2015 IEEE 39th Annual International Computers, Software & Applications Conference

# Internet of Things-based Temperature Tracking System

Amir Atabekov, Marcel Starosielsky, Dan Chia-Tien Lo, and Jing (Selena) He
Department of Computer Science
Kennesaw State University
Kennesaw, USA
{aatabeko, mstarosi}@students.kennesaw.edu; {dlo2, jhe4}@kennesaw.edu

***Abstract*—The work described in this paper consists of a temperature tracking system that follows a Client-Server architecture. A RaspberryPi, a System-on-a-Chip (SoC) device, is responsible for sensing the temperature and streaming it to a server; the readings then are displayed in a mobile android application. For this system, a python application was developed to sense and stream the temperature, a servlet was developed to read and store the temperature in a SQLite database, and a mobile Android application was developed to read and display the temperature readings from the server. The initial versions of the project used the SoC device as a server (storing temperature readings into a local SQLite database), and both the SoC device and the mobile device needed to be connected in a local area network. However, the project was further developed to separate the server responsibilities from the SoC device. The system now supports user authentication, and both devices are connected through the Internet. This implementation allows the temperature readings to be viewed and displayed anytime from anywhere in the world since the database is hosted on a server which can be accessed over the internet. Also, this solution allows multiple SoC devices to stream temperatures to the server, to different mobile clients using the same database. The Android client application was also implemented to graphically show the temperature readings recorded by RaspberryPi using RESTful architecture. Moreover, an alert message notification was implemented in Android application so that a user is notified whenever the temperature reading reaches the preset threshold. On the other hand, the smart chair system has brilliant commercial prospects, which can be helpful to build health care products with the help of wearable sensors, intelligent refrigerator/oven temperature tracking system and etc.**

***Keywords— RaspberryPi; Temperature Tracking; Python; Internet of Things, RESTful API.***

## I. INTRODUCTION

The Internet of Things (IoT) involves connecting objects to internet so that the information related to the objects can be accessed at anytime from anywhere. The motivation of this project is to develop a device that makes temperature readings available over internet. We developed a system that helps address overheating issues that can occur in a home environment, or at a server warehouse. Heat generated by electronic devices must be dissipated in order to prevent premature failure. Although electronic devices have heat-dissipating mechanisms, sometimes they are not enough to effectively solve the issue.

Most electronic devices include a cooling system that expels heated air out while letting fresh air move in. However, when such a device is in an environment with insufficient airflow, the surrounding air gets hotter and further increases the temperature of the environment, causing the equipment to overheat.

On smart devices, such as a server cluster, there are sensors that will detect when the device is overheating and will shut itself down to prevent damage to its inside components. However, these sensors will track the temperature of the device only and will not be aware of the environment's temperature (i.e. server closet), where the problem might reside. The temperature tracking system we developed can be used to provide information that will help administrators or users diagnose the problem, and remediate it before the device takes drastic measures (such as turning itself off).

Furthermore, this system can also be used in a home environment. Most houses have a thermostat that controls the air conditioner to make sure the room is always at the desired temperature. However, this will only work effectively at rooms where the thermostat is present. At rooms without a thermostat, the system can be used to spot the difference between the desired temperature and the actual temperature of the room; the user can then use this information to try to remediate the problem by increasing airflow in air conditioning ducts in rooms that have uneven air conditioning.

The device used to sense the temperature is the Raspberry Pi as show in Fig. 1, and a circuit had to be assembled and connected through its GPIO. The Raspberry Pi was chosen for this project due to its small size and vast availability of third party sensors, extensions and accessible APIs. In this system, the following components were assembled and connected to the Raspberry Pi (shown in Fig. 2):

- Temperature sensor TMP36
- MCP3008 interface chip
- Breadboard
- Wi-Fi dongle
- GPIO to breadboard interface.

0730-3157/15 $31.00 © 2015 IEEE
DOI 10.1109/COMPSAC.2015.261

The TMP36 sensor was hardwired into the MCP3008 interface chip, which was responsible for providing digital input to the Raspberry Pi. The Plotly Python API and Scripts [1] were used and modified to poll and stream the temperature readings to a servlet. The servlet reads the information streamed from the python application and stores it in a SQLite database on the device. In order to successfully stream the temperature, the python application streams the temperature under a username and password (different SoC devices from different users stream the readings to the same server), the servlet then authenticates the credentials before storing the temperature readings; readings are only stored if the authentication was successful. Another servlet was also implemented to process the requests from the client application, and provide the temperature readings in JSON format.

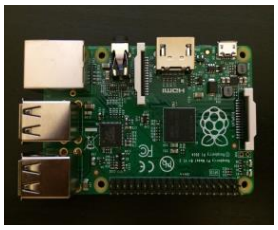

Fig 1. RaspberryPi System-On-a-Chip

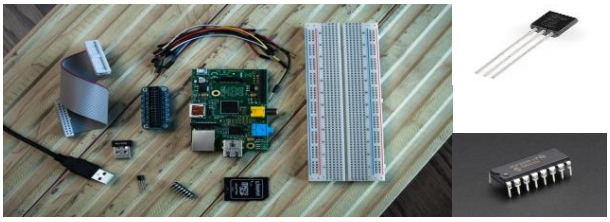

Fig 2. Components of Temperature Tracking System

An Android mobile application was developed as the client side, to obtain the temperature data for from the server; the application obtains the data in JSON format wirelessly, and once the data is obtained, the Android application displays a graph showing the temperature readings of any given day. Also, the application has the option to set a temperature threshold, and if this threshold is reached, the application will notify the user even if the application is running in the background.

The remaining of the paper is organized as follows. In Section II, we discuss some related works. Section III describes the high level implementation of the project and describes the tools used. Section IV describes the actual implementation with design diagrams of both the server and client application. Section V discusses the performance evaluation of the system. We conclude the paper and discuss some future extensions of the system in Section VI.

## II. Related Works

Internet of Things (IoT) has emerged as a new internet paradigm, which allows various physical entities in the world to connect with each other. The observed or generated information of these entities have a great potential to provide useful knowledge across different service domains, such as building management, energy-saving systems, surveillance services, smart homes, smart cities, etc. [2 - 5]. IoT was coined in 1999 by Kevin Ashton, who is the cofounder of Auto-ID center at the Massachusetts Institute of Technology (MIT) [6]. Now, IoT represents a technological revolution, which includes several advanced technologies, such as identification and contactless data exchange (RFID [7], and Near Field Communication (NFC) [8]), wireless sensor networks [9], short-range wireless communication (ZigBee [10] and Bluetooth [11]), and universal mobile accessibility (Wi-Fi hotspots [12] and cellular networks [13]).

In this project, we built a temperature tracking system, a case study of IoT applications. One related temperature-tracking project has been completed by Boonsawat et al. in [14], where Arduino microcontroller board [15] was used. The project consisted of several Arduino boards which transmitted the temperature to the master node with ZigBee shields. The RaspberryPi [16] can serve a similar purpose, however unlike Arduino, the RaspberryPi can also serve as a stand alone web server with Tomcat application server on the local network. This can obviate the need to use third party RESTful Web APIs [17] if the data is to be accessed locally. Hence, in this project we adopt RaspberryPi using RESTful architecture to implement temperature-tracking system.

## III. System/Network Model and Envoronment

### A. *System/Network Model:*

The initial versions of the project had the SoC device working as a server, and both the SoC device and Android mobile device needed to be connected on the same local area network, as the Fig. 3 shows.

However, the project was further developed beyond our initial plans. The system now supports user authentication, devices are connected through the internet and temperature readings are viewed and displayed from a server which is accessed over the internet anytime from anywhere in the world, as Fig. 4 shows.

In Fig. 4, the python application streams the readings to a java servlet. The servlet then authenticates the credentials provided by the python application's post request and stores the data in the SQLite database. On the client side, the android application requests the temperature readings from another java servlet, providing the credentials that the user typed at the log-in screen. The servlet authenticates the credentials and sends the temperature readings in JSON format to the android application. The android application retrieves new temperature readings at every 3 minutes (live readings), and readings are collected by the Raspberry Pi at every minute.

### B. *Tools and Environment:*

The android application was developed using the Java programming language, using the GraphView library [18] to draw the graphs that display the temperature readings. And the JSON library in Java [19] was used to send and parse the

temperature readings. Android Studio was used to develop the android application, and the Eclipse IDE was used to develop the Servlet Java application. Previously, the temperature readings were polled at every few seconds and the python application would run for unlimited time. However this approach is not reliable because if an error occurs during execution of the script, the temperature readings are terminated. This issue was fixed by establishing a timed execution of the python application, where the execution is controlled by the Linux Kernel of the Raspberry Pi (Raspbian OS [20]), a web-based interface for Unix system administration was used (Webmin [21])) to run the python application at every minute (Cron Job [22]).

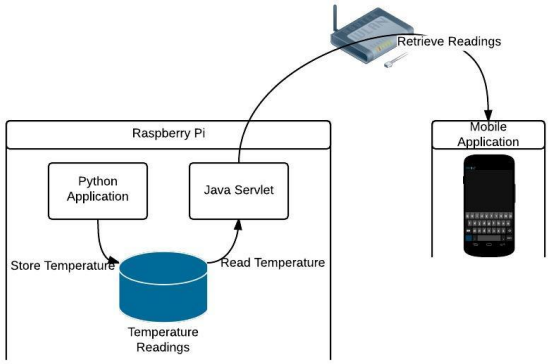


Fig 3. The original version of the system.

## IV. Methodology and Implementation

Our solution consists of two parts; the first part is the Web Service that runs in the cloud. The Web Service was implemented as a Java Servlet and runs on Tomcat application container. The second part consists of an Android Application, which acts as a client for the web service. Table 1 describes all system requirements for the project.

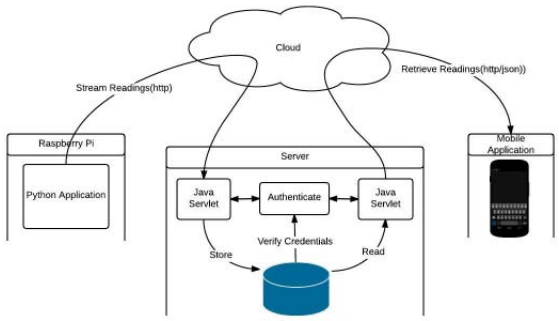


Fig 4. The new version of the system.

### A. Design of the Web Service

The web service was developed as five separate Java Servlets, which run on Tomcat application container as depicted in Fig. 5. Servlet named "NewRecord" accepts a POST request with user credentials and temperature reading as parameters and stores it into a SQLite database. Servlet "RaspberryPi" outputs all the temperature readings in JSON format. Servlets "Authenticate" and "CreateUser" are used to authenticate and register new users of the web service.

Table 1. System Requirements

| Requirement | Description |
|---|---|
| Read temperature from sensor | The python application running in the Raspberry Pi will collect data from the temperature sensor |
| Send temperature readings to our server and insert into SQLite database | The python application running in the Raspberry Pi will connect to our server in the cloud and POST a temperature reading into a SQLite database |
| Read the data from SQLite and output it in JSON format | Java web service that reads the temperature readings stored in the SQLite database and output them in JSON format. |
| Retrieve and parse JSON data | The android application retrieves and parses the temperature readings from Web service in JSON format. |
| Displaying and scaling the graph with temperature readings | The android application displays a graph with readings of the current day. Scaling is done automatically based on the range of data available. |
| View graphs from other days with previous/next buttons | The android application allows the user see the temperature graphs from other days, if there's data available in that day. |
| Login screen and progress bar of data transmission between android application and server | The android application allows the user to log in using his e-mail address and specify the server's IP address. After the login screen, a progress bar is shown to display the status of data transmission. |
| Authentication/Registration of users | The mobile application will let new users create an account which can be used with our web service. Afterwards, they can authenticate with their credentials. |
| Temperature threshold warnings | The application will let the user set a temperature threshold and will warn the user if it's reached. |
| Web service authentication | The web service will authenticate the user first, before either retrieving the data, or inserting a new record into the database. |

### B. Design of the Web Service

The web service was developed as five separate Java Servlets, which run on Tomcat application container as depicted in Fig. 5. Servlet named "NewRecord" accepts a POST request with user credentials and temperature reading as parameters and stores it into a SQLite database. Servlet "RaspberryPi" outputs all the temperature readings in JSON format. Servlets "Authenticate" and "CreateUser" are used to authenticate and register new users of the web service.

### C. JSON output from the Web Service

When "RaspberryPi" servlet is requested all the temperature readings are displayed in JSON format as described in Fig. 6. The Servlet outputs a JSON array called "temperature". The array consists of objects, each of which contains three key-value pairs. The first key-value pair is "date" which represents the date the temperature was recorded, the second key-value pair represents the actual temperature in Fahrenheit and the last pair represents the ID of the record.

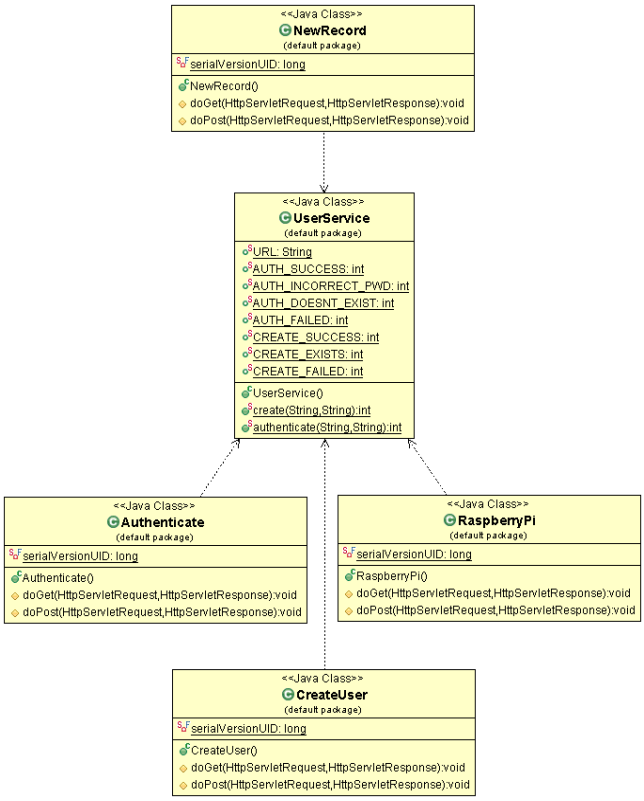


Fig 5. The web servlet class UML diagram.


"temperature": [

{"date": "2014-10-06 03:05:24.025496", "temp": "74.3", "id": 1},

{"date": "2014-10-06 03:05:29.160119", "temp": "74.8", "id": 2},

{"date": "2014-10-06 03:05:34.310643", "temp": "74.3", "id": 3},

{"date": "2014-10-06 03:05:39.470450", "temp": "74.8", "id": 4}]


Fig 6. JSON format of temperature readings.

### D. Design of the Android Application

Fig. 7 depicts the high level UML class diagram of the android application. JsonTask class is an asynchronous task, which periodically polls the server for updates, which it then passes to TemperatureActivity class. TemperatureActivity class uses GraphView library to display properly formatted graph of the temperature throughout the day.

### E. Layout Screens and Technical Details:

To display the temperature readings, the GraphView library [18] was used. Readings for every day were stored in a Hashmap data structure, and then passed to the API methods. Swipe gestures using the standard android library were implemented, which trigger events that cause the graph to change according to the previous or next day available; this is done by searching the Hashmap data structure for the next or previous available day, and then calling the GraphView API methods to display a new graph (as shown in Fig. 8). When the button Change Scale is pressed, the graph is re-displayed with a new scale (either Fahrenheit or Celsius). The Data flow of the application is summarized in Fig. 9.

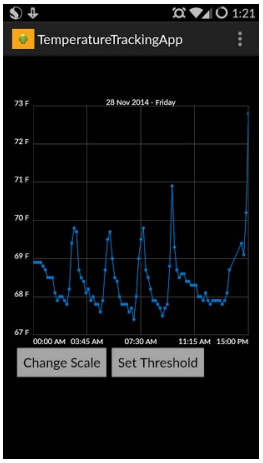


Fig 8. Application displaying the hourly graph

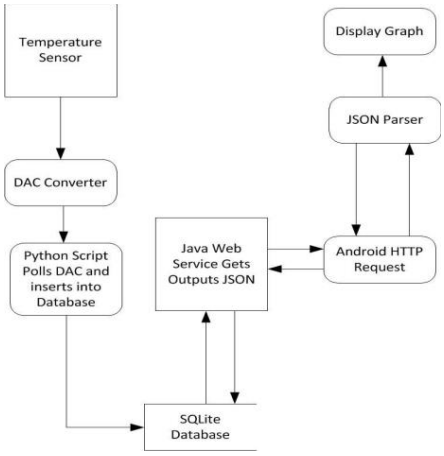


Fig 9. Data Flow Diagram

As Fig. 10 shows, when the button "Set Threshold" is pressed, a dialog appears where the user can chose a maximum limit for the temperature. If that limit is reached, then a warning will appear for the user warning that the threshold has been reached (shown in Fig. 11). The component used to scroll through the threshold options is a

NumberPicker, where the user can swipe up and down to choose the desired value. Also, the user can reset the threshold by selecting the option “Reset” and then pressing the ok button.

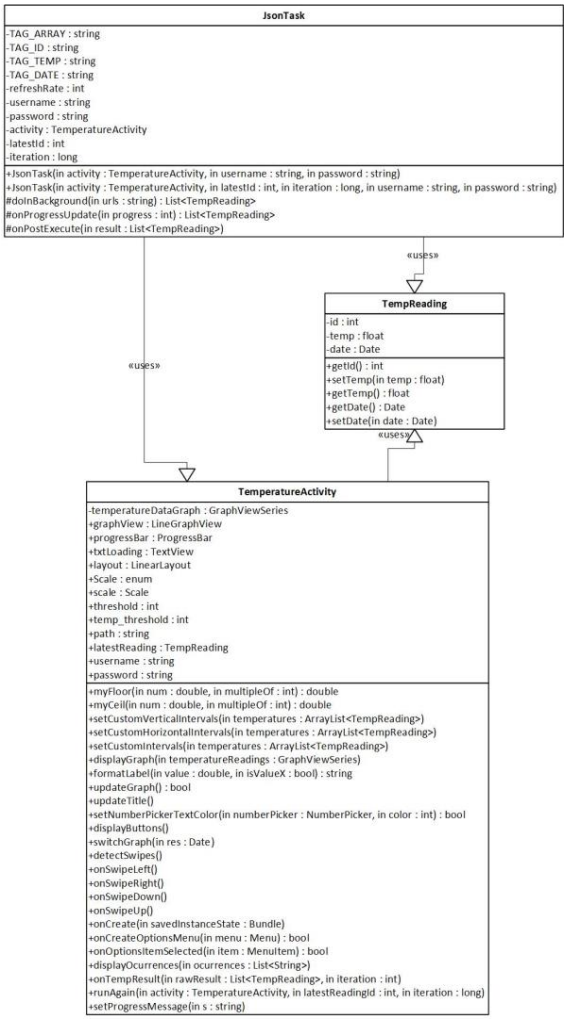


Fig 7. UML Class Diagram.

## V. PERFORMANCE EVALUATION

The temperature tracking system has the disadvantage that it needs Internet connection in order to view the latest readings. However, to access the achieved readings from previous days internet is not required. The android application is one of the most interesting components, filled with interesting features, where the following are the most distinguishable features:

- **Temperature threshold:**

The android application allows the user to set a maximum limit for the temperature. If that limit is reached, the application will display a message informing him/her that the temperature has reached the limit.

- **Refresh graph with latest readings:**

The android application re-loads the graph representing the readings of the current day, with the latest data available.

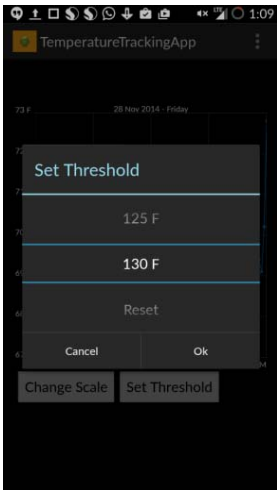


Fig 10. Displaying number picker to set the threshold

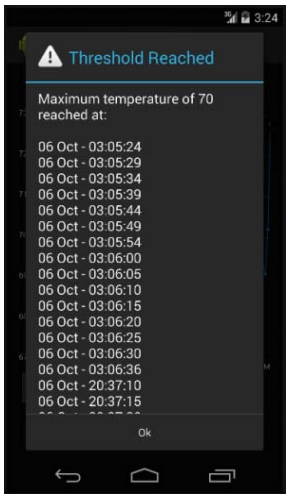


Fig 11. Displaying the threshold reached message

- **Temperature conversion:**

The android application allows the user to press a button, which will convert the temperature from Fahrenheit to Celsius, and display the new graph. Also, since all the functionalities were developed in time, the following functionalities were also developed to further improve the mobile application’s performance, reliability and user comfort.

- **Users connect to Raspberry Pi through internet:**

A hosted Web Service was developed so that multiple users are able to access the temperature readings from the Raspberry Pi through the Internet without the previous limitation of being on the same network as Android client. Also, the python application was modified to stream the temperature to the hosted web service, rather than storing it in a local database.

- **User authentication in Web service:**
  Since access to the readings is provided through the internet, user authentication is essential. The Web service authenticates the data first before sending the temperature readings to the client application. Temperature readings are streamed under different user names, which allow the database at the hosted server to store readings for multiple users, and yet each client application only displays the readings belonging to the username informed at the initial log-in screen.
- **Swipe gestures**
  To navigate through the days where each graph is shown, the user can now swipe left and right, instead of using the previous/next buttons.

## VI. Conclusion and Future Work

In conclusion, we would like to point out that we, as a team, have gained invaluable knowledge in hardware design, prototyping and python scripting. After developing this project we have identified a simpler and cheaper alternative to RaspberryPi, which is Texas Instruments (TI) LaunchPad, *i.e.,* MSP-EXP432P401R [23]. This LaunchPad is low powered but with high performance controller. In future we are interested to develop the same project based on TI LaunchPad instead of RaspberryPi, just so that we can gain more knowledge in hardware prototyping and further cutting the budget and energy consumption. By using a different board, we would only have to re-write the code for the TI LaunchPad to send Http POST request to our Web Service. The architecture as it stands now is very robust and uses open protocols for data exchange such as JSON. This makes modifying or using a different board much easier in future.

## Acknowledgment

This work is funded in part by the Kennesaw State University the Office of the Vice President for Research (OVPR) Pilot/Seed Grant, by the College of Science and Mathematics Interdisciplinary Research Opportunities (IDROP) Program, by the Department of Computer Science Mini-Research Grant, and supported by the National Science Foundation under Grant Number 1438858. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## References

1. Assembly tutorial and Plotly Python API sample Scripts: https://plot.ly/raspberry-pi/tmp36-temperature-tutorial/#hookup (accessed in April 2015).
2. D. Bandyopadhyay, and J. Sen, *Internet of things: Applications and challenges in technology and standardization*, Wireless Personal Communication, 58, 49–69, 2011.
3. R.V. Kranenburg, E. Anzelmo, A. Bassi, D. Caprio, S. Dodson, and M. Ratto, *The internet of things.* In Proceedings of 1st Berlin Symposium on Internet and Society, Berlin, Germany, October, 2011.
4. J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, *Internet of Things (IoT): A vision, architectural elements, and future directions,* Future Generation Computing System, 29, 1645–1660, 2013.
5. O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, and P. Doody, *Internet of things strategic research roadmap,* In Internet Things-Global Technology Social Trends, River Publishers, pp. 9–52, 2011.
6. K. Ashton, *That "Internet of Things" Thing*, RFiD J. 2009, 22, 97 – 114, 2009.
7. R. Want, *An introduction to RFID technology,* Pervasive Comput, 5, 25–33, 2006.
8. T.G. Zimmerman, *Personal area networks: Near-field intrabody communication*. IBM System Journal, 35, 609–617, 1996.
9. L. Eschenauer, and V.D. Gligor, *A key-management scheme for distributed sensor networks*. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, DC, pp. 41–47, November, 2002.
10. P. Baronti, P. Pillai, V.W. Chook, S. Chessa, A. Gotta, and Y.F. Hu, *Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and ZigBee standards,* Computer Communication, 30, 1655–1695, 2007.
11. Bluetooth, *S.I.G. Specification of the Bluetooth System*, version 1.1. Available online: http://www. bluetooth.com (accessed in April 2015).
12. G. Anstasi, M. Conti, E. Gregori, and A. Passarella, *802.11 power-saving mode for mobile computing in Wi-Fi hotspots: limitations, enhancements and open issues*, Wireless Networking, 14, 745–768, 2008.
13. M.K. Karakayali, G.J. Foschini, and R.A. Valenzuela, *Network coordination for spectrally efficient communications in cellular systems,* Wireless Communication, 13, 56–61, 2006.
14. V. Boonsawat, J. Ekchamanonta, K. Bumrungkhet, and S. Kittipiyakul, *XBee Wireless Sensor Networks for Temperature Monitoring*, the Second Conference on Application Research and Development (ECTI-CARD 2010), Chon Buri, Thailand, May 2010.
15. Arduino, http://arduino.cc/ (accessed in April 2015).
16. RaspberryPi, http://www.raspberrypi.org/ (accessed in February 2015).
17. RESTful Web APIs, http://restfulwebapis.com/ (accessed in April 2015).
18. Graphview: Android Java API library, http://android-graphview.org/ (accessed in April 2015).
19. JSON for Java API library, http://www.json.org/java (accessed in April 2015).
20. Raspbian, http://www.raspbian.org/ (accessed in April 2015).
21. Webmin, http://www.webmin.com/ (accessed in April 2015).
22. Cron Job, http://www.unixgeeks.org/security/newbie/unix/cron-1.html (accessed in April 2015).
23. Texas Instruments LaunchPad, http://www.ti.com/ww/en/launchpad/launchpads-msp430.html#tabs (accessed in April 2015).