Heatapp - Remote Temperature Controller Project

Degree Project in Electrical Engineering

Kaan Orgunmat

Supervisor, Sven Johansson
Dept. Electrical Engineering
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden
## Contents

1 Abstract .................................................. 3

2 Introduction ............................................ 4

3 The Problem ............................................. 6

4 Experimental setup ...................................... 6
   4.1 Hardware ............................................. 6
   4.2 Software .............................................. 10
   4.3 Programming Language ............................... 12

5 Results .................................................. 26

6 Summary .................................................. 27

7 References ............................................... 28
1 Abstract

The project is a part of series of hardware and software challenges merging in one. It is fundamentally an application to monitor the temperature changes and get notifications based on those changes. Also through this application an electrical device that is connected to an outlet will be controlled remotely. It is a feature for the users that do not have access to certain type of devices and want to turn on/off them.

This project is built by using a digital remote controller and an outlet (REMOTE CONTROLLED OUTLETS 310000) that a potential heating system is connected to. The remote controller is controlled via Arduino. On Arduino there are two relays (PRMA2A12B) that turns the remote control on and off based on the input signal that is sent.

There is also a temperature sensor connected to the Arduino. The readings from the temperature sensor are sent to a web server and stored there in order to track the changes. On the web interface, other than read and tracking temperature changes, it is possible to turn on/off an outlet with an integrated button.
2 Introduction

This project has its roots from my studies in Ireland. After participating in a competition for an application idea and became a finalist with it in the Cork Institute of Technology, the idea carried on, re-formulated and integrated in my Electrical Engineering degree project and took its last form as the Heatapp project today.

This project can be considered as a part of the greatly evolving The Internet of Things world. It is the compilation of the network of physical objects embedded with electronics, software, sensors and connectivity to achieve intriguing services by using data with connected devices.

The project will cover the basis of this application’s functionalities, components and their way of working all together as a whole.

There are several variations of this application. The remote controlling part is more leaning towards the central heating systems and how they are monitored. For example, British Gas is an energy and home services provider in the United Kingdom. Britain’s biggest energy supplier in the UK serving around 10 million homes. They offer a Remote Heating Control as a standalone product.

It is called AlertMe[3], the Smart Home technology company, has its online and Smartphone-enabled Remote Heating Control service available in the UK exclusively from British Gas. AlertMe platform, which allows consumers to monitor, control and automate a range of devices in the home via a single user interface.

Focusing on the most viable product, the project scope was determined in a way that it required some alterations of home heating and car heating systems.

Initial thoughts were;

• Temperature checks and changes in unattended/remote locations
• Notifications in case of unexpected temperature changes
• Control over the energy usage:
  – Turn the heating on or off
  – Override all heating settings
  – Lower energy bills and reduce carbon footprint
Figure 1: Early block diagram of the whole project

- Controls supported vehicles’ temperatures
- Controls the water heating system
- Calendar collaborated control system for any prospective events/vacations
- Temperature adjusting GPS system that follows users daily routines
- Controls Air Conditioner

Of course this scope would not be a viable option as it requires extensive research, time and project partners.
3 The Problem

Having an instant temperature monitoring system through a mobile phone seemed to be either expensive or impractical. There are different applications like La Crosse Alerts D111.101.E1.WGB Wireless Monitor System Set with Dry Probe (105 U.S Dollars) that only monitors the temperature changes or Climote that offers temperature controlling that goes up to 400 Euros[4]. This project aimed to offer easy control with a more practical and maybe cheaper setting.

To achieve this solution, an Arduino with an Ethernet Shield, a digital temperature sensor and a remote controller with an outlet has been purchased. The total cost was around 90 U.S Dollars.

4 Experimental setup

4.1 Hardware

- Arduino Uno R3
- DS18B20 Dallas Temperature Sensor
- A 4.7k Ohm Resistor
- Arduino Ethernet Shield
- PRMA2A Relay
- Proove Start-up Kit Basic TSP200 *TSP200 Remote and TSR100 Outlet*
- Router connected to a network
- Laptop connected to a router

Arduino Uno R3

The Arduino Uno (see: Figure 1) is a microcontroller board based on the ATmega328 (The high-performance Atmel 8-bit AVR RISC-based microcontroller[6]). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It is programmed via a computer by connecting it through a USB cable or power it with a AC-to-DC adapter or battery. It is programmed as a USB-to-serial converter.
Arduino Ethernet Shield

The Arduino Ethernet Shield (see: Figure 2) requires the Arduino Board and operates on 5V voltage that is supplied from the Arduino Board. Ethernet controllers properties are; W5100 with internal 16K buffer and the connection speed is 10/100Mb[7]. The Ethernet Shield is connected to the Arduino main board from above. It allows the temperature sensor to communicate with the web server. An RJ45 cable is connecte between an in-house-modem and to the Ethernet shield.

![Arduino Uno with the Ethernet Shield](image)

Figure 2: Arduino Uno with the Ethernet Shield

Dallas Temperature Sensor - DS18B20

The one wire Digital Temperature Sensor - DS18B20 (see: Figure 4) is a programmable chip for measuring temperatures. It also offers a Dallas Temperature library for the Arduino which makes using this sensor very practical (see: Figure 2).

The DS18B20 digital thermometer reports 9 to 12-bit Celsius (C) temperature measurements. It has an operating precision range of -55C to +125C and is accurate to 0.5C over the range of -10C to +85C. In addition, the DS18B20 can derive power directly from the data line (parasite power, 1-Wire protocol), eliminating the need for an external power supply[2].

1-Wire Protocol

1-Wire Protocol is a protocol produced by Dallas Semiconductor. On a 1-Wire network, a single device communicates with one or more 1-Wire devices over a single data line (see: Figure 5), which can also be used to provide power to the secondary devices[1]. (parasitic power mode) It is an inexpensive and easy to use method and provides accurate digital temperature readings directly.
4.7k Ohm Resistor (Parasite power mode) When operating in parasite power mode, only two wires are required: one data wire, and ground. In this mode, the power line must be connected to ground. At the master, a 4.7k pull-up resistor must be connected to the 1-wire bus. When the line is in a “high” state, the device pulls current to charge an internal capacitor.
PRMA2A Relay

A relay is an electrically operated switch. In this project PRMA2A relay is used as a switch control (see: Figure 6). A set of one or more contacts allow to open or close an independent electrical circuit. There are two relays connected between Arduino to Proove remote controller in order to turn on and off the outlet through the signal that is sent from the web server.

Proove Start-up Kit Basic TSP200 (TSP200 Remote and TSR100 Outlet)

This is an energy management kit produced by Proove AB. The remote control operates at a distance up to 30 metres with free line of sight. Its powered with a CR2032 -a button cell lithium
battery-rated at 3.0 volts.

Figure 7: Outlet and Remote

The kit’s remote controller allows the communication between the Arduino and the kit’s outlet by turning it on/off. TSP200[9] remotes casing has been removed and soldered on a board (see: Figure 8) in order to command from the Arduino directly. There are two relays between the Arduino and the remote controller in order to send the on/off signals to the TSP100[8] outlet. After the Arduino integration, the connection between the remote controller and the outlet is established hence the web interface is connected to the system to have a remote access to the turning on/off function.

4.2 Software

- Arduino version 1.6.3 (C / C++)
- Adobe Photoshop CS6
- Invision Prototyping Tool

The project is to be improved by developing an iOS application in the future. There is already an existing UX (User Experience)/UI (User Interface) phase designed and a functional visual prototype of this project in the following link; http://bit.do/5D66
Figure 8: Arduino, Remote Controller, Relays and Temperature sensor connections

Figure 9: Adobe Photoshop design of the mobile app
4.3 Programming Language

- Arduino Programming (C / C++)
- Php
- HTML and CSS
- Javascript / jquery

Arduino Software and Programming

The Arduino integrated development environment (IDE) is a cross-platform application the programming languages are written in C or C++. The Arduino comes with libraries and in this project, OneWire and Dallas libraries are used.

Arduino Codes

```c
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <stdlib.h>
#include <stdio.h>

// Starting to setup the ethernet sensors

// assign a MAC address for the ethernet controller.
// fill in your address here:
byte mac[] = {0x90, 0xA2, 0xDA, 0x0D, 0xEA, 0x73};
```
// initialize the library instance:
EthernetClient client;

char inString[32]; // string for incoming serial data
int stringPos = 0; // string index counter
boolean startRead = false; // is reading?

char server[] = "jimmi.eu";

unsigned long lastConnectionTime = 0;
// last time you connected to the server
boolean lastConnected = false;
// sstate of last time with the main loop
const unsigned long postingInterval = 30*1000;
// delay between updates, in milliseconds

// on off remote control
int off = 12;
int on = 13;
//pinMode(off, OUTPUT);  // sets the digital pin as output
//pinMode(on, OUTPUT);

// Starting to setup the Temp sensors
// ------------------------------------------
char temperature[4];
unsigned long lastTemperatureTime = 0;
// last time you checked the temperature, in milliseconds
// Data wire is plugged into port 2 on the Arduino
#define ONE_WIRE_BUS 2

// Setup a oneWire instance to communicate with any OneWire devices
// (not just Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);

void setup() {
// start serial port:
Serial.begin(9600);
// Start up the sensors library
sensors.begin();
// give the ethernet module time to boot up:
delay(1000);
// start the Ethernet connection using a fixed IP address and DNS server:
Ethernet.begin(mac); // myDns
// print the Ethernet board/shield’s IP address:
Serial.print("My IP address: ");
Serial.println(Ethernet.localIP());
}

void loop() {

// if there’s incoming data from the net connection.
// send it out the serial port. This is for debugging
// purposes only:
/*
String response = ""
if (client.available()) {
response += client.read();
}
Serial.print(response);
*/

// if there’s no net connection, but there was one last time
// through the loop, then stop the client:
if (!client.connected() && lastConnected) {
Serial.println();
Serial.println("disconnecting.");
client.stop();
}

// if you’re not connected, and ten seconds have passed since
// your last connection, then connect again and send data:
if(!client.connected() && (millis() - lastConnectionTime > postingInterval)) {
if(httpRequestConnect()) {
checkTemperature();
httpRequestActive();
}
}
// store the state of the connection for next time through the loop:
lastConnected = client.connected();
}

void checkTemperature() {
Serial.print("Checking temperature: ");
sensors.requestTemperatures(); // Send the command to get temperatures
dtostrf(sensors.getTempCByIndex(0),4,2,temperature);
// note the time that the temperature was checked:
lastTemperatureTime = millis();
Serial.println(temperature);
}

bool httpRequestConnect() {
Serial.print("connecting to remote server.. ");
// if there’s a successful connection:
if (client.connect(server, 80)) {
Serial.println("connected.");
return true;
}
else
{
// if you couldn’t make a connection:
Serial.println("connection failed, disconnecting.");
client.stop();
return false;
}

String readPage(){
//read the page, and capture & return everything between '<' and '>'

stringPos = 0;
memset( &inString, 0, 32 ); //clear inString memory

while(true){

if (client.available()) {
char c = client.read();
if (c == '{') { //'<' is our begining character
startRead = true; //Ready to start reading the part
else if(startRead){

    if(c != '}'){ //'>' is our ending character
        inString[stringPos] = c;
        stringPos ++;
    }else{
        //got what we need here! We can disconnect now
        startRead = false;
        client.stop();
        client.flush();
        Serial.println("disconnecting.");
        return inString;
    }
}

// this method makes a HTTP connection to the server:
void httpRequest() {
    // Setup the data to send
    Serial.println(temperature);
    String data;
    data="";
    data="temperature=";
    data+=temperature;
    data="&host=arduino";
    // send the HTTP POST request:
    client.println("POST /heatapp/temperature HTTP/1.1");
    client.println("Host: jimmi.eu");
    client.println("User-Agent: Arduino/1.0");
    client.println("Content-Type: application/x-www-form-urlencoded");
    client.println("Connection: close");
    client.print("Content-Length: ");
    client.println(data.length());
    client.println();
    client.print(data);
    client.println();
    // debug:
Serial.println("POST /heatapp/temperature HTTP/1.1");
Serial.println("Host: jimmi.eu");
Serial.println("User-Agent: Arduino/1.0");
Serial.println("Content-Type: application/x-www-form-urlencoded");
Serial.println("Connection: close");
Serial.print("Content-Length: ");
Serial.println(data.length());
Serial.println();
Serial.print(data);
Serial.println();
// note the time that the connection was made:
lastConnectionTime = millis();

}

void httpRequestActive() {
  // Setup the data to send
  Serial.println(temperature);
  String data;
  data+="";
  data+="device=arduino";
  // send the HTTP POST request:
  client.println("POST /heatapp/active HTTP/1.1");
  client.println("Host: jimmi.eu");
  client.println("User-Agent: Arduino/1.0");
  client.println("Content-Type: application/x-www-form-urlencoded");
  client.println("Connection: close");
  client.print("Content-Length: ");
  client.println(data.length());
  client.println();
  client.print(data);
  client.println();
  // note the time that the connection was made:
  lastConnectionTime = millis();
  String response = "";
  Serial.println("get response");
  response = readPage();
  Serial.println("get response done.");
  char responseCharArray[100];
  Serial.println(response);
response.toCharArray(responseCharArray,100);
if(strstr( responseCharArray, "active") && strstr( responseCharArray, "1")) {
  Serial.println("should be active");
digitalWrite(off, HIGH);
delay(250);
digitalWrite(off, LOW);
}
else if(strstr( responseCharArray, "active") && strstr( responseCharArray, "0")) {
  Serial.println("should be inactive");
digitalWrite(on, HIGH);
delay(250);
digitalWrite(on, LOW);
}

char * floatToString(char * outstr, double val, byte precision, byte widthp){
  char temp[16]; //increase this if you need more digits than 15
  byte i;

  temp[0]='$'0';
  outstr[0]='$'0';

  if(val < 0.0){
    strcpy(outstr,"-\0"); //print "-" sign
    val *= -1;
  }

  if( precision == 0 ) {
    strcat(outstr, ltoa(round(val),temp,10)); //prints the int part
  }
  else {
    unsigned long frac, mult = 1;
    byte padding = precision-1;

    while (precision--)
      mult *= 10;

    val += 0.5/(float)mult;       // compute rounding factor

    strcat(outstr, ltoa(floor(val),temp,10)); //prints the integer part without rounding
    strcat(outstr, ".\0"); // print the decimal point
  }
}
frac = (val - floor(val)) * mult;

unsigned long frac1 = frac;

while(frac1 /= 10)
    padding--;

while(padding--)
    strcat(outstr,"0\0"); // print padding zeros

strcat(outstr,ltoa(frac,temp,10)); // print fraction part
}

// generate width space padding
if ((widthp ! = 0) && (widthp >= strlen(outstr))){
    byte J=0;
    J = widthp - strlen(outstr);

    for (i=0; i < J; i++) {
        temp[i] = ' ';
    }

    temp[i++] = '\0';
    strcat(temp,outstr);
    strcpy(outstr,temp);
}

return outstr;
}

end of codes
**HTML and CSS**

HTML commonly known as Hypertext markup language is used to create web pages. A web browser can read html files and can compose them into audible or visible web pages. Html uses tags to display the content of the webpage but are not shown on the web page. HTML elements are the building blocks of all webpages. It has built in quality to embed different scripts such as javascript that effects the behavior of html.

CSS (Cascading style sheets) defines the layout or look of a web page in terms of text, pictures, colours etc.

**PHP**

PHP is server side scripting language designed mainly for web development but it is also used as general purpose programming language. PHP code can be mixed with html code. PHP can also be used in combination with different web frameworks. In this project PHP script is used to execute the code to control the turn on and off the machine. The PHP also reads the data of the temperature changes.

**Javascript / jquery**

Javascript is another computer programming language. It is mostly commonly used as a part of web browser. Javascript is also a prototype-based scripting language. Its syntax was influenced by C. In this case most of the programming was done in jquery and javascript.

By using these languages, a webserver is created in the LAN and by using the IP to access the webserver through a browser, web-interface can be seen. After that it shows a webpage, a current and past temperature readings screen and an on/off button can be seen. By clicking on Turn On (1) or Turn Off (0) the script changes and the arduino reads that information and turns the outlet on/off.

**Server and Arduino connection**

1) Arduino connects with internet, establish IP address from DNS. After getting IP address (of local network) opens a connection with the web server.

2) Checks the temperature readings. Sends a POST call to server with the readings and disconnects. It repeats this process every 30 seconds.
3) PHP side accepts incoming connections. When it receives a call with the signature of "temperature" it checks if it contains the host and temperature values. The temperature is a float value. If the value is not null, it stores the data in the server. A POST call is expected but also arranged in a way to accept GET calls. GET calls are visible as a URL string. POST calls are send in the background of the request. A framework that is called "fatfree" is used here. It is a lightweight php framework, where you can build fast dynamic web applications.

4) There are different routes setup in the PHP. Default route shows the webpage. Another route registers the temperature, which does most of the work. It checks for the temperature and the host values and stores them on the server. The third route gives you the history of the temperature readings.

5) It has been a struggle to get the POST calls to work. Sometimes there were delays between the server and the outlet and this often lead a change to GET calls instead.

The PHP code that connects the Arduino with the server:

```php
<?php

// Kickstart the framework
$f3 = require('vendor/bcosca/fatfree/lib/base.php');
// Kickstart parse
require('vendor/parse/php-sdk/autoload.php');
use Parse\ParsePush;

$f3->set('DEBUG',1);

// Load configuration
$f3->config('config.ini');

$logger = new Log('incomming.log');
$logger->write('Incomming request from ' . $_SERVER['REMOTE_ADDR'], 'r');
$logger->write(json_encode($_POST), 'r');

function set_rest_headers() {
    date_default_timezone_set('europe/stockholm');
    header('Access-Control-Allow-Origin: *');
```
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');
header( 'Last-Modified: ' . gmdate( 'D, d M Y H:i:s' ) . ' GMT' );
header('Pragma: no-cache');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Content-Type: application/x-www-form-urlencoded');

function get_db_connection($f3) {
    return new DB\SQL('mysql:host=' . $f3->get('db.host') . ';
    ;port=' . $f3->get('db.port') . ';
    ;dbname=' . $f3->get('db.name'),
    $f3->get('db.user'), $f3->get('db.password'));
}

// this is the main route, that you'll see if you visit the site.
$f3->route('GET /',
    function($f3) {
        $f3->set('content','temperature.html');
        echo View::instance()->render('layout.html');
    });

// this route is used to get active status of device
$f3->route('POST /active',
    function($f3) {
        set_rest_headers();
        $db = get_db_connection($f3);

        if(is_null($_POST['device']))
        {
            echo json_encode(array(
                'success' => false,
                'error' => 'Provide a 'device.'
            ));
            die();
        }

        $result = $db->exec('SELECT * FROM device_settings WHERE device = :device', array(
            ':device' => $_POST['device']
        ));
echo json_encode(array(
    'success' => $result ? true : false,
    'active' => $result[0][active]
));

// this route is used to update device settings.
$f3->route('POST /settings',
    function($f3) {
        set_rest_headers();
        $db = get_db_connection($f3);

        if( ! is_null($_POST['device']) && ! is_null($_POST['settings']))
            {
            $result = $db->exec('SELECT * FROM device_settings WHERE device = :device', array(
                ':device' => $_POST['device']
            ));

            if(count($result) != 1)
                {
                echo json_encode(array(
                    'success' => false,
                    'error' => 'Device not found.'
                ));
                die();
            }

            // merging incomming data with current settings.
            $settings = array_merge($result[0], $f3->get('POST.settings'));

            // saving it.
            $result = $db->exec('UPDATE device_settings SET active=:active WHERE device = :device', array(
                ':device' => $settings['device'],
                ':active' => $settings['active']
            ));

            // output result.
            echo json_encode(array(
                'success' => $result ? true : false
            ));
        }
} else {

    // output error
    echo json_encode(array(
        'success' => false,
        'error' => 'Provide a 'device' and 'settings' array values.'
    ));
}

// this route is used to post new temperature values to database,
// and also send push notification to phone if desired.
$f3->route('POST /temperature',
    function($f3) {
        set_rest_headers();
        $db = get_db_connection($f3);

        if( ! is_null($_POST['host']) && ! is_null($_POST['temperature'])) {
            $result = $db->exec('INSERT INTO temperature VALUES(:host,:ip,:datetime,:temperature)', array(
                ':host' => $_POST['host'],
                ':ip' => $_SERVER['REMOTE_ADDR'],
                ':datetime' => date('Y-m-d H:i:s'),
                ':temperature' => $_POST['temperature']
            ));

            // if to hot or cold, send push request to ios.
            if($f3->get('POST.temperature') > 30) {
                // Push to Channels
                \Parse\ParseClient::initialize($f3->get('parse.app_id'), $f3->get('parse.rest_key'), $f3->get('parse.master_key'));
                ParsePush::send(array(
                    "channels" => [ $f3->get('POST.host') ],
                    "data" => array( 'temperature' => $f3->get('POST.temperature') )
                ));
            }
        }

        // output result
        echo json_encode(array(
            'success' => $result ? true : false
        ));
    });
5 Results

During the project work, there has been several failures and successes. Several sensors got burnt, the webserver-arduino connection took a long time to establish but finally it all worked out.

Addressing the best method to turn on/off a remote outlet seemed difficult initially until the Proove Start-Up Kit (see: Figure 7) took its role in the setting.

By using the indicated languages, a webserver is created in the LAN and by using the IP to access the webserver (can be reached at www.jimmi.eu/heatapp) through a browser, an web-interface (See:
Figure 10) can be seen. After that it shows a webpage, a current and past temperature readings screen and an on/off button can be seen. By clicking on 'Turn on' or 'Turn off' the script changes and the arduino reads that information and sends the signal that turns the outlet on/off.

6 Summary

Being able to touch the basis of The Internet of Things world has been an exciting experience. With this project, I was only able to scratch the surface of that world. Having the project functioning as planned and combining the network of physical objects embedded with electronics and computer languages gave an in depth insight.

As mentioned in the report there are future plans to develop further on this project, such as the iOS application which requires Objective C programming and an Apple account. All in all, this project has been a good analysis and progress of my understanding of many different disciplines of electronics and programming languages.
7 References