorithms are two examples of filter algorithms developed in an interactive process built on the use case in Section III.B. The first was a step counter, where the sensor was placed in the left jeans pocket. Fig. 4 shows the raw data (black line) and filtered data (red line) from the combined acceleration vector of the triaxial accelerometer. The algorithm was developed in two phases. First, after several iterations, a simple EWMA filter was selected. In the second phase, a threshold value was adjusted to detect a step with the left foot. The speed of the iterations to construct and tune the filter algorithm was increased by the framework. The result of the filter algorithm code in the text field was visualized immediately in the HTML5 canvas on the gateway or front-end, by pressing the Program sensor button. Fig. 3 shows the final filter for the simple step counter. The number of steps is indicated in Fig. 4 (top left), using the selector infoText.

![Simple step counter based on accelerometer data when walking.](image)

The second example of iterative development is the detection of heart rate, which was retrieved from a pulse oximeter attached to the micro:bit. Here, an iterative approach resulted in an EWMA filter that changed the weight of the moving average depending on whether the value of the received sample was higher or lower than the previous sample. Subsequently, a zero-crossing rate filter was applied to find the intersection crosses between the raw and filter curve to detect a heartbeat. After identifying a heartbeat, the heart rate was calculated and visualized in infoText as shown in Fig. 5 (top left). The figure shows the raw (in black) and filtered (in red) data from the pulse oximeter. The final filter algorithm to detect the heart rate is seen in Fig. 3.

![Heart rate data from a pulse oximeter. The black line is the raw accelerometer data, and the red line is the filtered data.](image)

2) The Framework

The framework presented in this paper is a tool for rapid application development to be used by application developers. The web gateway and front-end allow application developers to construct and tune filter algorithms when new sensors are introduced. The filter algorithms exemplified in the step counter and the heart rate monitor depend on noise, sampling frequency and the resolution of incoming samples. These characteristics may change when introducing new sensors, i.e., requiring construction and tuning of new filter algorithms by an application developer. This step by step development can be done remotely over-the-air in real-time with the framework without the need to reboot the sensor node or restart any part of the system. Raw data can be recorded and saved to a file. This saved file lets developers try different filter algorithms on the same data or process the data elsewhere. Measurements of heart rate and physical activity are frequently used in an AAL scenario. The filter algorithms in the previous subsection demonstrate a proof-of-concept using the framework to develop AAL applications that monitor physical activity and heart rate as described in Section III.B. Moreover, an application developer can easily integrate other external sensors in the home with the framework.

3) Connecting New BLE Sensors to the Web Gateway

The web gateway is a central part of the framework and is two-thirds of the total code size. The web gateway also solves the problem of connecting to new BLE sensors requiring new applications to be downloaded. A sensor can act as a Bluetooth beacon broadcasting an URL that refers to the web gateway that can be configured explicitly for the specific BLE sensor. This solution enables an end user to connect to a BLE sensor directly from a web browser and transmit data.

B. Throughput Comparison Between Native and Web Bluetooth Solutions

A throughput test was conducted to compare a web gateway implementing the Bluetooth Web API with an application developed in Swift running natively in OS X. Fig. 6 shows the throughput for different packet frequencies. The payload of
each packet was 12 bytes, as this is equal to the sample size transmitted from a triaxial accelerometer. The blue line represents the theoretical maximum. The lines below show the different results for one or two sensors, either connected to a browser with Bluetooth Web API or a native application developed in Swift. The test also compared throughput using notifications and indications. The difference is that indications require an acknowledgement for every packet sent. The conclusion was that at a higher transmission rate a native application outperforms the HTML5 Web Bluetooth application. The overhead of running a JavaScript virtual machine explains the difference in performance. The second observation was the higher throughput when multiple sensors were connected. The higher throughput was because the request interval for every sensor to transmit data decreases from 18 ms to 14 ms. Connecting more than two sensors did not decrease the interval time further.

![Graph showing throughput comparison of BLE sensors sending 12 byte packets.]

**Fig. 6.** Throughput comparison of BLE sensors sending 12 byte packets.

V. DISCUSSION

In sport and eHealth applications there is often a need for multiple iterations during construction and tuning of filters to obtain the correct setup. This type of development can be done over-the-air in real-time without rebooting the nodes or restarting programs. Furthermore, the framework also works offline with stored sensor data. In education, it supplies students with a tool to construct and understand how digital filters work. This framework is usable in a deployed BLE IoT solution since the Web Bluetooth API also works on a node.js. A node.js gateway can also act as a bridge to work in combination with MQTT and CoAP. However, currently, the framework is limited to a web gateway. Our solution does not have the same performance as a native solution as shown in Section IV.B. However, it is a generic solution that is platform independent, and the performance differences are at a throughput higher than customarily used for BLE sensors. Visible differences in throughput were observed at 100 Hz sampling and above with notifications (Fig. 6). Development of the framework to support a node.js gateway and multiple sensors will be considered in future work.

VI. CONCLUSIONS

This paper presented a JavaScript framework for the rapid development of applications in distributed BLE IoT systems, exemplified with a tool for development of filter algorithms. We have shown that it is possible to combine the HTML5 Canvas, WebSocket and Bluetooth Web API in a framework to iteratively develop, tune and deploy filter algorithms for eHealth and sports applications. The framework can be seen as a vital part of a web-based software system for healthcare services using JavaScript and HTML5 to enable rapid development and dynamic software updating in real-time to test different algorithms dynamically without rebooting the nodes or restarting programs.

ACKNOWLEDGEMENT

Thanks you to Martin Jacobsson for introducing micro:bit and the problems with digital filters.

REFERENCES

[9] OMA, Client-Side Enabler API Requirements, Apr 2010
[16] AAL Joint Program D2, AAL Use Cases and Integration Profiles, Aug 2014