IEIE Transactions on Smart Processing and Computing, vol. 3, no. 6, December 2014
http://dx.doi.org/10.5573/IEIESPC.2014.3.6.393

Design of Lightweight JavaScript Software Platform for the Internet of Things

Wonjun Lee, Yeoul Na, and Seon Wook Kim

Department of Electrical and Computer Engineering, Korea University / Seoul, South Korea
{rriiaa, seon}@korea.ac.kr, rapidsna@gmail.com

***** Corresponding Author: Seon Wook Kim

Received February 20, 2014; Revised April 20, 2014; Accepted August 28, 2014; Published December 31, 2014

** Short Paper*

***Abstract*:** Recently, the internet of things (IoT) has become increasingly attractive in many areas to realize smart worlds. JavaScript has become prevalent for IoT programming because of its familiarity with web programmers. On the other hand, JavaScript does not allow the direct control of IoT hardware due to its language limitation. This paper proposes the IoT software platform for JavaScript programming to resolve the limitation. For proof of concept, the platform based on SpiderMonkey and Raspberry Pi was implemented.

***Keywords*:** SpiderMonkey, Raspberry Pi, JavaScript, IoT, Node.js

1. Introduction

The Internet of Things (IoT) [1] is a new technology that can connect various objects, supporting sensors and communication functions to the Internet. The objects can be any device around us, such as consumer electronics, mobile devices, wearable computers, and various embedded systems.

In addition to the advent of IoT technology, *Node.js* [2] has become increasingly attractive in both academia and industry. *Node.js* is an open source, cross-platform runtime environment for easily constructing both an Internet-based server and client applications [3]. As it is written in JavaScript, which is a very intuitive and interactive programming language, Node.js provides a simple way to create Internet-based applications, and is becoming increasingly prevalent in IoT environments.

Despite the merits, however, there is a significant shortcoming of using *Node.js* for IoT, which is poor accessibility to hardware because of its language limitations, i.e., inability to control hardware directly. For example, most IoT devices contain several sensors that can manipulate analog input signals. To handle these signals, they are supposed to be converted to digital signals through an analog-to-digital converter (ADC). On the other hand, *Node.js* basically does not provide any primitive method to manipulate such hardware modules accessed via I/O ports and memory addresses. In this situation, additional native extensions to *Node.js* are necessary. Of course, Node.js can control hardware through add-on modules available on the NPM registry [4] but the modules only provide simple functions, such as LED ON and OFF.

This paper proposes a light-weight JavaScript software platform for IoT devices, which consists of JavaScript APIs, the JavaScript engine, and native libraries for hardware control. The proposed platform was tested by implementing the prototype system. The proposed IoT platform provides not only the original merits of JavaScript, such as productivity and efficiency, but also accessibility to hardware.

This paper is organized as follows. Section 2 presents the overall architecture of the proposed platform. Section 3 elaborates on how the proposed software platform is implemented in detail. Section 4 discusses future work and reports the conclusions.

2. Software Platform for IOT

This section describes the proposed JavaScript software platform for IoT.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MEST) (No.2011-0020128).

2.1 Overall Architecture

Fig. 1 shows an IoT runtime environment that consists of a hardware device connected to multiple hardware modules, such as I/Os, sensors and networks, and the proposed JavaScript software platform that can execute JavaScript applications. To make it possible for JavaScript applications to manipulate the hardware modules directly, the following two components were newly employed in the proposed software platform: 1) proprietary APIs for IoT and 2) native libraries for hardware control. The proprietary APIs consist of several JavaScript functions to execute the native libraries through the JavaScript engine of the software platform. Some of the APIs are listed and explained briefly in Section 2.2. The native libraries include C/C++ functions for handling several peripherals, GPIOs, timers, etc., which are embedded in the JavaScript engine. When the JavaScript application is executed on the software platform, the JavaScript engine automatically configures all hardware modules to be ready to run. The platform supports the application to identify the type of hardware module, and gather additional information, including the ports and memory addresses of the modules and the permission for handling them. After acquiring the hardware information and permission, the JavaScript application can control the hardware modules through the proprietary APIs. The JavaScript engine will execute its corresponding native library functions.

2.2 Proprietary APIs for IoT

The proprietary APIs for IoT provide several JavaScript functions to help manipulate various hardware modules, such as I/O, sensors, and networks. In this section, only representative functions of the proprietary APIs are explained briefly below, and ***JSBool*** is JavaScript's Boolean type.

- **JSBool PiSetup()** : Opens the master /dev/memory device, GPIO, PWM, and timer. This should be called at the start of program execution.
- **JSBool SPISetup(int CLK, int ADC_PIN)** : CLK is the clock speed and ADC_pin is the ADC input pin. This is a function to open the SPI device, and to set up the clock speed and ADC input pin that is the chip's analog input.
- **JSBool pinmode(int PIN, int STATEMENT)** : Determines the input or output statement of the specified GPIO pin.
- **JSBool dWrite(int PIN, int VALUE)** : Sets an output bit as HIGH or LOW.
- **int dRead(int PIN)** : Reads a value from a given pin, which returns a HIGH or a LOW value.
- **double read_mcp3208_adc(int PIN)** : Reads an analog value from a given ADC input pin.
- **JSBool delay(int TIME)** : Waits for a specified time of milliseconds.
- **double pow(double BASE, double EXPONENT)** : Calculates the exponentiation.
- **JSBool ReadAll()** : Reads all the pins on the board and prints the lists on the screen.

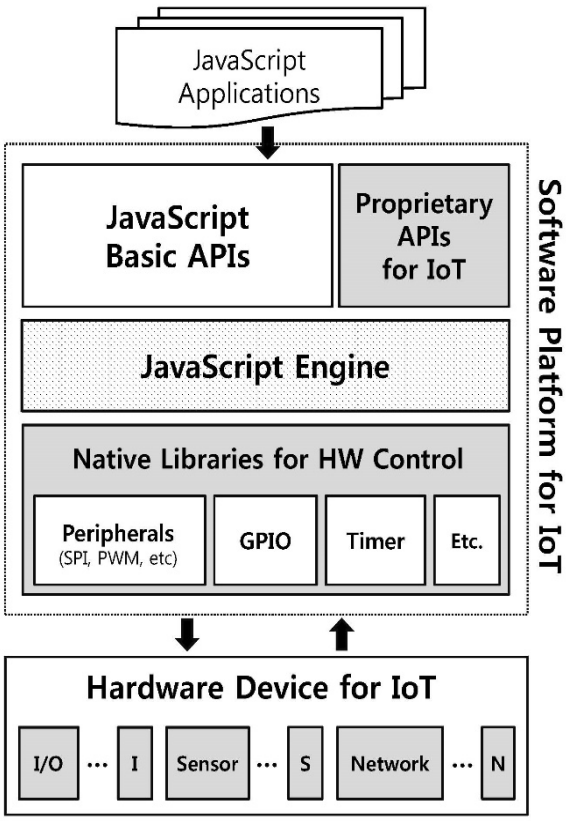

Fig. 1. IoT runtime environment with the proposed JavaScript software platform.

3. Implementation

In this study, an IoT prototype system based on *SpiderMonkey* [5] and *mozjs-24* [6] was implemented. SpiderMonkey is a Mozilla's JavaScript engine that provides only primitive JavaScript data types (e.g. *Number*, *String*, *Array*, and *Object*), basic methods (e.g. *Array.push*) and basic keywords (e.g. *if*, *while*, and *for*). Accordingly, the engine itself is very light and expandable so that users can append an arbitrary function depending on the target of the JavaScript engine. As the engine automatically compiles and executes a code, it is very easy to modify and debug a code.

The target device is the Raspberry Pi [7] with raspbian, a customized debian OS. In addition, two LEDs and several sensors, such as temperature, humidity, distance, and illuminance are employed in the prototype system.

3.1 Board Design

The Raspberry Pi can communicate with modules, such as sensors through the *SPI* bus protocol. The SPI bus specifies three signals, MOSI (Master Out Slave In), MISO (Master In Slave Out), and SCK (Serial Clock), which can be connected to the GPIO pins in the Raspberry Pi. On the other hand, the SPI bus specifies digital signals and the Raspberry Pi does not have a GPIO port for analog signals. Therefore, to read the analog values from sensors and convert them to digital signals, an ADC chip was

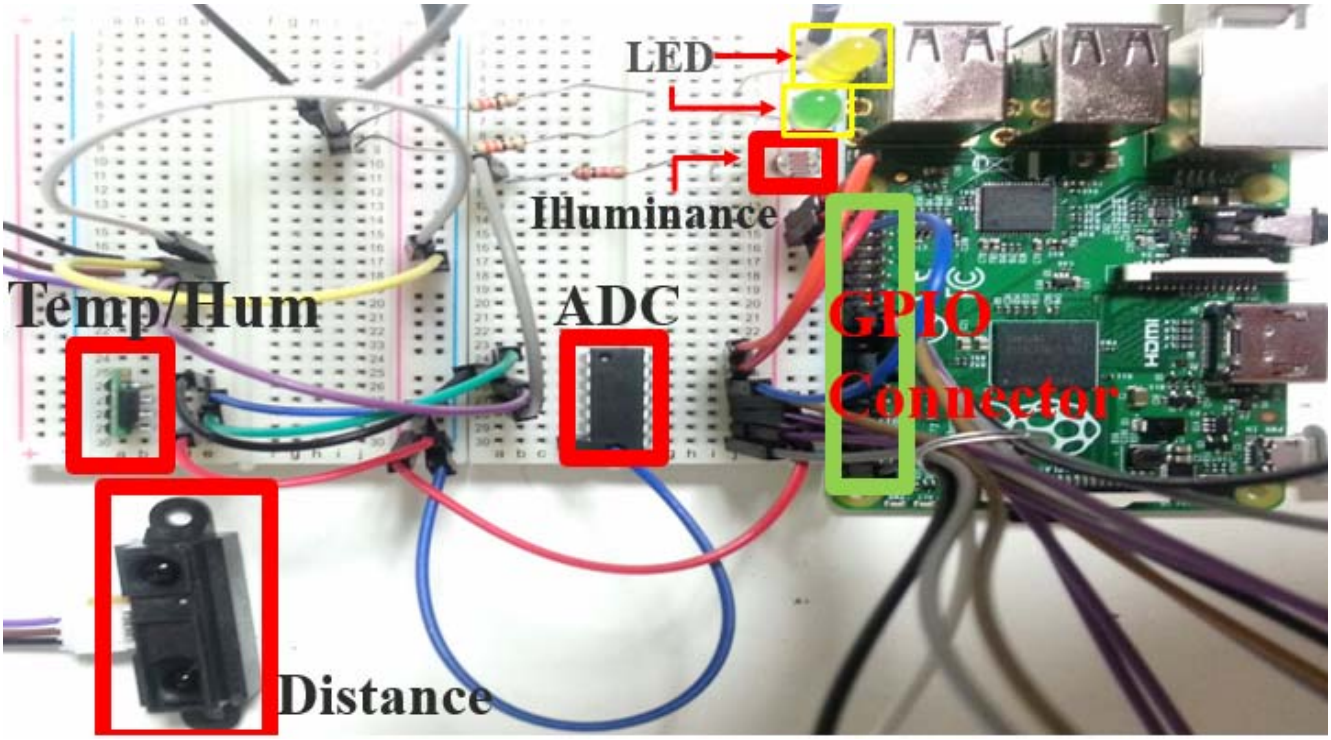

Fig. 2. Raspberry Pi board for the experiment.

installed in the prototype system.

Once MISO, MOSI and SCLK are connected to the ADC chip, the sensors are connected to each ADC input pin of the ADC chip. The Raspberry Pi reads the value of the corresponding ADC input pin through the MISO and makes the ADC chip read a value from its ADC input pin through MOSI pin. As shown in Fig. 2, the sensors used in this paper are temperature, humidity, distance, and illuminance sensors. In addition, two LEDs are augmented for testing purposes.

3.2 Test Code

Fig. 3 presents example test code. In (1) the pin numbers of Raspberry Pi and SPI are initialized to read a value from the ADC chip. In addition, the clock speed of the Raspberry Pi is set in the SPISetup function. In (2) GPIO or sensors are set to determine the in/output statement of the pin numbers. In the prototype system, two LED are connected to pin numbers 1 and 2 to test **pinmode**() and **dWrite**() functions. The functions, such as **pinmode** (pin, statement), **dWrite** (pin, value), **dRead** (pin, value) and **ReadAll** () are used in the shell mode interpreter of SpiderMonkey. These functions that can be used in shell mode are useful for testing GPIO and checking the *GPIO pin's information* [7] simply. Finally, as shown in (3), the **read_mcp3208_adc**(Distance) function can read the ADC value from a distance sensor.

```
function main()
{
  ......
  (1) if(!PiSetup())
      {
          print("Unable to start device initialization₩n");
          return false;
      }
      if(!SPISetup(0,CLOCK_1MHz,ADC_Output)
      {
          print("Unable to start SPI₩n);
      }
  (2) pinmode(1, OUTPUT);
      pinmode(2, OUTPUT);
      ......
      while(1)
      {
        ......
  (3)   adcValue[Distance] = read_mcp3208_adc(Distance);
        adcValue[Distance]
            = 27*pow((adcValue[Distance]*VCC/4095), -1.10);
        if(adcValue[Distance]<10)
            dWrite(4,ON);
        else
            dWrite(4,OFF);
        ......
      }
}
```

Fig. 3. Test code.

4. Conclusion

This paper proposed a JavaScript software platform for IoT. The platform allows the control of various hardware of IoT devices using JavaScript. The module control functions are embedded in the SpiderMonkey engine, and Raspberry Pi was used as an IoT device to implement the proposed software platform.

As IoT devices are supposed to be low-cost platforms, embedding a JavaScript engine would become excessively heavy for the lightweight IoT devices. As future work, we plan to simplify the JavaScript engine by removing the unnecessary components for IoT devices, such as garbage collectors. With this simplification, the proposed software framework can become applicable to actual IoT devices.

References

[1] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, D. Boyle. From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. Elsevier, 2014
[2] Node.js home page. Article (CrossRef Link)
[3] S. Tilkov, S. Vinoski. Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing, 14(6):80-83, November 2010.
[4] NPM is a package manager for JavaScript. Article (CrossRef Link)
[5] Mozilla Foundation. SpiderMonkey. Article (CrossRef Link) Mozilla wiki: Article (CrossRef Link)
[6] Mozilla Foundation. mozjs-xx is Spidermonkey version name. Article (CrossRef Link)
[7] RASPBERRY PI FOUNDATION. Raspberry Pi. Article (CrossRef Link)
[8] GPIO pin number library. Article (CrossRef Link)

Wonjun Lee received his B.S. degree in Electrical Engineering from Korea University, Jo Chi Won, Rep. of Korea, in 2014. Currently he is a student in Ph.D. course in Korea University. His research interests include IoT architecture and NFC. He is a student member of IEEE

Yeoul Na received her B.E. degree from Electrical Engineering at Korea University, Seoul, Korea, in 2008, and is working on her Ph.D. at the same department. Her research interests include microprocessor architecture, system on chip designs, and compilers. She is a student member of IEEE.

Seon Wook Kim received the BS in Electronics and Computer engineering from Korea University, Seoul, Rep. of Korea in 1988. He received the MS in Electrical Engineering from Ohio State University, Columbus, Ohio, USA, in 1990, and the Ph.D. in Electrical and Computer engineering from Purdue University, West Lafayette, Indiana, USA, in 2001. He was a senior researcher at the Agency for Defense Development from 1990 to 1995, and a staff software engineer ant Inter/KSL from 2001 to 2002. Currently, he is a professor with the School of Electrical and Computer Engineering of Korea University and Associate Dean for Research at the College of Engineering. His research interests include compiler construction, microarchitecture, and SoC design. He is a senior member of ACM and IEEE.