CHAPTER

A WOT TESTBED FOR RESEARCH AND COURSE PROJECTS 6

Mina Younan*, Sherif Khattab†, Reem Bahgat†

Computer Science Department, Minia University, Minia, Egypt Computer Science Department, Cairo University, Cairo, Egypt†*

6.1 WHAT YOU NEED TO GET STARTED

The main objective of this chapter is to gain the required practical knowledge and skills for building simple physical testbeds for the WoT, which integrates the real world into the digital world. Concrete steps for building a WoT testbed are presented in the form of four experiments and a mini-project. Testbed evaluation is out of the scope of this chapter; our work elsewhere [1] discusses evaluation of an integrated WoT testbed. This chapter focuses on the interaction between microcontrollers, sensors, actuators, and PCs using HTTP and Zigbee protocols. The required platforms and devices for running examples in this chapter are as follows:

- Platforms: C#, ASP.net, and Arduino programming language.
- Devices: Examples in this chapter are for WoT-based smart home applications and will use the components listed in Table 6.1.

6.2 INTRODUCTION

Augmenting everyday's objects (e.g., light bulbs, curtains, and appliances) with embedded computers or visual markers (e.g., LEDs and small LCD displays) allows things and information about them to be digitally accessible through the Web or mobile phones [1,2]. They become the Internet's interface to the physical world by converging the physical world into digital world [3,4]. With a partial lack of efficient and scalable communication standards, the number of devices connected to the Internet will increase rapidly as soon as IP becomes the core standard in the field of embedded devices. It is expected to reach the order of billions in 2020 [5]. The IoT

Managing the Web of Things. DOI: 10.1016/B978-0-12-809764-9.00008-1

Table 6.1 Smart Building Components

Component	Count	Description
Arduino	2	Microcontroller of type (Uno-R3 or Mega2560)
XBee Series	2	R3 Pro 3 For building a WSN using Zigbee protocol
XBee Shield	2	For installing XBee modules on Arduinos
XBee PC Shield	1	For joining PC to the network
DHT11/LM35	2	Humidity and Temperature sensors
Light Dependent Resistor (LDR)	2	LDR is sometimes called a Photoresistor or a Photocell. It is used for sensing brightness or darkness level in a certain place (e.g., room).
LEDs (2 × RGB, 2 × Blue)	4	RGB LED is a triple-light led (red, green and blue). It is used for representing power consumption levels, room state, and so on. For example, it becomes red to represent *hot* state.
Fan	2	Fan device should consume at most 5.0 voltages.
Resistors (6 × 150 Ω, 2 × 10 kΩ)	8	For adjusting voltage or current to other devices (i.e., for adjusting sensitivity of sensors (e.g., LDR)).
Bread board	2	Connection board.

***Additional accessories**: jumper wires for connecting components and USB programming cables for programming microcontrollers (Arduinos).*

and the WoT address this challenge by using IP and IPv6 (6LoWPAN) for embedded devices [6].

From the Wireless Sensor Network (WSN) to the IoT and moving forward to the WoT, this trend has spread for the last two decades [7]. Haller [8] discusses main concepts about the IoT (e.g., SThs and EoIs). The IoT focuses on the infrastructure layer for connecting and controlling SThs through the Internet, whereas the WoT is the application layer that visualizes IoT data (sensory data) using standard web tools (e.g., HTTP) and services such as Representational State Transfer services (REST) and RESTful APIs [9]. The HTTP protocol is used in the WoT as an application protocol [10]. Some efforts have been done on HTTP libraries to be compatible with embedded devices' capabilities, so that data and services become accessible using web standards [7]. Due to web 2.0, users can get static as well as dynamic information about resources.

Using a single protocol in the WoT will not satisfy all communication needs between heterogeneous things, and HTML has to embed additional formats for representing SThs and EoIs' properties and states [11]. As a result, the WoT needs special search engines like Dyser [3] and WoTSF [12].

Muhammad et al. [13] summarize differences between the concepts of emulators, simulators, and physical testbeds. They conclude that physical testbeds provide more accurate results. Thus, building a real WoT testbed that simulates the desired set of conditions and events in certain environments produces more accurate results [2].

This chapter addresses the problem of designing and implementing testbeds for WoT research and course projects. Practical knowledge about building the WoT testbeds starts with configuring and connecting components at the IoT layer (i.e., building the WSN layer). Building the WoT follows the architecture discussed in [7]. Mini-projects in this chapter cover general services of the testbed, such as (1) getting real-time data from SThs, (2) monitoring SThs and EoIs using standard web tools and services, and getting and saving datasets. The WoT testbed can act as a simulator for the physical environment by attaching datasets to the application layer to run in offline mode like the integrated testbed environment presented in [1].

The rest of this chapter is organized as follows. The next section discusses the WoT features and challenges. Section 6.4 briefly surveys IoT and WoT testbeds. Section 6.5 discusses the hardware and software components of a WoT testbed. Section 6.6 presents concrete steps for building mini-projects and course modules, which are combined after that in Section 6.7 to build a physical testbed for the WoT. Section 6.8 summarizes the chapter.

6.3 WOT FEATURES AND CHALLENGES

Sensors can provide great benefits when their readings and states are presented in a meaningful and friendly way to users and machines. For example, users need to know the logical path representing physical location (e.g., Building X, Floor Y, Room Z) instead of sensors' longitude and latitude. The potential of the WoT lies in interconnecting and integrating services with human users in different WoT networks. Searching for SThs and EoIs is one of the most important services in the WoT, where users search in real-time in WoT datasets that are collected in different formats [3,12]. This service needs special search engines due to dynamic nature of SThs readings and EoIs states.

Due to the great interest in converting things into SThs, more challenges have been found in the WoT [14–16]. Challenges are classified in brief as follows. Firstly, some challenges are concomitant to the IoT; these are: (1) huge number and heterogeneity of connected devices; (2) no standardized naming for SThs' attributes (during registration process); (3) dynamic states (e.g., readings) and dynamic attributes (e.g., locations for movable objects on which sensors and actuators are attached); and (4) logical path not considered as a SThs's attribute. Secondly, other challenges are concomitant to the WoT; these are: (1) partially non-crawlable WoT pages; as most WoT pages host dynamic parts (e.g., coded using AJAX) for monitoring SThs and EoIs in real-time, most of search engines' spiders cannot crawl them and (2) none standardized naming for states; a single STh state is represented using different wordings that has the same semantic meaning.

Incorporating a sense of WoT challenges and features (e.g., dynamic information) in datasets generated by WoT testbeds allows for producing more accurate results. In the light of the challenges and dataset requirements (e.g., lack of information about the infrastructure layer) discussed in [2], we summarize our observations on dataset contents. If the research is only interested in SThs values or in EoIs states, the used dataset is based on the WoT level (dynamic information), whereas if the research is interested in the network infrastructure, the used dataset is based on the IoT level (static information). Because the SThs may be movable objects (i.e., their locations may be changing frequently), then the research may need an additional type of information, which is called quasi-dynamic information about SThs. In this case, such a property about SThs will be considered as a special type of their readings (dynamic information). An integrated dataset contains information about both sensor readings and network infrastructure, that is, it is based on both IoT and WoT levels.

6.4 A BRIEF SURVEY OF IOT AND WOT TESTBEDS

Several studies [13,17–19] discuss and compare between existing simulators and testbeds using general criteria, such as the number of nodes, heterogeneity of hardware, and portability, but none of them discusses WoT features, such as STh's logical path, supported formats in which EoIs' states are presented, and accuracy of the datasets generated by the testbeds.

In the following sub-sections, we briefly survey testbeds and measurement platforms that combine both IoT and WoT features.

6.4.1 IOT SIMULATION

There is no general way for simulating the IoT [13,18,19]. Moreover, there are situations in which simulators and real datasets containing raw information (e.g., sensor readings [20]) (less information about the IoT layer are present) are not enough for modeling an environment under testing. When datasets miss the sense of one or more of the WoT features or challenges (discussed above) [2], they miss main factors for accurate WoT evaluation [1]. Also, many datasets are not actually related to the problem under investigation, but were generated for testing and evaluating different algorithms or methods in other research efforts. For instance, an evaluation of WSNs' simulators according to a different set of criteria, such as the Graphical User Interface (GUI) support, the simulator platform, and the available models and protocols, concludes that there is no general way for simulating WSNs, and hence IoT and WoT [18,19]. None of these criteria addresses the previous challenges. So, it is desirable to embed the unique IoT and WoT challenges within the datasets and to make simulators address as much of these challenges as needed.

6.4.1.1 WSN Simulators

Several studies [13,18,19] summarize the differences between existing WSN simulators according to a set of criteria, such as heterogeneity, scale, user involvement, limitations, etc. The Cooja simulator gives users the ability to simulate WSNs easily using a supporting GUI [18,19] and different types of sensors (motes) for different sensor targets. For instance, sensor applications are written in the nesC [21] language then built in the TinyOS environment [22] (e.g., the RESTful client server application [23]). However, there are limitations and difficulties for testing the extensible discovery service [24] and sensor similarity search [25] in Cooja, because there is no information about the network infrastructure and entities. In particular, static information about sensors and schematics information about buildings and locations of sensors need to be presented.

6.4.1.2 *WSN Physical Testbeds*

Physical testbeds produce accurate research results [13]. Different testbeds are found in this field due to different technologies and network scales. MoteLab [26] supports two ways for accessing the WSNs, (1) by retrieving stored information from a database server (i.e., offline) and (2) by direct access to the physical nodes deployed in the environment under test (i.e., online). However, the WoT challenges mentioned previously are not fully supported in MoteLab. User access in MoteLab is similar to what is done in the WoT testbed in [2].

SmartCampus [27] tackles gaps of experimentation realism, supporting heterogeneity (of devices), and user involvement [17] in IoT testbeds. CookiLab [28] gives users (researchers) the ability to access real sensors deployed in Harvard University. However, it does not support logical paths as a property for sensor nodes and entities (WoT features).

Nam *et al.* [29] present an Arduino [30] based smart gateway architecture for building IoT testbeds, which is similar to the architecture of the testbed environment proposed in [1] and [2] (e.g., they all use periodic sensor reporting). They build an application that discovers all connected Arduinos and lists the devices connected on each Arduino. However, the framework does not cover all scenarios that WoT needs, especially for searching. For example, information of logical paths and properties of devices and entities are missing in the framework.

6.4.2 WOT SIMULATION

Using websites (e.g., [31–33]), a WoT environment can be built online by attaching STHs like Arduinos [30]. These websites monitor the states of devices and provide RESTful services (GET, PUT, UPDATE, DELETE) [9] for uploading and accessing reading feeds. Moreover, the values (sensor readings) are visualized for users. The services of the aforementioned websites are similar to services of the testbed environment in [2]. However, these websites are limited by available service usage and formats of the responses, which are hardcoded and embedded within the website code or at least not exposed to users. The testbed architecture in [1], which is built specially for testing WoT, provides more general services, such as monitoring live information fed from attached STHs, visualizing sensor readings and states of EoIs over time, controlling actuators, triggering action events, and periodic sensor reporting.

A comparison of state-of-the-art IoT and WoT Testbeds has been discussed in our previous work [1] along two main axes: (1) the infrastructure layer elements (e.g., device heterogeneity), and (2) the application-layer elements (e.g., reusability).

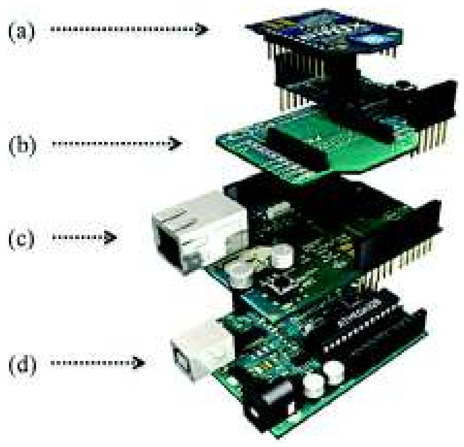


FIGURE 6.1

Hardware components for building IoT node: (a) XBee module of type pro S2B; (b) XBee shield; (c) Ethernet shield; and (d) Arduino board.

6.5 HARDWARE AND SOFTWARE COMPONENTS OF A WOT TESTBED

Because the WoT is foreseen as the future application of the IoT [11,23], the WoT will be built in two axes; the first axis (IoT layer) is to build the IoT layer by establishing a connection between all things (nodes) after converting them to SThs, and the second axis is to build a web application on top of the IoT layer to monitor and control SThs and EoIs that represent the set of required environmental events (WoT layer).

Building the IoT layer needs sensors, such as temperature sensor (e.g., LM35), actuators (e.g., RGB-LED), and microcontrollers, such as Arduino, Netduino, and Raspberry-Pi boards, to be connected or attached to things and objects using bread-boards. Microcontrollers act as gateways in the IoT. Building the IoT is done by implementing a certain network topology that covers the required environment. The main hardware components are Arduino, Ethernet shield and XBee module; these components are presented in the next three sub-sections followed by a brief description of Digi's configuration software (X-CTU [34,35]).

6.5.1 ARDUINO

Arduino [30], shown in Fig. 6.1(d), is the simplest type of microcontrollers, whereby its hardware and software components are open-sourced [30]. Arduino programming language is based on C++ programming language in the Arduino IDE. Arduino is considered as a tiny computer that can be programmed to perform a task, control the functionality of other components that are connected on its board (e.g., LM35 and LDR), and to process data according to program instructions. Arduino family

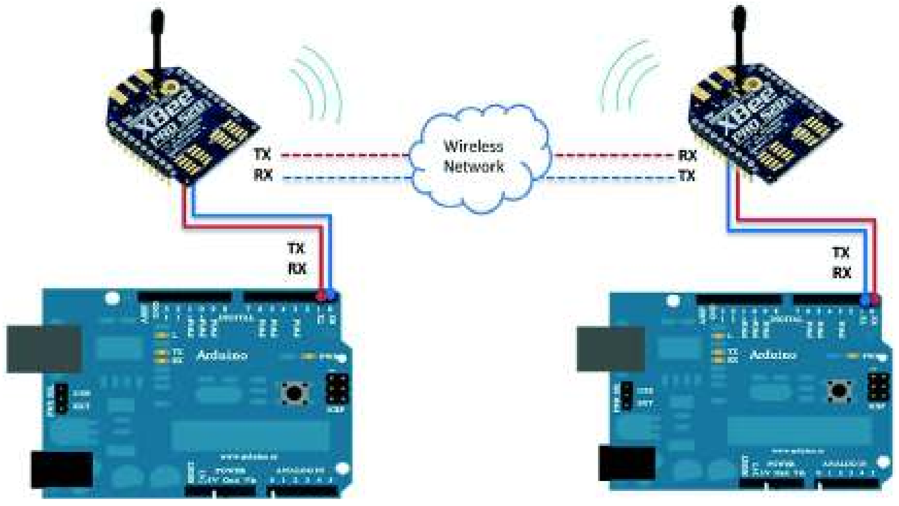


FIGURE 6.2

Two Arduinos connect wirelessly using XBee modules, which implement Zigbee protocol.

has different versions; the most famous one is of type UNO. The ATmega328 is main component of the Arduino, where it acts as a processor; it has 32KB of flash memory.

The Arduino can operate either independently (e.g., robot), connected to a computer (i.e., act as base-station), connected to other Arduinos (e.g., local area network between Arduinos, such as shown in Fig. 6.2), or connected to other controller chips. In other words, Arduino connects to the physical world through electronic sensors and actuators providing information about environmental events that sensors represent in the surrounding environment. It can use an Ethernet shield or Wi-Fi shield, so that it can act as a web server for sending and receiving data. Therefore. they can be used to represent things' states in real-time.

Arduino and Raspberry-Pi boards' families are of the most famous IoT boards for building IoT applications. Raspberry-Pi is like the Arduino, but it has more powerful features and capabilities; for example, it has bigger internal memory and can be programmed using different programming languages. But for simplicity, we will use Arduino boards (e.g., UNO and Mega2560) for building the infrastructure layer (IoT layer).

6.5.2 ETHERNET SHIELD

Monitoring components connected on Arduinos could be done in two methods. The first method is using Serial Peripheral Interface (SPI) and thread functions on base-stations (gateways), whereby sensors measure their values and Arduinos send devices' readings to base-stations, which in turn store the readings after analyzing the messages in the database. In this case, Arduino acts as a client to *push* device readings to base-stations [4]. The second method is to retrieve information from Arduinos directly using RESTful services. In this case Arduino acts as a web server [4], which provides RESTful services to control devices and *pull* device readings. Both methods require web applications (to host device pages) for monitoring states of devices connected on Arduinos and for retrieving information directly (i.e., real-time information) or indirectly from a database (i.e., stored information) using special embedded code (e.g., AJAX) in some parts of the web pages. As a result, the IoT is considered as a combination of push and pull methods for more and ever-increasing connectivity with any physical object or environmental events happening in the immediate and wider environment [36]. In order to make Arduino act as a web server, an Ethernet Shield [35] is installed on top of the Arduino board such as shown in Fig. 6.1(c).

6.5.3 XBEE PRO-SERIES2 HARDWARE

XBee [35,37], shown in Fig. 6.1(a), is a special type of small, cost-effective radios that enable microcontrollers (e.g., Arduinos) to communicate wirelessly with low power and low bandwidth. It is used with the Arduino Wireless Shield. XBee, Bluetooth, and Wi-Fi are used for allowing wireless communication between microcontrollers (gateways) in the WSN. XBee shield is a special type that is used for connecting XBee on Arduinos, as shown in Fig. 6.1(b).

As mentioned previously, the IoT is a global network of computers and sensors that communicate together through the internet using high-level (IP) or low-level (6LowPAN) protocols. So, we need to connect microcontrollers to each other forming a WSN according to a selected network topology. The XBee modules with ZigBee firmware (ZB firmware) [38] are designed to form networks with star, cluster, tree, or mesh topologies, whereby there is a hierarchy of devices and one coordinator is always necessary. For establishing wireless communication between Arduinos, radios should have the same type of firmware (e.g., ZigBee). Fig. 6.2 shows an example of communication between two Arduinos using XBee modules.

XBee uses ZigBee, 802.15.4, and DigiMesh protocols. Building a WSN using the ZigBee protocol is slower than 802.15.4 but allows for building well-structured network with nondeterministic throughput and sleeping endpoints based on a coordinator (base-station) and routers between the endpoints. The WSN that is built using ZigBee modules has three types of devices: (1) coordinator, (2) router, and (3) endpoint or end device. If the network topology is star, then the WSN contains two types only, coordinator and endpoint. The number of nodes per master node in the network may be 65,000 nodes with communication ranges from 70 meter to 300 meter (1.6 km possible).

6.5.4 X-CTU SOFTWARE (XBEE API PROTOCOL)

To configure XBee (ZigBee module) [34] as a coordinator, router, or an endpoint, we need to change the firmware files found on Digi's RF products using Digi's configuration software (X-CTU) [34,35]. X-CTU application is a Windows-based application.

It has multiple functions; each window tab in the application has a different function. For example, we can select *Range Test* tab for testing signal range between two Arduinos attached to XBee modules.

The ZigBee protocol allows setting up a radio link between the endpoints for covering a wide area by sending messages through multiple routers between endpoints. For enabling simple communication between two nodes (Arduinos with XBee modules), one of them should be configured with the coordinator firmware, and the other with router or endpoint firmware.

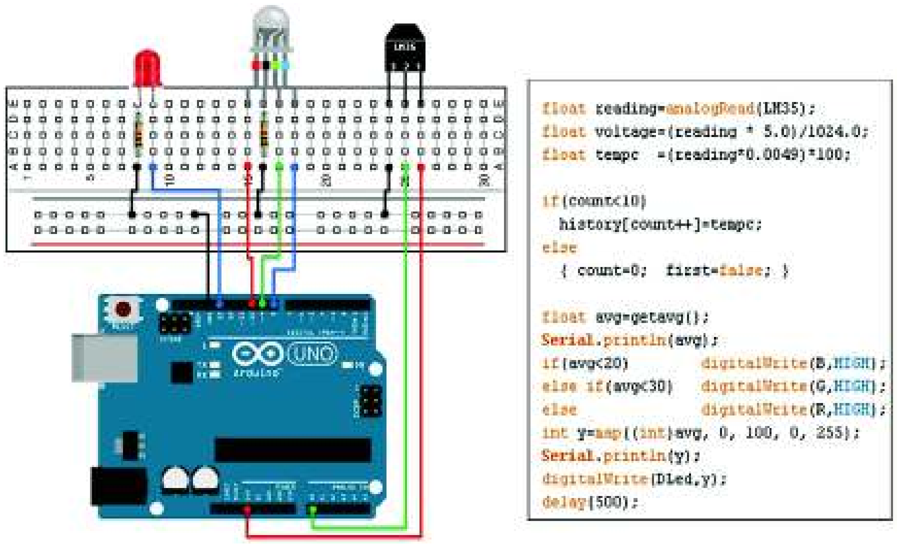
6.6 EXPERIMENTS FOR THE WOT

This section presents concrete steps for building a WoT testbed in the form of experiments that show how to build WoT-based smart home applications using Zigbee modules [38,39]. They are combined to eventually represent the main components of the WoT testbed. All experiments code can be downloaded [40]. The IoT layer consists of the physical network between STHs (Arduinos and sensors) and the IoT platform (also called IoT cloud) that provides services for charting sensory data in real-time using existing protocols (e.g., Xively [31] and Thingspeak [32]). The WoT enhances the IoT platform by building web applications that visualize IoT data and provide more services such as searching for STHs and EoIs [1,12]. In this section, we present experiments following the WoT architecture starting from converting things to STHs till visualizing STHs' data and EoIs' states, then these experiments are combined together forming a mini-project of a testbed for the WoT.

6.6.1 EXPERIMENT 01: CONVERTING THINGS TO STHS

Converting things to STHs is done by attaching sensors and actuators to things [7,8], so that STHs represent states of physical things online through the WoT. Integration of STHs is direct if they support IP connection and indirect if they speak other low-level protocols [41]. Converting things to STHs is the key element of the IoT [2,12, 42]. For example, converting a door to a smart door by adding a touch screen and a certain actuator allows locking and unlocking from anywhere. Representing power consumption of a building clarifies the difference between two concepts, STh and EoI. Representing power consumption could be done by converting devices to STHs through attaching special types of sensors (e.g., INA219 sensor). The building here is called an EoI, while devices attached with sensors are called STHs. The experiment for this example is simplified by using RGB-LEDs and Arduino whereby the RGB-LED becomes red when power consumption is higher than the average rate (calculated from historical data), becomes green when power consumption is in the average range, and becomes blue when power consumption is less than the average rate.

The first experiment, shown in Fig. 6.3, represents temperature state of a room using light level (brightness or diming level) of a LED located in that room. The



```
float reading=analogRead(LM35);
float voltage=(reading * 5.0)/1024.0;
float tempc  =(reading*0.0049)*100;

if(count<10)
  history[count++]=tempc;
else
  { count=0;  first=false; }

float avg=getavg();
Serial.println(avg);
if(avg<20)         digitalWrite(B,HIGH);
else if(avg<30)    digitalWrite(G,HIGH);
else               digitalWrite(R,HIGH);
int y=map((int)avg, 0, 100, 0, 255);
Serial.println(y);
digitalWrite(Dled,y);
delay(500);
```

FIGURE 6.3

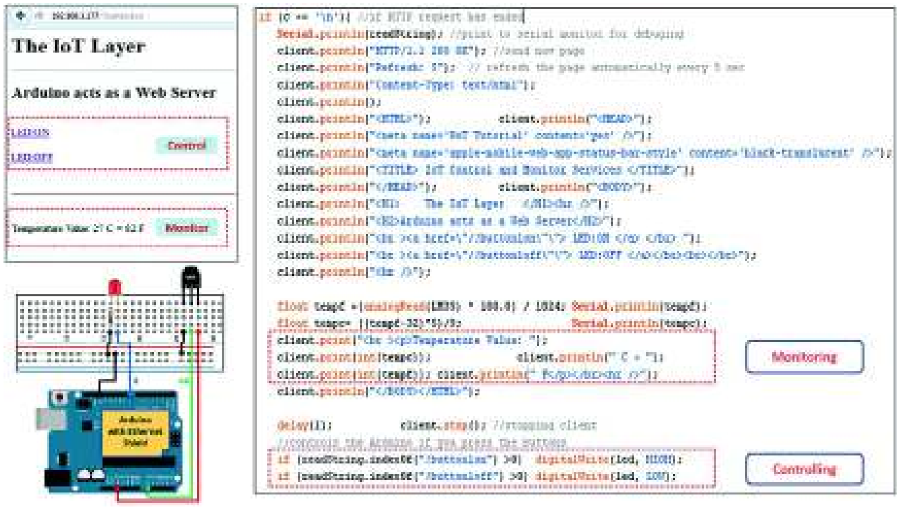
Experiment 01: Converting things and places to STHs and EoIs.

higher the dimming of the LED, the lower the room temperature, and vice versa. The RGB-LED is used as well, for representing the room state as a discrete set of states (hot, warm, and cold), where RGB-LED becomes red to represent *hot* state, green to represent *warm* state, and blue to represent *cold* state. A code segment of controlling the RGB-LED is shown in Fig. 6.3, where *getavg* is a function for calculating the average rate from sensor's historical readings, 50th percentile could be implemented by this function for getting more accurate results.

6.6.2 EXPERIMENT 02: INTEGRATING STHS AND EOIS IN THE IOT

Integrating STHs and EoIs in the IoT is done in two steps. The first step is to create profiles for newly joined STHs assigning them to EoIs. The second step is to monitor and control STHs through the Internet, forming bases of the IoT layer [4]. Creating profiles in this chapter is done manually using a configuration application (described in [1,43]), but it can be done automatically by discovering similar STHs in the IoT and selecting the most similar one, fetching its profile contents using semantics [44] and global ontologies as proposed in [45]. They propose a service-oriented middleware that abstracts things as services. Sensor similarity search could be used for this purpose [25] and the WoTSF search framework [12] enhances this process by using two-level indices; this automation is out of this chapter's scope and is planned as future work.

STHs' profiles are registered in a database, so that their static information could be retrieved. Registration is performed using a network setup application (as in [1]).


192.168.1.177
The IoT Layer
Arduino acts as a Web Server
LED:ON
LED:OFF
Control
Temperature Value: 27 C = 62 F
Monitor
Arduino with Ethernet Shield
Monitoring
Controlling

```
if (c == '\n'){ //if HTTP request has ended
  Serial.println(readString); //print to serial monitor for debuging
  client.println("HTTP/1.1 200 OK"); //send new page
  client.println("Refresh: 5"); // refresh the page automatically every 5 sec
  client.println("Content-Type: text/html");
  client.println();
  client.println("<HTML>");          client.println("<HEAD>");
  client.println("<meta name='IoT Tutorial' content='yes' />");
  client.println("<meta name='apple-mobile-web-app-status-bar-style' content='black-translucent' />");
  client.println("<TITLE> IoT Control and Monitor Services </TITLE>");
  client.println("</HEAD>");         client.println("<BODY>");
  client.println("<H1>   The IoT Layer   </H1><hr />");
  client.println("<H2>Arduino acts as a Web Server</H2>");
  client.println("<br /><a href=\"/?buttonlon\"\"> LED:ON </a> </br> ");
  client.println("<br /><a href=\"/?buttonloff\"\"> LED:OFF </a></br><br></br>");
  client.println("<hr />");

  float tempC =(analogRead(LM35) * 100.0) / 1024; Serial.println(tempC);
  float tempc= ((tempF-32)*5)/9;                  Serial.println(tempc);
  client.print("<br /><p>Temperature Value: ");
  client.print(int(tempc));            client.println(" C = ");
  client.print(int(tempF)); client.println(" F</p></br><hr />");
  client.println("</BODY></HTML>");

  delay(1);          client.stop(); //stopping client
  //controls the Arduino if you press the buttons
  if (readString.indexOf("/?buttonlon") >0)  digitalWrite(led, HIGH);
  if (readString.indexOf("/?buttonloff") >0) digitalWrite(led, LOW);
```

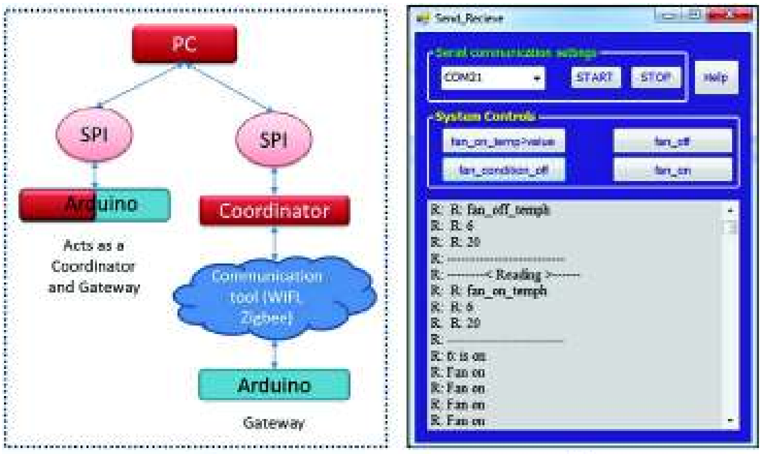
FIGURE 6.4

Experiment 02: Arduino acts as a web server supporting monitoring and controlling services for the IoT layer.

This application controls SThs and monitors EoIs locally (i.e., control is done using base-station connected to PC using SPI), and uses logical path in a smart home as an attribute for the STh [1].

The second step is monitoring SThs and EoIs in the IoT (e.g., monitoring indoor temperature) and controlling them online. This step is done (1) directly, when the Arduino gateway acts as a web server hosting its page on the attached SD-card or embedding HTML code in the response message (experiment is shown in Fig. 6.4), and (2) indirectly, by calling web services hosted on a base-station connected to the Arduino gateways, whereby the Arduinos receive commands and send raw sensory data to base-stations, which in this case are similar to the Dataset Collector application (DsC) in [1,2,46] (more details in the next sections). SThs pages are built so that they can be refreshed (i.e., reloading whole page contents) in less than a minute for monitoring SThs' states in real-time. Also, pages are coded using AJAX, so that only some parts are reloaded (partial refresh) instead of refreshing the whole page. The integrated WoT testbed in [46] codes SThs pages using AJAX.

Components in Fig. 6.3 are updated to be such as shown in Fig. 6.4, where the Ethernet shield is installed on top of the Arduino board, so that components could be monitored and controlled through the Internet, forming the IoT layer. Arduino code is also updated so that Arduino can act as a web server for monitoring room temperature using LM35 connected on pin A0, and for controlling LED connected on pin 8. GET RESTful API is implemented on the Arduino that is attached with Ethernet component and has an IP address (e.g., 192.168.1.177) by embedding HTML code in the reply message. Arduino checks if there is an available client and acts according



```
R:  R: fan_off_temph
R:  R: 6
R:  R: 20
R: ------------------------------
R: ---------< Reading >------
R:  R: fan_on_temph
R:  R: 6
R:  R: 20
R: ______________________
R: f: is on
R: Fan on
R: Fan on
R: Fan on
R: Fan on
```

FIGURE 6.5

Experiment 03: Generating Sensory Data from the IoT. (a) Base-station architecture; (b) Dataset Collector Application (DsC) for the IoT.

to client action written in the received message, as indicated in the code segment in Fig. 6.4 for controlling the LED. Arduino's web page is refreshed or reloaded every 5 seconds.

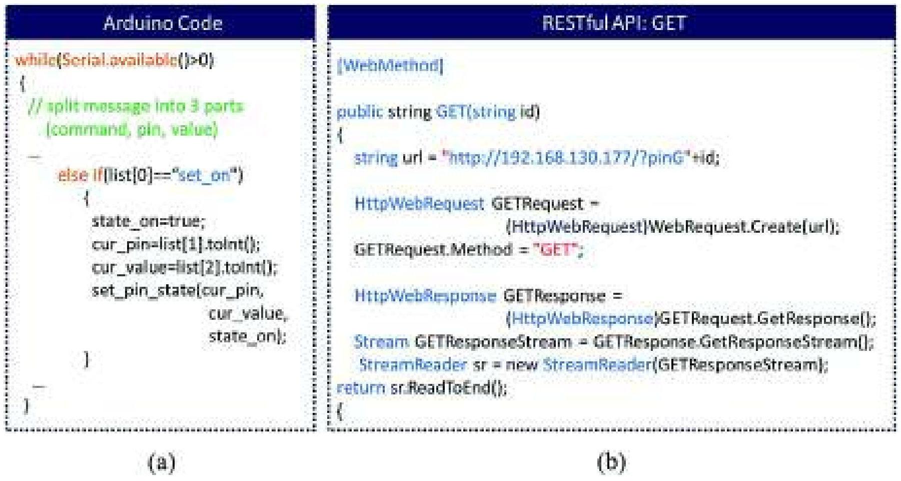
The most appropriate method for designing a web page for visualizing STh's data despite resource constraints (allowing monitoring and controlling its state in real-time) is to implement and use APIs (RESTful services). APIs act as a middle layer between physical gateways (Arduinos) and virtual gateways (web pages). This method is preferable for building WoT applications, where it can host static and dynamic information in the same page (e.g., WoT testbed in [46]).

To sum up, the concrete steps to build a WoT are so far as follows.

1. Building the physical layer for measuring the environmental events (using sensors) and for controlling the surrounding environment (using actuators).
2. Building web pages for SThs.
3. Building APIs for sending and receiving commands between the Arduinos and their web pages. These APIs are RESTful services [15,47].

6.6.3 EXPERIMENT 03: GENERATING SENSORY DATA FROM THE IOT

The following experiment discusses how to build a base-station that contacts Arduinos in the IoT for sending commands and receiving information. The base-station runs DsC application to collect sensory data about a set of environmental events in real-time. It sends a set of configurations or commands to Arduinos in the IoT network. The base-station monitors and controls the SThs in addition to collecting the



```
while(Serial.available()>0)
{
 // split message into 3 parts
   (command, pin, value)
 …
    else if(list[0]=="set_on")
       {
        state_on=true;
        cur_pin=list[1].toInt();
        cur_value=list[2].toInt();
        set_pin_state(cur_pin,
                      cur_value,
                      state_on);
       }
 …
}
```

```
[WebMethod]

public string GET(string id)
{
  string url = "http://192.168.130.177/?pinG"+id;

  HttpWebRequest GETRequest =
                 (HttpWebRequest)WebRequest.Create(url);
  GETRequest.Method = "GET";

  HttpWebResponse GETResponse =
                 (HttpWebResponse)GETRequest.GetResponse();
  Stream GETResponseStream = GETResponse.GetResponseStream();
   StreamReader sr = new StreamReader(GETResponseStream);
return sr.ReadToEnd();
{
```

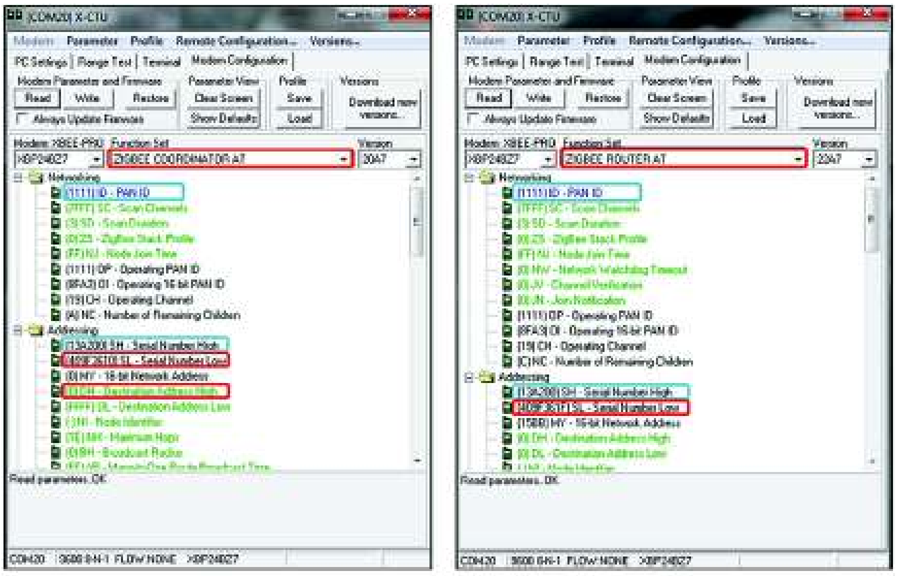
FIGURE 6.6

RESTful API: (a) Arduino code for handling incoming commands; (b) RESTful API GET for getting SThs' states.

datasets. The execution of this experiment starts by establishing a connection between the base-station and gateways (Arduinos), whereby the gateways are configured to implement a certain topology (e.g., mesh or star). Base-station is implemented by attaching a microcontroller (that acts as a XBee coordinator) to a PC via SPI cable, as shown in Fig. 6.5(a). The coordinator device connects to IoT gateways using a certain connection protocol (e.g., Wi-fi or Zigbee). The DsC runs on a PC and connects to the coordinator device via serial port connection to send and receive commands and messages.

In this experiment, the collected sensory data are about the status of the fan, and there is a dependency between the fan and temperature value (Fig. 6.5(b)). The code behind the buttons of the DsC are calls for SThs' RESTful APIs. Fig. 6.6(b) shows RESTful service *GET* that monitors STh's state, where it takes STh's id as an input parameter. In this experiment, because the DsC connects directly to Arduino, it calls serial port function *writeline* to send a command message (e.g., *serialPort.WriteLine("set_on,6,20")*). On the other hand, the Arduino receives the message and executes commands, such as shown in Fig. 6.6(a). In this example, *set_on* is a command for turning on the device that is connected on *pin 6* (the fan) with degree 20 (i.e., light or speed value is 20). Maximum digital value in Arduino is 255.

Sensory data are generated in the form of dataset in order to represent device states in real-time online and offline using a web application [1]. Arduino acts as a coordinator for sending and receiving messages or commands in the network, and acts as a gateway in the same time to report states of SThs connected on it (Fig. 6.5(a)). For a larger IoT network, the coordinator contacts the main gateways, which are configured



(a)

XBP24BZ7 ZIGBEE COORDINATOR AT 20A7

(1111) ID - PAN ID
(FFFF) SC - Scan Channels
(3) SD - Scan Duration
(0) ZS - ZigBee Stack Profile
(FF) NJ - Node Join Time
(1111) OP - Operating PAN ID
(8FA3) OI - Operating 16-bit PAN ID
(19) CH - Operating Channel
(A) NC - Number of Remaining Children
(13A200) SH - Serial Number High
(409F2610) SL - Serial Number Low
(0) MY - 16-bit Network Address
(0) DH - Destination Address High
(FFFF) DL - Destination Address Low
( ) NI - Node Identifier
(1E) NH - Maximum Hops
(0) BH - Broadcast Radius
(FF) AR - Many-to-One Route Broadcast Time

Read parameters...OK

COM20 9600 8-N-1 FLOW:NONE XBP24BZ7

(b)

XBP24BZ7 ZIGBEE ROUTER AT 22A7

(1111) ID - PAN ID
(FFFF) SC - Scan Channels
(3) SD - Scan Duration
(0) ZS - ZigBee Stack Profile
(FF) NJ - Node Join Time
(0) NW - Network Watchdog Timeout
(0) JV - Channel Verification
(0) JN - Join Notification
(1111) OP - Operating PAN ID
(8FA3) OI - Operating 16-bit PAN ID
(19) CH - Operating Channel
(C) NC - Number of Remaining Children
(13A200) SH - Serial Number High
(409F3617) SL - Serial Number Low
(158B) MY - 16-bit Network Address
(0) DH - Destination Address High
(0) DL - Destination Address Low
( ) NI - Node Identifier

Read parameters...OK

COM20 9600 8-N-1 FLOW:NONE XBP24BZ7

FIGURE 6.7

Experiment 04: Configuring ZigBee modules: (a) Coordinator profile and (b) Router profile.

as routers in a mesh topology, or end-devices in a star topology. Configuring the IoT network to implement a certain network topology is discussed in details in the next section.

6.6.4 EXPERIMENT 04: CONFIGURING IOT NODES USING X-CTU SOFTWARE

As mentioned previously, for building a WSN using XBee modules (i.e., using ZigBee firmware) [35,38], one node in the WSN should be configured as a coordinator to manage all endpoints in the WSN [39].

- **Configuring coordinator device using X-CTU**. As shown in Fig. 6.7(a), the device is configured as a coordinator by changing the function set to ZIGBEE COORDINATOR AT, setting PAN ID to 1111. XBee-Shield button *XBEE/USB* should be set on *USB* selection during code burning and on *XBEE* otherwise (e.g., sending and receiving signals). Coordinator and endpoints should have the same PAN ID (default gateway in IPv4), so that all nodes send their messages directly to the coordinator or indirectly though routers (i.e., nodes that share the same PAN ID can communicate with each other [34,35]). PAN ID, high word (SH), and low word (SL) are important items in node configuration. For sending requests from the coordinator to all nodes in the network, nodes should have the same PAN ID in their configuration and Destination Address High (DH) is set to zero in the coordinator. But for building a peer-to-peer communication, DH in the coordinator is set to the value of SH in the endpoint device.

- **Configuring router device using X-CTU**. In order to enable end-point devices to send their information directly to the PC (coordinator) implementing a star topology using ZigBee modules, the XBee module attached to the PC through the USB-shield should be configured as a coordinator (Fig. 6.7(a)), and end-point devices should be configured as routers or end-point devices, as shown in Fig. 6.7(b). This figure shows configuration of Arduino-Uno, where attributes surrounded with red rectangles in this figure are kept similar in all end-point devices or router devices, while attributes surrounded with light-blue rectangles are set to different values in all end-point devices.
- **Configuring end-point device using X-CTU**. For enlarging the network (i.e., implementing a different network topology, such as mesh topology), all the remaining nodes should be configured as endpoint devices, changing the function set in Fig. 6.7(b) from ZIGBEE ROUTER AT to ZIGBEE END DEVICE AT. More details about types of configuration are in [34,35,38,48].

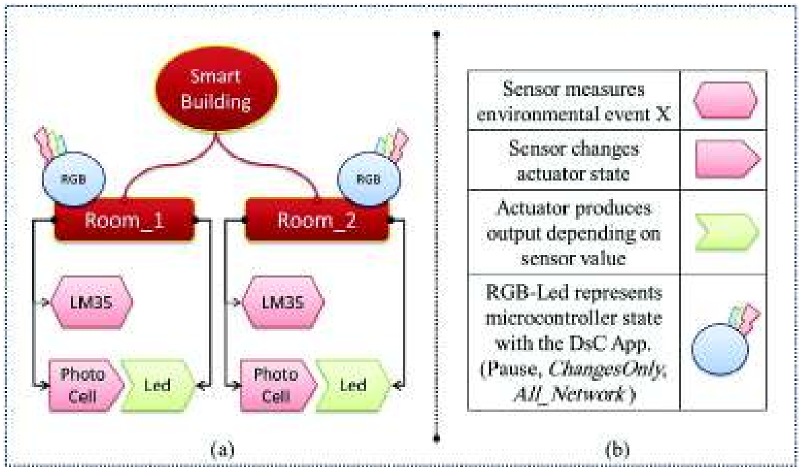
6.7 PROJECT: BUILDING A TESTBED FOR THE WOT

Experiments in the previous section implement main services for the WoT. These services are combined together to build a testbed for the WoT following the architecture in [1] and [2]. Main elements of this architecture are as follows.

1. IoT infrastructure, implementing appropriate network topology (e.g., star topology).
2. Network setup or configuration application: Instead of creating SThs' profiles manually, joined SThs should be discovered dynamically enabling auto generation for their profiles by searching for similar SThs [12,25]. SThs and gateways should speak the same protocol (command messages).
3. WoT web application and APIs: for representing, monitoring and controlling SThs and EoIs. Searching for SThs and EoIs in real-time (dynamic information) using simple query language is the key service in the IoT and the WoT [12].
4. Dataset collector application (DsC): for collecting sensory data in the form of datasets that could be used offline for simulating the WoT. Running the WoT testbed offline using datasets generated from physical environments enhances test results [1] as compared to simulators like Cooja [3].

6.7.1 THE IOT INFRASTRUCTURE

The project presented in this section implements a smart home application for the building architecture shown in Fig. 6.8(a). Component description of the building architecture is shown in Fig. 6.8(b). This project uses simple components indicated



Sensor measures environmental event X	⬡
Sensor changes actuator state	⬠
Actuator produces output depending on sensor value	⬠
RGB-Led represents microcontroller state with the DsC App. (Pause, *ChangesOnly*, *All_Network*)	●

FIGURE 6.8

Smart building structure with sensors and actuators used in the project.

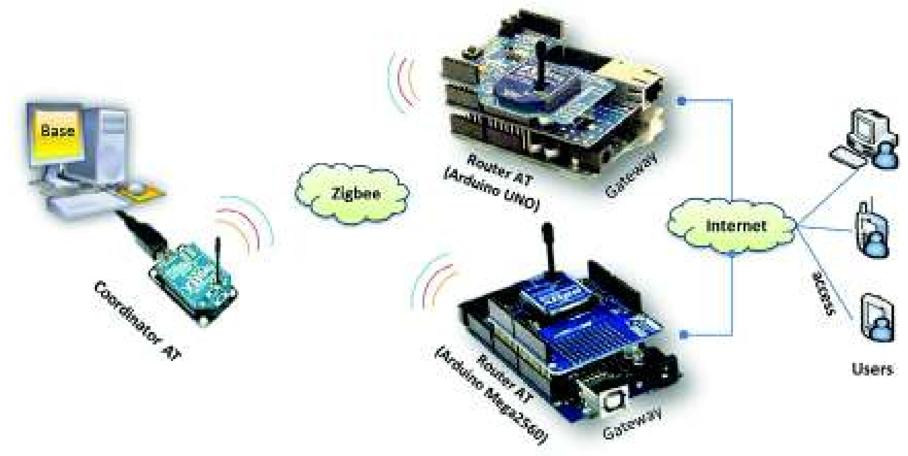


FIGURE 6.9

The IoT network of the project consists of one coordinator and two router devices (router device acts as an end-point device when the IoT implements the star topology).

in Table 6.1. For simplicity, the building has two rooms; each room has a temperature sensor, photocell and LED. Each room is represented by a physical gateway (Arduino microcontroller) and a virtual gateway (web page).

As shown in Fig. 6.9, the IoT network of the project implements a star topology to save the power consumed in message transmission (star topology configures

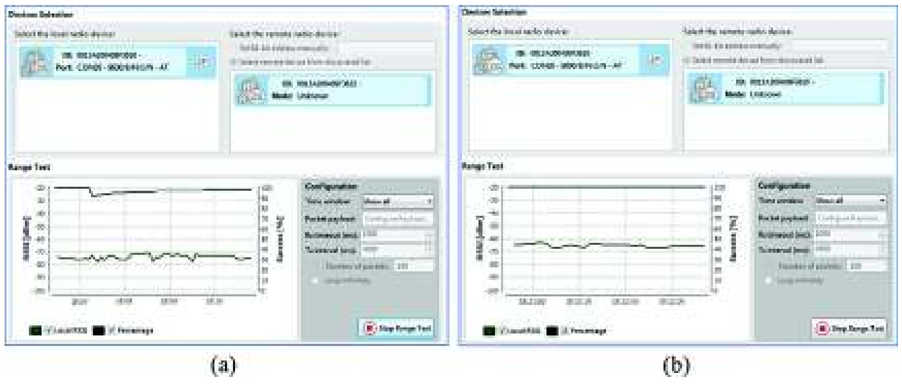


FIGURE 6.10

Measured RSSI values: (a) Arduino Mega2560 and (b) Arduino UNO.

nodes using less number of hops between coordinator and endpoint devices). In this project, the network connects two Arduinos with the coordinator wirelessly using XBee modules and Zigbee connection tools and protocols. The IoT layer consists of three nodes. The first node is of type Arduino-UNO; the second of type Arduino-Mega2560, and the third is a PC. USB-shield connects XBee module to the PC. The PC receives incoming messages wirelessly using Zigbee from Arduino-UNO and Arduino-Mega2560 then stores them in a database. As mentioned above, to implement such a WSN, Arduino-UNO and Ardunio-Mega2560 are configured as routers or end-point devices, and the PC is configured as a coordinator (base-station).

Device firmware configurations of this project are available for download [49]. Arduinos are distributed after testing Received Signal Strength Indicator (RSSI) [50, 51]. RSSI was measured in decibels using Digi's X-CTU software to determine signal noise, and its value ranges from 0 to 120; the closer it is to zero, the stronger is the signal. RSSI for Arduinos is shown in Fig. 6.10.

6.7.2 NETWORK PROTOCOLS

Internal protocols are written for managing STs work in the IoT network (e.g., sending and receiving messages). The coordinator, routers, and end-point devices speak the same language (command messages). These commands are sent by the coordinator using DsC as in [1,2]. The list of the common messages is shown in Table 6.2.

In the project, Arduinos represent EoIs (rooms) and act as physical gateways; each Arduino is attached with two sensors (LM35 and LDR) and two actuators (LED and RGB-LED) in addition to XBee module that is connected to Arduino using XBee shield component as shown in Fig. 6.1. The Arduino circuit and a snippet of its code is shown in Fig. 6.11. This code is for the loop() function of the Arduino code. The loop() function is organized into three parts. In the first part (part 01 in Fig. 6.11), Arduino listens to all available requests (e.g., *GET*) coming from clients on the Internet

Table 6.2 List of protocol commands between Arduinos and DsC

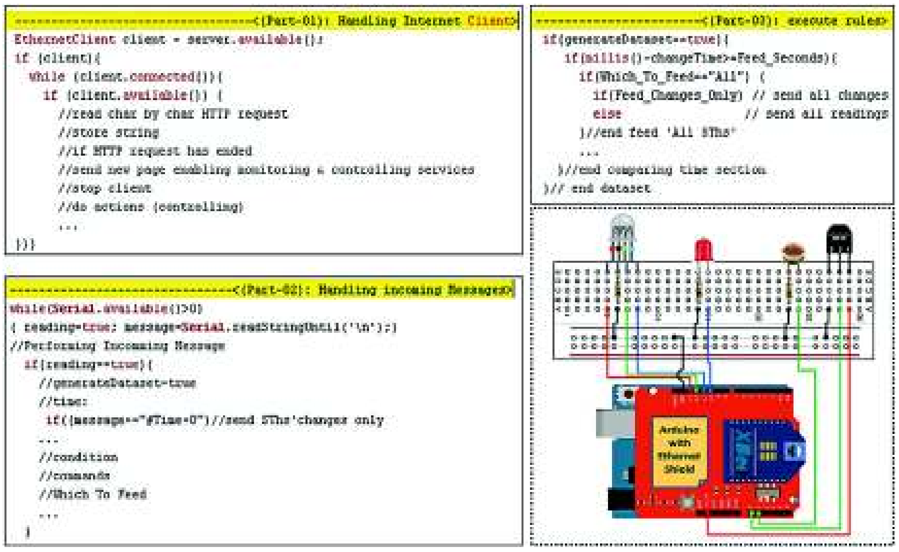
Protocol command	Description
#Who	For discovering all connected gateways in the network, whereby they reply with a message containing their ID_Name_Type.
#Open, #Close	To turn on/off a session between DsC and gateways
#Led	Check all Connected Leds on Arduinos and send back feeds about their states
#All	Feedback about all connected devices on selected Arduinos
#Temp_on, #Temp_off	Turn on/off dependencies between temperature sensors and other devices
#LDR_on, #LDR_off	Turn on/off dependencies between Photocell sensors and other devices
#Feed_Temp_on, #Feed_Temp_off	Turn on/off feeds from temperature sensors.
#Feed_LDR_on, #Feed_LDR_off	Turn on/off light sensor (photo cell) feeds.
#Feed_Changes_Only_on	To set Arduino on change mode where it feeds information only whenever a change occurs in its objects (push method).
#Feed_Changes_Only_off	Do not feed information when a certain change occurs (poll method).
#Time = 1000	Time interval to feed their information back to DsC.
#Quick, #Normal	To initiate mode in which Arduinos send and receive signals

#Conditions_on = #Temp_on + #LDR_on (to fire all dependencies between devices).

and serves them by sending back its HTML page containing SThs readings. In the second part (part 02 in Fig. 6.11), Arduino receives and handles incoming messages (commands) from the DsC. In the third part (part 03 in Fig. 6.11), Arduino prepares all SThs readings following the configurations that are received from the DsC. In the case of the command message *All_Network*, Arduino sends all current SThs' readings, while in the case of *ChangesOnly*, Arduino sends all currently changed readings of the SThs).

6.7.3 WEB APPLICATION AND WEB SERVICES

The WoT is considered the application layer, which visualizes the IoT data using HTTP protocol and standard RESTful services. Thus, adding WoT layer to the testbed is done in two steps: (1) building the web application and (2) building SThs APIs (web services). The web application of the WoT is built using ASP.net, where each EoI (e.g., room) is represented by a web page which may contain more than one STh's link. Each STh is represented as a web page containing dynamic parts for visualizing its state in real-time (using charts). More details with code segments are in [1,2,46]. A set of web services were written in C#. The web application loads the available



```
------------------------------------<(Part-01): Handling Internet Client>
EthernetClient client = server.available();
if (client){
  While (client.connected()){
    if (client.available()) {
      //read char by char HTTP request
      //store string
      //if HTTP request has ended
      //send new page enabling monitoring & controlling services
      //stop client
      //do actions (controlling)
      ...
}}}
```

```
--------------------------------<(Part-02): Handling incoming Messages>
while(Serial.available()>0)
{ reading=true; message=Serial.readStringUntil('\n');}
//Performing Incomming Message
  if(reading==true){
    //generateDataset=true
    //time:
     if((message=="#Time=0")//send STHs'changes only
    ...
    //condition
    //commands
    //Which To Feed
    ...
  }
```

```
-------------------------<(Part-03): execute rules>
 if(generateDataset==true){
    if(millis()-changeTime>=Feed_Seconds){
      if(Which_To_Feed=="All") {
        if(Feed_Changes_Only) // send all changes
        else                  // send all readings
      }//end feed 'All STHs'
      ...
   }//end comparing time section
 }// end dataset
```

FIGURE 6.11

Arduino circuit and code snippet of its loop function: Arduino circuit represents room_1 in the building architecture.

STHs APIs dynamically. A special tag *GET#* is added as an additional service that is executed by default for the device webpage.

Because it is difficult for traditional web search engines to crawl and index pages containing dynamic information [12,16] (e.g., when pages are coded in AJAX), Google optimization rules [52] will be considered for directing spiders to index default URLs instead of parts coded in AJAX, like the WoTSF in [12]. In this project, the *GET#* service is called instead of the dynamic part. WoTSF [12] locates a server-root-file for each WoT following Google optimization rules [52] to build high-level indices, saving time in crawling and indexing dynamic pages of the WoT. Building special crawlers for WoT pages is another solution, where spiders can execute dynamic parts on the fly and index results (e.g., AJAX crawler [53]).

The main services of the web application are monitoring sensors, controlling actuators, triggering action events, and periodic sensor reporting [29]. The device page loads the RESTful services dynamically using Web Services Description Language (WSDL) [54] according to the Arduino IP and selected device ID. The web application operates in two modes: offline and online [1,2]. In offline mode, it attaches a dataset generated by the DsC (details in the next section) and simulates environmental events using historical information of real STHs, while in online mode, the web application retrieves STHs information directly from the IoT (Arduinos) in real-time.

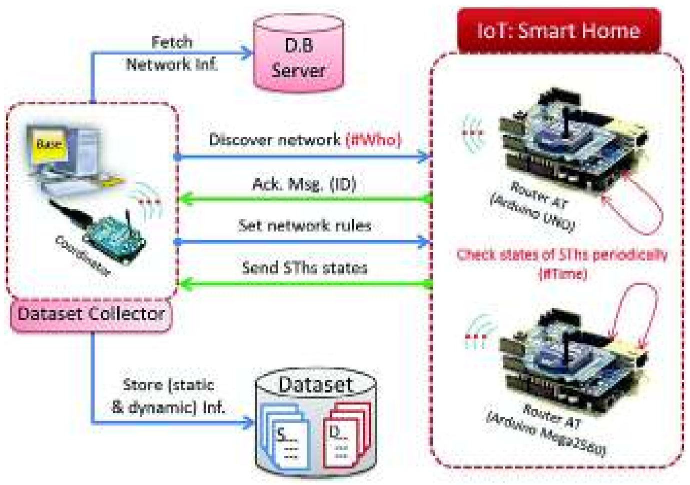


FIGURE 6.12

The architecture of the Dataset Collector Application (DsC).

6.7.4 DATASET COLLECTOR APPLICATION (DSC)

The coordinator (PC attached with XBee modules through a USB-shield) of the WoT testbed acts as a gateway by running the DsC [1,2] application for discovering and controlling SThs in the IoT. DsC generates datasets from discovered gateways by commanding the IoT using different rules, as shown in Fig. 6.12. Gateways periodically check SThs states and send back SThs information to be stored in the base-station; the period length for checking SThs states is controlled by the DsC in the beginning, whereby the DsC sends *#Time* commands to all of them (e.g., *#Time = 1000*) [46]. DsC can store SThs information written in different formats (e.g., Microformat). The main parts of the DsC are as follows.

1. Discovery part: in this part the DsC discovers all known gateways in the IoT network by sending *#Who* command (Table 6.2). It can get a list of the registered gateways from the database, where the network configuration application stores such information about the network.
2. Rule selection part: using this part, users of the WoT testbed instruct gateways about certain types of information and methods (*All_Network* or *ChangesOnly*) according to which gateways send SThs feeds. On the other hand, gateways update their settings in order to send back the required information.
3. Message analyzer part: this part handles and analyzes incoming feeds and stores information written in pre-selected format.
4. Monitoring part: it monitors the contents of the received messages, calculates the time delay of each processed message, and assesses dataset time accuracy of the generated dataset.

The DsC generates two types of information [1]: files that contain static information about building architecture and SThs profiles (i.e., static attributes of the STh) and dynamic information about SThs values and EoIs states. The network configuration application stores static information about the IoT layer (SThs profiles) in a database. DsC retrieves this type of information by implementing search queries on the database. Dynamic information is stored by IoT base-stations; this type of information is retrieved from the historical data or generated directly in real-time by requesting discovered gateways and instructing them to send their feeds following a certain dataset generation rule.

6.8 SUMMARY

In this chapter we presented the main practical knowledge and skills for building WoT applications and WoT testbeds based on a smart home application. Traditional things or daily-life objects and places are converted at first to SThs and EoIs by attaching sensors and actuators to things and attaching or locating more than one STh to represent states of the EoIs.

Building the WoT layer on top of the IoT layer needs to use current web tools and services to visualize SThs values and EoIs states. The IoT layer can have different types of nodes: coordinator, router, and end-device. Zigbee Network configuration is done using X-CTU software. For covering a wide area, building a WSN needs at least one node that acts as a coordinator and the other nodes to act as routers and end-point devices. Concrete steps for building the WoT testbed are discussed in the form of experiments and a mini-project. The experiments implement main WoT services, such as integrating SThs and visualizing their states.

The main parts of the WoT testbed project are as follows.

1. Building the IoT layer:
   - Converting things to SThs and connecting them to gateways (the IoT nodes).
   - Naming SThs: creating profiles for SThs and EoIs using network configuration application or automatically using semantic technology and SThs similarity search tools (only manual naming is presented in the chapter) [12,25].
   - Using or creating a common protocol between gateways and SThs.
   - Configuring gateways (coordinator, router, or end-device) to implement an appropriate topology (e.g., star or mesh).
2. Building the WoT application layer:
   - Building web services (connections between physical and virtual gateways).
   - Building virtual gateways (web pages) in a hierarchical structure following building or environment structure. These web pages call RESTful APIs to monitor and control SThs and EoIs.

- Discovering all available gateways, getting a list of connected devices on each gateway.
- Sending command messages (rules) to instruct the gateways to send back specific information about a specific list of devices according to a specific action or event.
- Listening to router devices following a dataset generation rule (*All_Network* or *ChangesOnly*), and analyzing incoming messages to extract SThs values.
- Storing incoming feeds (SThs states) written in the selected format (e.g., microformat).

REFERENCES

[1] Younan M, Khattab S, Bahgat R. Evaluation of an integrated testbed environment for the web of things. Int J Adv Int Sys 2015;8(3&4):467–82.

[2] Younan M, Khattab S, Bahgat R. An integrated testbed environment for the web of things. In: ICNS 2015: the eleventh international conference on networking and services, Rome, Italy. ISBN 978-1-61208-404-6, May 2015. p. 69–78.

[3] Ostermaier B, Elahi M, Römer K, Fahrmair M, Kellerer W. A real-time search engine for the web of things. In: The 2nd IEEE international conference on the Internet of things (IoT), Tokyo, Japan. Nov. 2010. p. 1–8.

[4] Pfister C. In: Jepson B, editor. Getting started with the internet of things. 1st edn. United States of America: O'Reilly Media; 2011 [Ch. 1].

[5] Blockstrand M, Holm T, Kling L, Skog R, Wallin B. Operator opportunities in the internet of things – getting closer to the vision of more than 50 billion connected devices [Online]: http://www.ericsson.com/news/110211_edcp_244188811_c, 2011.

[6] What is DPWS and uDPWS all about [Online]: http://code.google.com/p/udpws/wiki/IntroductionGeneral, 2016.

[7] Guinard D, Trifa V. From the internet of things to web of things. In: MEAP – building the web of things. 2015. p. 1–15 [Ch. 1].

[8] Haller S. The things in the internet of things. In: Poster at the (IoT 2010), Tokyo, Japan. Nov. 2010.

[9] Elkstein M. Learn REST: a tutorial [Online]: http://rest.elkstein.org/2008/02/what-is-rest.html, 2008.

[10] Hong S. Mobile discovery in a web of things. MSc. Thesis. Swiss Federal Institution of Technology – ETH Zurich; April 22. 2010.

[11] Raggett Dave, W3C. An introduction to the web of things framework. Document [Online]: https://www.w3.org/2015/05, 2015.

[12] Younan M, Khattab S, Bahgat R. WoTSF: a framework for searching in the web of things. In: INFOS2016: the 10th international conference on informatics and systems. Egypt, Cairo: ACM; May 2016.

[13] Muhammad I, Said AM, Hasbulla H. A survey of simulators, emulators and testbeds for wireless sensor networks. In: 2010 international symposium in information technology (ITSim), vol. 2. Kuala Lumpur; June 2010. p. 897–902.

[14] Mayer S, Guinard D, Trifa V. Searching in a web-based infrastructure for smart things. In: Proceedings of the 3rd international conference on the Internet of things (IoT 2012). Wuxi, China: IEEE; October 2012. p. 119–26.

[15] Guinard D. A web of things application architecture – integrating the real-world into the web. PhD Thesis. Zürich: Computer Science, Eidgenössische Technische Hochschule ETH Zürich; 2011.

[16] Jin X, Zhang D, Zou Q, Ji G, Qian X. Where searching will go in Internet of things?. IEEE; 2011.

[17] Gluhak A, Krco S, Nati M, Pfisterer D, Mitton N, Razafindralambo T. A survey on facilities for experimental internet of things research. IEEE Commun Mag 2011;49(11):58–67. http://dx.doi.org/10.1109/MCOM.2011.6069710, inria-00630092.

[18] Miloš J, Zogović N, Dimić G. Evaluation of wireless sensor network simulators. In: The 17th telecommunications forum (TELFOR 2009), Belgrade, Serbia. 2009. p. 1303–6.

[19] Sundani H, Li H, Devabhaktuni VK, Alam M, Bhattacharya P. Wireless sensor network simulators – a survey and comparisons. Int J Comput Netw (IJCN) Feb. 2011;2(6):249–65.

[20] Bodik P, Guestrin C, Hong W, Madden S, Paskin M, Thibaux R. Intel lab data [Online]: http://www.select.cs.cmu.edu/data/labapp3/index.html, 2004.

[21] Gay D, Levis P, Culler D, Brewer E, Welsh M, von Behren R. The nesC language: a holistic approach to networked embedded systems. In: PLDI '03 proceedings of the ACM SIGPLAN 2003 conference on programming language design and implementation, New York, NY, USA. May 2003. p. 1–11.

[22] Levis P, Madden S, Polastre J, Szewczyk R, Whitehouse K, Woo A. TinyOS: an operating system for sensor networks. In: Weber W, Rabaey JM, Aarts E, editors. 1st edn. Berlin, Heidelberg: Springer; 2005. p. 115–48 [Ch. 2].

[23] Thingsquare AS IS. Contiki: the open source OS for the internet of things [Online]: http://www.contiki-os.org/, 2016.

[24] Mayer S, Guinard D. An extensible discovery service for smart things. In: Proceedings of the 2nd international workshop on the web of things (WoT 2011). San Francisco, CA, USA: ACM; June 2011. p. 7–12.

[25] Truong C, Romer K, Chen K. Sensor similarity search in the web of things. In: 2012 IEEE international symposium world of wireless, mobile and multimedia networks (WoWMoM), San Francisco, CA. June 2012. p. 1–6.

[26] Werner-Allen G, Swieskowski P, Welsh M. MoteLab: a wireless sensor network testbed. In: Fourth international symposium on information processing in sensor networks. 2005. p. 483–8.

[27] Nati M, Gluhak A, Abangar H, Headley W. SmartCampus: a user-centric testbed for internet of things experimentation. In: 2013 16th international symposium on wireless personal multimedia communications (WPMC), Atlantic City, NJ. June 2013. p. 1–6.

[28] Mujica G, Rosello V, Portilla J, Riesgo T. Hardware-software integration platform for a WSN testbed based on cookies nodes. In: IECON 2012 – 38th annual conference on. Montreal, QC: IEEE Industrial Electronics Society; October 2012. p. 6013–8.

[29] Nam H, Janak J, Schulzrinne H. Connecting the physical world with arduino in SECE. Computer science technical reports. Columbia University, New York: Department of Computer Science; 2013. p. 6013–8. Technical reporting CUCS-013-13.

[30] Arduino. Arduino.com [Online]: http://www.arduino.cc/, 2015.

[31] LogMeIn Inc. Xively.com [Online]: http://www.Xively.com, Nov. 2014.

[32] Powered by ioBridge. ThingSpeak – the open data platform for the internet of things [Online]: http://www.thingspeak.com, 2016.

[33] XMPro. XMPro internet of things [Online]: http://xmpro.com/xmpro-iot/, 2012.

[34] Digi International Inc. Digi's xctu-software [Online]: http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu.
[35] Arduino. Arduino wireless shield with XBee series 2 radios [Online]: https://www.arduino.cc/en/Guide/ArduinoWirelessShieldS2, 2015.
[36] Kramp T, van Kranenburg R, Lange S. Enabling things to talk: designing IoT solutions with the IoT architectural reference model. Springer; 2013. p. 115–48 [Ch. 1].
[37] Faludi R. In: Jepson B, editor. Building wireless sensor networks. 1st edn. United States of America: O'Reilly Media; 2011. p. 115–48 [Ch. 1].
[38] Vedvei S. XBee wireless communication setup [Online]: https://eewiki.net/display/Wireless/XBee+Wireless+Communication+Setup, Jan. 2014.
[39] Boonsawat V, Ekchamanonta J, Bumrungkhet K, Kittipiyakul S. XBee wireless sensor networks for temperature monitoring. In: The second conference on application research and development (ECTI-CARD 2010), Chon, Buri, Thailand. May 2010. p. 221–6.
[40] Younan M. A wot testbed for research and course projects. Course project package-source codes [Online]: https://github.com/MinaYounan-CS/WoT-CourseProjects, Jan. 2016.
[41] Guinard D, Trifa V. Towards the web of things: web mashups for embedded devices. In: Workshop on mashups, enterprise mashups and lightweight composition on the web (MEM 2009), in proceedings of WWW (international world wide web conferences), Madrid, Spain. Nov. 2009. p. 1–5.
[42] McEwen A, Cassimally H. In: Hutchinson C, editor. Designing the internet of things. 1st edn. John Wiley & Sons; Nov. 2013 [Online]: https://books.google.com.eg/books?id=oflQAQAAQBAJ.
[43] Younan M. STh registration in the WoT. Code [Online]: https://github.com/MinaYounan-CS/WoT_SThRegistration, Jan. 2016.
[44] Datta SK, Da Costa RPF, Bonnet C. Resource discovery in internet of things: current trends and future standardization aspects. In: 2015 IEEE 2nd world forum on Internet of things (WF-IoT), Milan. Dec. 2015. p. 542–7.
[45] Hachem S, Teixeira T, Issarny V. Ontologies for the internet of things. In: ACM/IFIP/USENIX 12th international middleware conference, Lisbon, Portugal. Dec. 2011.
[46] Younan M. An integrated tested environment for the web of things. Source code package [Online]: https://github.com/MinaYounan-CS/WoT-Testbed-Environment, Jan. 2016.
[47] Mainetti L, Mighali V, Patrono L. A software architecture enabling the web of things. IEEE Int Things J Dec. 2015;2(6):445–54. http://dx.doi.org/10.1109/JIOT.2015.2477467.
[48] Digi International Inc. Official XBee website-connect devices to the cloud [Online]: http://www.digi.com/xbee, Jan. 2015.
[49] Younan M. IoT XBee pofiles [Online]: https://github.com/MinaYounan-CS/IoTXBeeProfiles, Jan. 2016.
[50] Wikipedia. Packet transfer delay [Online]: https://en.wikipedia.org/wiki/Packet_transfer_delay, Apr. 2011.
[51] Randolph G, Hirsch N. PIC32MX: XBee wireless round-trip latency [Online]: http://hades.mech.northwestern.edu/index.php/PIC32MX:_XBee_Wireless_Round-trip_Latency, Mar. 2010.
[52] Google. Search engine optimization (SEO) – starter guide. Jan. 2010.
[53] Suganthan PGC. AJAX crawler. In: 2012 international conference on data science & engineering (ICDSE). Cochin, Kerala: IEEE; 2012. p. 27–30.
[54] Wikipedia. Web services description language [Online]: http://en.wikipedia.org/wiki/Web_Services_Description_Language, Nov. 2014.