6-7-2013

The blue plug : a power consumption regulation system

Loquen Jones
Santa Clara University

Sahil Verma
Santa Clara University

SANTA CLARA UNIVERSITY

COMPUTER SCIENCE AND ENGINEERING

Date: June 7, 2013

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER
MY SUPERVISION
BY

Loquen Jones and Sahil Verma

ENTITLED

The Blue Plug

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE
DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND
ENGINEERING

The Blue Plug: A Power Consumption Regulation System

By

Loquen Jones and Sahil Verma
(BS Computer Science and Engineering)

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Computer Science and Engineering

Thesis Advisor: 

Department Chair: 

School of Engineering
Santa Clara University
Santa Clara, California
June 7, 2013

Abstract

Our goal is to develop a system that reduces peak grid loads by 60%, which saves consumers and businesses money as well as protecting the environment by eliminating the use of fast start up generators. We accomplish this by implementing Arduinos --which are versatile microcontrollers-- equipped with networking technology that allows communication between houses in a neighborhood. Through this communication, we can schedule when appliances are used to avoid everyone using their appliances at the same time and we can eliminate the need for such exorbitant energy usage during peak hours. Our work implements ongoing research that is looking into cost-effective and environmentally friendly grid management.

Acknowledgements

We would like to acknowledge Dr. Amer for letting us continue this project from last year. Without his prior research and that of his colleagues none of our work would have been possible. We would also like to acknowledge Arun Koshy, Anirudh Rao and Siddharth Parmar. What we accomplished would not have happened with out the work they did last year for their senior design project.

Contents

1. Introduction

1.1 Current Grid System

No matter where electricity is used across the world, there is a common fact: it must be used immediately after it is generated. This causes logistical problems for energy companies trying to predict the amount of energy required at any given time, especially during peak hours, where most families will consume the most energy. During these peak hours, energy companies must accommodate extra load by running start-up generators, which tend to be coal and oil based. These methods prove inefficient, consequently burning fossil fuels in an unsustainable manner. Not only are these plants less efficient, they are also more expensive. So energy companies have not only an environmental benefit to reducing these peak-hour loads but also a monetary one.

A system that addresses this issue of peak hour energy production would present both energy providers and consumers with financial benefits as well as reduce pollution and greenhouse gas production. The current system for dealing with peak hours revolves around the energy companies charging consumers more for peak hour usage in the hopes of encouraging users to avoid adding to an already full energy load. What occurs is that consumers all try to avoid the peaks but they ended up just moving the peak hours around. This plan to regulate energy from a single point is not viable in our energy demanding world. When an entire grid goes down, houses and businesses are left powerless. These power outages cost Americans over $150 billion a year.

1.2 Product Overview

Our proposed solution, The Blue Plug, allows power regulation to be implemented in a distributed fashion, which reduces peak load and eliminates the possibility of major failures, thus increasing overall durability of our system. The Blue Plug eliminates the need for environment-killing rapid start-up generators by scheduling energy usage in a distributed fashion that stops intensive appliances from running at the same time. This regulation avoids peak hours by scheduling tasks in a smarter, more efficient way, according to the algorithm being developed for the Blue Plug system. The Blue Plug will be a peered network, distributing computing schedule information throughout a network of homes. This decentralization will increase the overall reliability and security of the system. A saboteur will not have the ability to take out the central control unit of a power grid. Instead the attacker will have to go after each node in the network, which would shut down a very small portion of the system. This increased reliability means that power outages are less likely to happen, so both businesses and households will benefit.

We will be adding support for ad-hoc networking and mobile interfaces will be our primary goal. By using ad-hoc networking we will make it easier to add and remove nodes from the system. These dynamic features will make the Blue Plug easier to use which will lead to widespread use. If the grid were only five percent more efficient, the energy savings would be equivalent to permanently eliminating 53 million cars from the road. The Blue Plug could possibly reduce energy costs by sixty percent saving families and businesses money while also helping the environment.

The Blue Plug system will consist of a hardware component composed of an Arduino board, a XBee radio shield and a WiFi shield. There will also be software running on the Arduino that will perform all of the scheduling of appliances. The beauty of the Blue Plug is its ability to be implemented on top of the existing power grid. This means that the power companies do not have to change how they distribute their energy; the Blue Plug simply gives users the option to schedule their appliances more efficiently. Even though the power companies don't have to change their power distribution they would see the same benefits from this smarter energy management as the individual users of the system.

2. Product Requirements

In order to implement our system and ensure its effectiveness we came up with the following functional and nonfunctional requirements:

The first functional requirement is that the user will be able to connect their Android device to the system through a USB connection. Once connected to the system they will be able to view all current appliances and select individual ones they wish to edit. They will be able to schedule an appliance to start and finish at a specified time, as well as specifying what days they want the appliance to run. The user will be able to start an appliance immediately and override existing scheduling. The Blue Plug system as a whole will be able to dynamically add new appliances over Wi-Fi (provided that the technology exists and is available). The Blue Plug will also be able to set up an ad-hoc network between other Blue Plugs in the area. These ad-hoc networks will be able to communicate remotely, sharing scheduling information provided by users, through the use of RF radios. They must be able to turn appliances on and off by implementing the algorithm developed by Dr. Amer and his colleagues.

The non-functional requirements were not technical features, but rather ideas about how the user would interact with the system and the overall user experience. The system had to be intuitive and easy to use. The system also had to provide a simple way for users to interact with their appliances, without the hassle of scheduling tasks by themselves.

3. System Overview

This section provides an overview of how the end-user interacts with the Blue Plug system as well as the main functions of the Blue Plug system as a whole.

3.1 Architecture

First we will go over the overarching architecture of the Blue Plug System. The following architectural diagram gives a high level view of our complete system. It is split into two parts, covering both the interactions between multiple Blue Plug systems as well as what goes on within a house using a single Blue Plug device. We will show where the users come in using the Android application we developed to schedule their appliances as well.

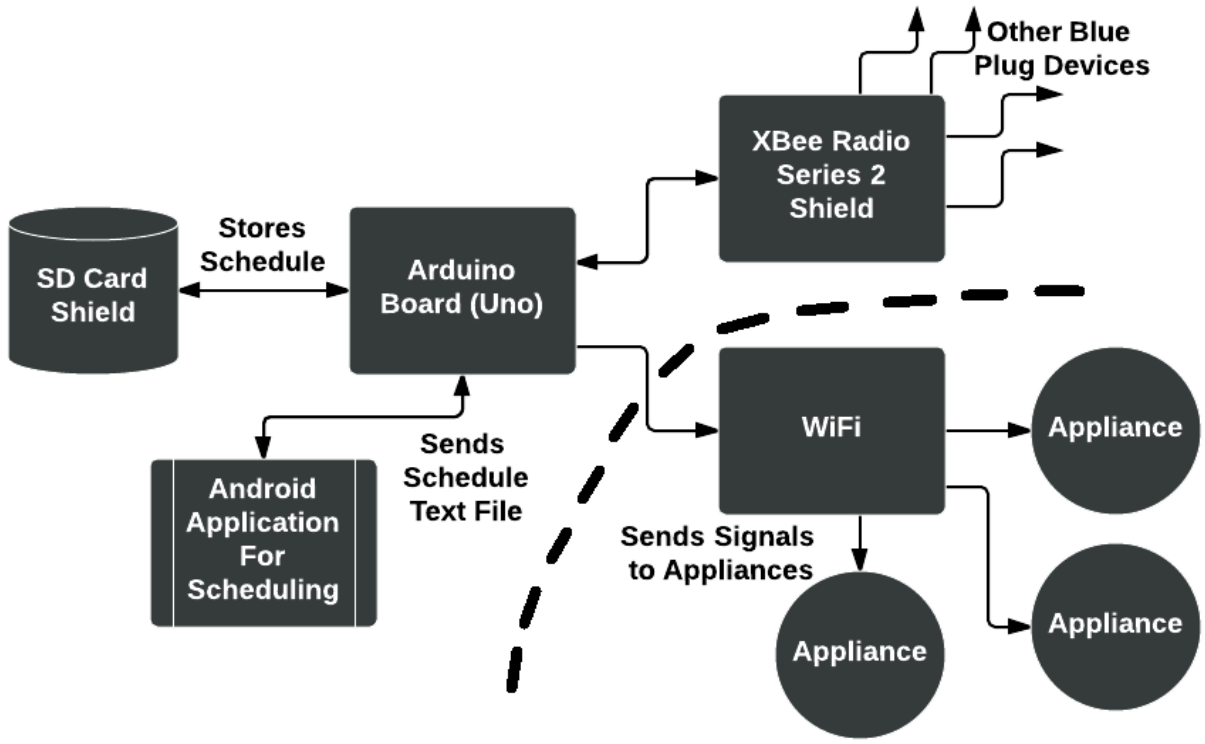

Figure 1 *Architectural Diagram*: Dotted line illustrates what is outside the scope of our project

The ability to distribute multiple schedules would seem like it might create a lot of overhead data that each Blue Plug system must deal with. In reality each Blue Plug's own schedule will just have more events on it. They won't actually be adding more than one schedule; they will just fill in more timeslots of their existing schedule so that scheduling can be done more efficiently as a neighborhood.

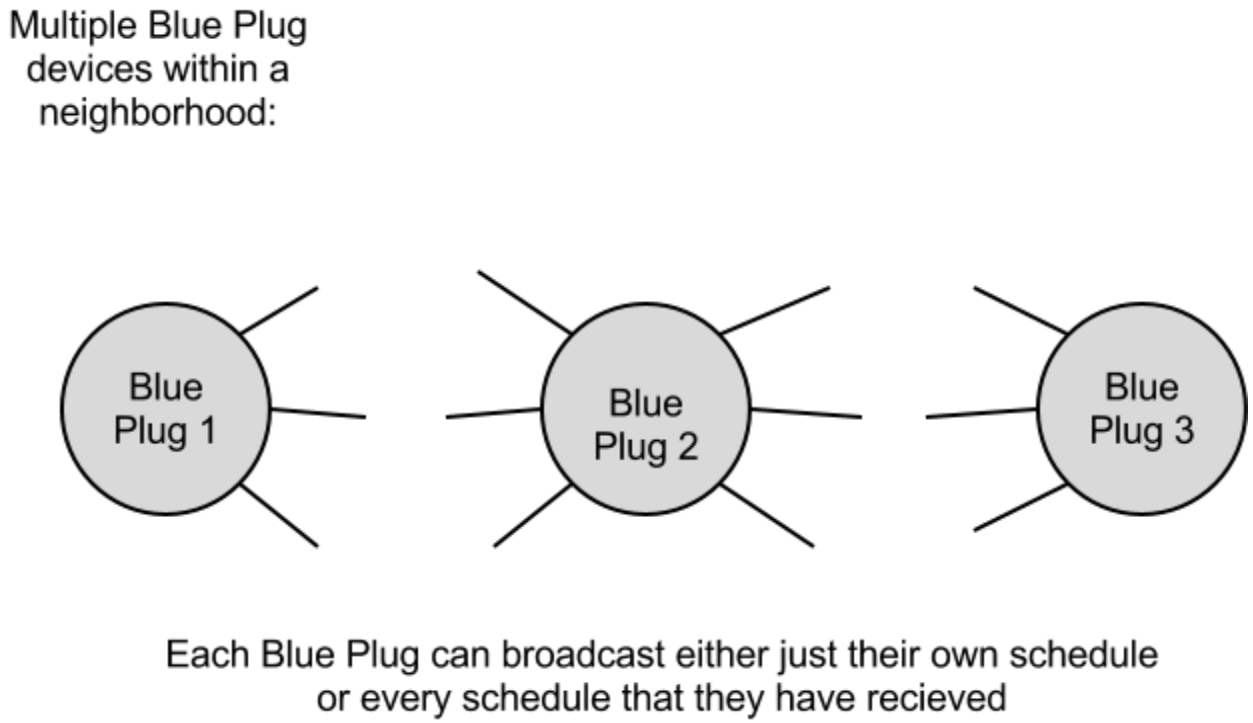

Figure 2 *Multiple Blue Plug Devices*

3.2 Android Application

For our system to work each house needs only a single Blue Plug device in their homes to control multiple appliances. We will look at the entire flow of the Android Application first before going into specifics for each of the three main screens. Figure 3 shows the entire flow chart for the application.

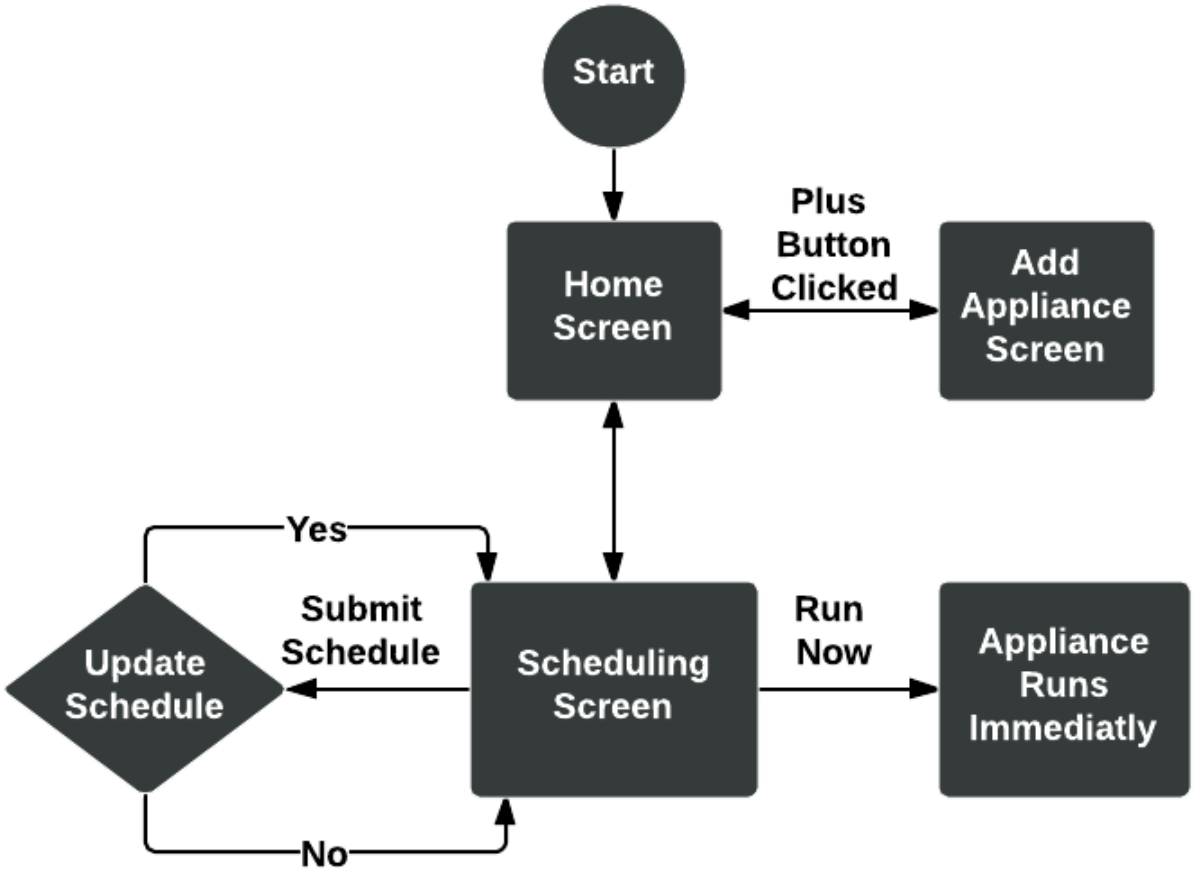

Figure 3 *Android Application Flow Chart*

The user can plug their android device into the Blue Plug and then they can bring up the Blue Plug Android application. The application will boot into the home screen (Figure 4) where they will see a list of appliances currently scheduled to run. The user can remove appliances from the main screen by selecting the trash can on the top action bar. They can also add new appliances to the main screen by selecting the plus button.

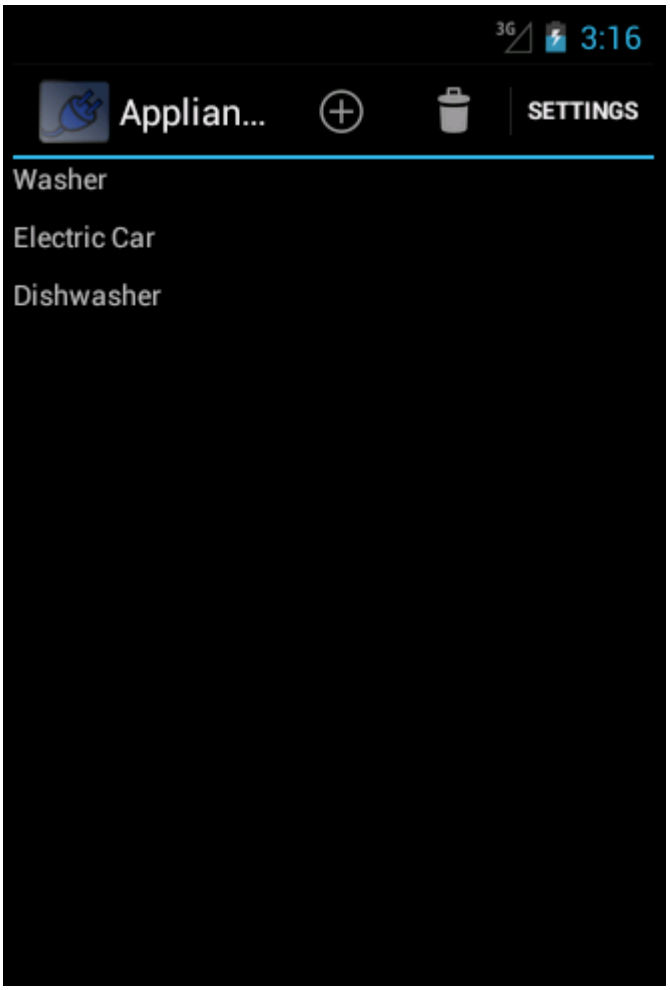

Figure 4 *Home Screen (All Appliances)*: All appliances currently scheduled for use will appear here.

When the plus button has been clicked a popup dialog (Figure 5) will appear and the user can select which appliances they would like to put on the main screen. Only appliances that are registered over Wi-Fi will appear in this secondary list. It is up to the user to ensure that any appliances they wish to control is Wi-Fi capable.

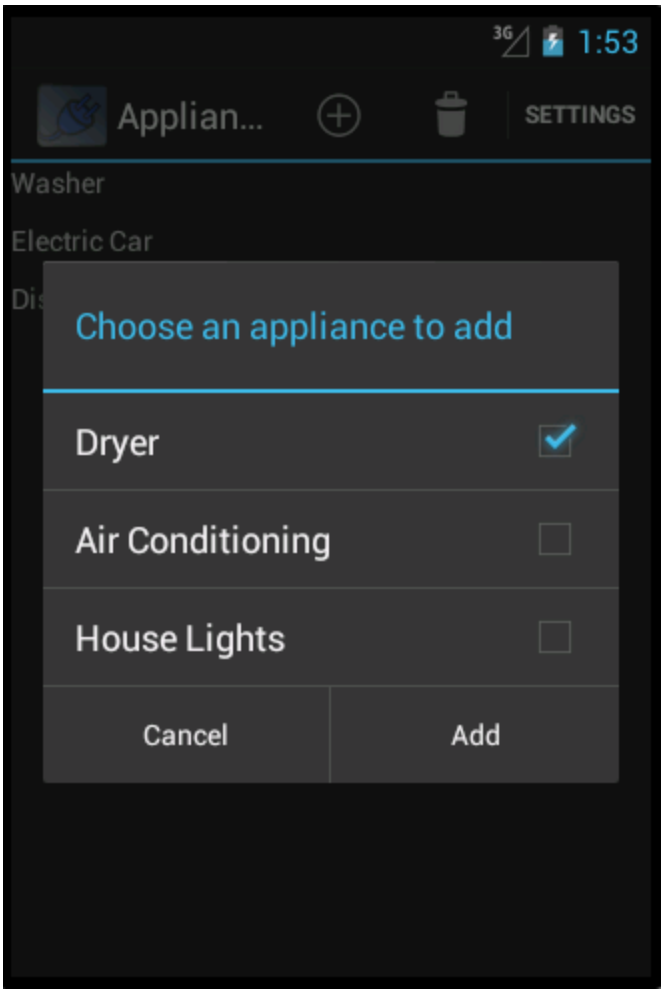

Figure 5 *Add Appliance Screen*: This screen shows appliances that are available to be added to the main screen.

If the user wishes to edit a specific appliance's schedule they simple select the desired appliance and they will be taken to the Specific Appliance screen (Figure 6). The first thing displayed on this screen is the next time the appliance is scheduled to run. Below that is a time picker for the user to input the deadline for that appliance. If the user wants to start the appliance immediately then they can select the "Run Now" button and the Blue Plug system will take car of the rest. If the user wants to submit the actual schedule that they have set then they can simply select the "Submit" button.

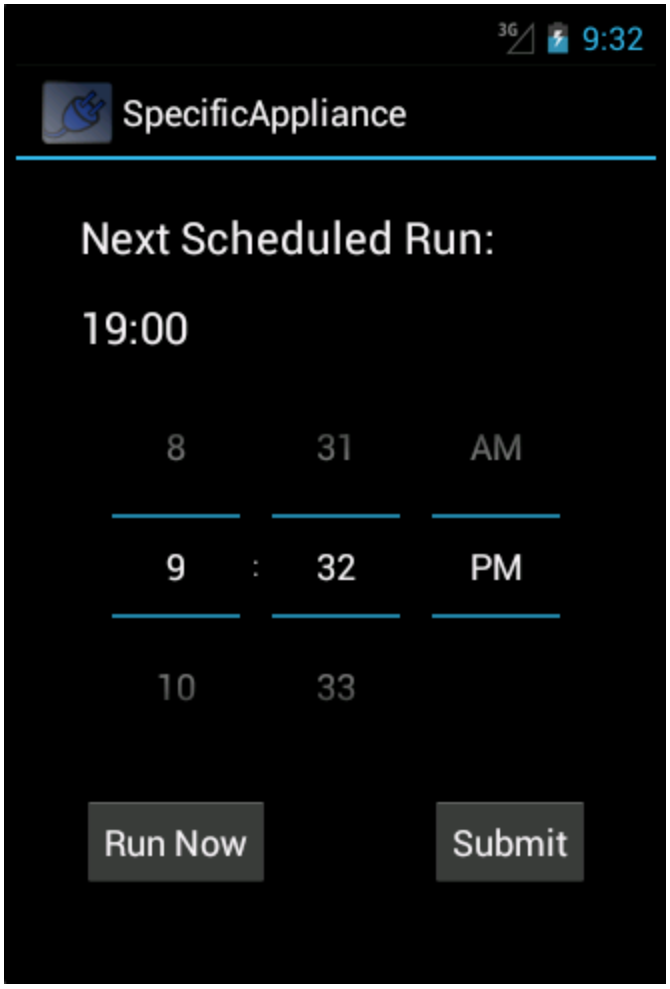

Figure 6 *Specific Appliance Screen*: This screen shows the specific appliance selected by the user and displays a time picker for inputting the deadline.

4. Technologies Used

4.1 Hardware Components

Arduino Uno: Used to control all other components of the Blue Plug System. It also controls the logic for scheduling appliances efficiently. The Arduino Uno is a microcontroller based on the ATMega328 chipset. There are 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog input pins, a 16 MHz crystal oscillator, a USB connection, a 9 volt power jack, an ICSP header, and a reset button.

XBee Series II Radios: Used for the communication between Blue Plug devices via radio waves. The Series II radios allow us to implement a Mesh network, enabling communication between multiple nodes as well as the ability to route information through nodes, much like a Wi-Fi router. The Series II radios implement the Zigbee Mesh protocol on top of the 802.15.4 protocol used by the Series I radios. These devices are designed for high-throughput applications requiring low latency and predictable communication timing.

Arduino SD card and XBee shield: A combined shield that provides an SD card reader for storing appliance schedules as well as pins for the XBee radio. Communication with micro SD cards is done through an SPI interface. The SCK, DI, and DO pins of the micro SD socket are connected to the ATmega168/328's standard SPI pins (digital pins 11-13), while the CS pin is connected to the Arduino's 8^{th} digital pin. The XBee shield unit works with all XBee modules including the Series I and Series II radios.

4.2 Software Components

C/C++: The Arduino language is based on a mixture of C and C++ features. It is an open source platform with many examples readily available as soon as you download the SDK. The Arduino platform runs on Windows, OS X, and Linux. The environment is written in Java and is based on the Java library Processing, the compiler avr-gcc, and other open source software.

Android SDK and Java: The Android Software Development Kit is the set of open source tools provided by Google to do Android Development. It can be installed into a specific Integrated Development Environment (IDE) which allows the user to build Android applications. Applications are developed using a combination of Java and XML files for layout and formatting different views for the application. Android development is also based on the Model View Controller paradigm, which splits up the tasks of managing data from displaying that data.

XCTU: A program that allows us to setup XBee Series II radio chips to communicate with each other by changing communication frequencies, Baud Rate, Channel IDs and Router/Coordinator roles. It also allows us to enable API mode for the Series II radios in order to send packets of data between nodes in the network.

5. Design Rationale

This section will discuss the many choices we made for the technologies we used in order to meet our functional requirements as well as the decisions we made to ensure we were meeting our non-functional requirements. In order to keep the cost of the project down we tried to use as much of the technology from last year as possible.

We use the Arduino Uno because of its inherent accessibility. It offers a cheap solution for us to run our algorithm on and it is readily available. The Arduino platform is also very customizable, which allows us to use both XBee radios as well as other shields without the hassle of wiring our own connections between components.

We chose to use XBee Radio II's because they offered the same capabilities as the Series I's, but they also added the ability to create an Ad Hoc network. This additional feature makes our system more dynamic, allowing us to add and remove nodes on the fly.

We chose to implement the user interface on the Android platform. This means we are targeting a larger audience. Since Android is a well-established platform there is plenty of documentation and information to help us along the way.

Python was an easy choice for the simulation of our scheduling algorithm because it has very useful built in features and additional libraries for simulating large scale systems. It is also easy to use and there is solid documentation and support for it. We chose the Pyro library specifically

because it offered a simple way to simulate radio communication between the different nodes in our simulated distributed network.

Since we chose to use Android as our interface platform we did not have much of a choice in our selection of a language. While we could have used some of the libraries that supported other languages, we thought that for the system we are trying to build sticking with Java would be the easiest solution.

As we said earlier, one of the reasons we chose to use an Arduino board as the main component of the Blue Plug system was its adaptability. We need a way for our system to actually talk to the appliances we want to schedule, which is where the addable shields for the Arduino make perfect sense. As well as allowing us to talk to the individual appliances we can also use the WiFi shield to talk to the Android interface.

Another shield we decided to use was a SD card reader shield that allows us to store scheduling data on each Blue Plug. This offers an easy to use and already existent technology that doesn't require any new technology.

6. Use Cases

Following from the previously mentioned design requirements this section describes the different actions a user can fulfill with our product. Each user can accomplish what they need by using the Android Application we developed.

6.1 Adding an appliance:

Goal: Add an electric appliance to the system for the Blue Plug to schedule.

Actor: Blue Plug User

Precondition: User has connected their Android device to the Blue Plug, electric imp or other Wi-Fi enabling technology is connected to appliance.

Postcondition: An appliance is added to the applications home screen.

Scenario: The user clicks the plus button in the top right corner of the screen. The user then selects the desired devices from a menu with a pre-populated list. The user selects "submit" and appliances are added to the home screen.

Exceptions: Desired appliances not Wi-Fi enabled.

6.2 Setting a deadline to schedule an appliance:

Goal: After selecting an appliance, ensure it executes before the entered time frame.

Actor: Blue Plug User

Precondition: The user has connected to the Blue Plug system and the desired device is already added.

Postcondition: The device executes and completes its task before deadline is reached.

Scenario: The user selects the device from the home screen. The screen for that specific device appears. The user inputs the time at which the appliance must finish. The user clicks “submit”.

Exceptions: The device is not connected to Wi-Fi. Another device is scheduled for the same time (The Blue Plug will find a way to schedule around it).

6.3 Removing an Appliance.

Goal: Remove an appliance that the Blue Plug is keeping track of.

Actor: Blue Plug User

Precondition: The appliance must already be added to the home screen.

Postcondition: The appliance is removed.

Scenario: The user clicks the trash can icon in the top right of the screen. The user selects the appliances to delete. The user then selects “confirm”.

Exceptions: The appliance is currently running. It will be removed after it has finished.

6.4 View Schedule

Goal: View schedule for a specific appliance

Actor: Blue Plug User

Precondition: The user’s Android device is plugged into the Blue Plug System

Postcondition: The user can see the next scheduled run for a given appliance.

Scenario: The user selects the desired appliance from the main screen. The user is then taken to a screen which displays the next scheduled run time for that appliance.

Exceptions: The user has not added the desired appliance to the Blue Plug home screen.

6.5 Turning an Appliance On Immediately

Goal: Turn an appliance on from the Android application.

Actor: Blue Plug User

Precondition: The user has plugged their Android device into the Blue Plug system.

Postcondition: The desired appliance is now running.

Scenario: The user selects the appliance they wish to turn on immediately from the home screen. The user is then taken to the scheduling screen for that appliance and they can select the “run now” button.

Exceptions: The user has not added the desired appliance to the Blue Plug home screen.

6.6 Set up an Ad Hoc Network with Other nearby Blue Plug Devices

Goal: Successfully communicate between other Blue Plug devices in the vicinity

Actor: A single Blue Plug device

Precondition: The system has a schedule and at least one appliance registered

Postcondition: The system has successfully sent and received schedules from other Blue Plugs in the area

Scenario: The user sends a new schedule from the Android device to that specific Blue Plug device. The schedule is then broadcast to other Blue Plug devices in the area. If the Blue Plug device also has schedules from other devices it will send those too.

Exceptions: The device is not within range of another Blue Plug device.

7. Project Risks

In the process of developing this project, we compiled a list of risks related to the project and what consequences they hold. The probability that such a risk will occur is on a scale of 0-1 with 0 being not possible and 1 signifying a guarantee that the risk will happen. Severity is a representation, from a 0-10 of the overall effect of the risk. The higher the severity, the more damaging the risk potentially is to the product. Impact is the result of multiplying the probability of a risk by the severity of the same risk. Finally, mitigation strategy is the proposed solution on how to prevent the specified risk from occurring during the development process.

Risks	Consequences	Probability	Severity	Impact	Mitigation Strategy
Unfamiliar technology	Take more time learning technology than building system Novice Mistakes	**0.9**	**8**	**7.2**	Teach ourselves using online resources, textbooks and asking our advisor and peers for help
Time	- System Features Cut - Product Contains Bugs and Errors	**0.6**	**7**	**4.2**	- Utilized Google Hangouts. - Stuck to Weekly Meetings
Sickness	-Product is not completed.	**0.3**	**5**	**1.5**	Stay healthy by taking vitamins and not getting overly stressed.
Hardware Failure	Data is lost, may have to recreate some code	**0.001**	**8**	**.008**	Back up code and other resources online.

8. Test Plan with Test Cases

During the testing phase of the project development phase we will test all the components of the system. We have broken the test plan into three sections, User Interface, Algorithm and Hardware. These are the three major parts of the system that we need to ensure all operate correctly.

8.1 User Interface

-User can log in with hard coded, device specific password

-User can successfully start an appliance immediately

-User can add an appliance to the current list of registered appliances

-User can schedule an appliance to run at certain time, finish by a certain deadline and set this to recur multiple times per week.

8.2 Algorithm

-Algorithm successfully schedules appliances to run in most efficient manner

8.3 Hardware

-Blue Plug can successfully set up and Ad Hoc network with other Blue Plugs in the area

-Blue Plug can register new appliances over WiFi

-Blue Plug can interact successfully with Android device, getting scheduling information from users input and implementing schedule efficiently.

9. Developmental Timeline

We used a Gantt chart that shows tasks to be completed, mainly for implementation, documentation, testing and presentations.

	July	Aug.	Sep	Oct.	Nov.	Dec.
Understanding Previous and Current Work						
Document Design						
Technology Research **NOTE: Red is BOTH**						

	Jan.	Feb.	Mar.	Apr.	May
Implementation					
Android Class					
Arduino Architecture					
Radio Comm.					
File Storage					
Android UX					
Android <-> Arduino					
	<-- Sahil		<-- Loquen		<-- Both

10. User Manual

Much like the test plan is broken into three parts; the user manual is split into the Android interface and the setup of a single Blue Plug system.

10.1 Blue Plug System Setup:

The Blue Plug merely requires you to plug the device into an outlet. Once you see an orange light, it means we need to set the system up with your home Wi-Fi network.

10.2 Blue Plug Interface:

Download the application from the Android Play Store. Then all you have to do is plug a USB cable from the Android device to your Blue Plug Device. This will let you update the Blue Plug Schedule from the Blue Plug Android application.

In order to add an appliance for scheduling, click the plus button, select the appliance under the list, and click add.

To delete an appliance, go to the home screen, and click the trash can button and then select the appliance you wish to delete. Finally hit confirm to finish the process.

To schedule an appliance, you must select it from the home screen. You will be taken to the scheduling screen for that specific appliance. From there you can input the deadline using the

time picker provided. You can then hit the submit button to send the new schedule to the Blue Plug device.

To turn an appliance on immediately, select it from the home screen. You will be taken to the scheduling screen for that specific appliance. Simply click the run now button to start your appliance.

11. Conclusion

As we look back and understand the body of work we have completed this last year, our aim was to create an open system that allows anyone with the passion and drive to reduce energy usage, save money, and protect the environment. The system we have created allows the independent parts to be smart enough that they do not need any extra work from the end user to implement the system. We hope that users will find the Blue Plug service beneficial and empower them to use our system to reduce the use of polluting power plants as well as reduce energy costs around the world.

12. Societal Issues

12.1 Ethical

With the rise in greenhouse gases and the decrease in our fossil fuel supply a change in our energy consumption is becoming a necessity. This issue of efficient energy consumption is about more than just saving money for energy consumers and producers. The development of the Blue Plug will lead to more efficient energy use which will help eliminate peak hours that occur when everyone comes home from work and turns on multiple appliances. This in turn will stop energy companies from running their quick start up generators which tend to be the coal and oil powered sites. By eliminating the use of these fossil fuel generators air quality will increase and pollution will decrease. All these benefits end up helping the entire world as a whole, not just those directly using the Blue Plug System. The Blue Plug also will not interfere with the operation of the existing power grid; instead it will work alongside it to provide consumers better ways to manage their energy consumption.

12.2 Social

The most common societal issue stems from the user's ability to schedule. Often, people wonder if they have to forsake their TV or AC time to let their neighbors run it. This is never the case, as these appliances, which are random, are not the intended products for the Blue Plug. The appliances are always ones that follow a stable usage, where the product cycle must run once, not subject to end user's decisions. The other societal issue is that of competition. Neighbors tend to not want to share electricity usage with neighbors. However, this is not the problem; we merely control the scheduling and make sure we follow the deadlines. The neighbors are not sharing

usage, they are dispersing it.

12.3 Aesthetic

Perhaps the most challenging aspect of software development lies in appealing to the end-user. What's very difficult to achieve is creating software that works as expected, in an intuitive manner where user-manuals are not required. When an end-user sees a plus button, it's expected that, by pressing the plus button, some action will occur that relates to augmenting whatever they are doing; the same applies with things like a minus button, or trash can. Ensuring that code is written in such a way that we do not have confusion for the end user is our highest priority. This is the only way we can expect adoptability. The end-user's time and patience will already be razor-thin, the software aesthetic should be designed in such a way where it takes a mere 15 seconds to execute major functionality.

13. Appendix

This section is split into two parts, Arduino and Android code. The Arduino code is taken from our demo as well as the final system.

13.1 Arduino Code

13.1.1 Receive Code:

```
#include <XBee.h>
#include <NewSoftSerial.h>
const int ledPin1 = 10;
const int ledPin2 = 11;
const int ledPin3 = 12;
const int ledPin4 = 13;
int incomingByte;
int r;
void setup()
{
  Serial.begin(9600);
  randomSeed(analogRead(0));
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
```

```
  pinMode(ledPin4, OUTPUT);
}
void loop()
{
  r = random(4);
Serial.print('C');
delay(1000);
Serial.print('D');
delay(1000);
 switch(r)
  {
    case 0:
      //Serial.println('A');
     digitalWrite( 10, HIGH );
      delay(2000);
      digitalWrite( 10, LOW );
      break;
    case 1:
      //Serial.println('B');
      digitalWrite( 11, HIGH );
      delay(2000);
      digitalWrite( 11, LOW );
      break;
```

```
  case 2:
    //Serial.println('C');
    digitalWrite( 12, HIGH );
    delay(2000);
    digitalWrite( 12, LOW );
    break;
  case 3:
    //Serial.println('D');
    digitalWrite( 13, HIGH );
    delay(2000);
    digitalWrite( 13, LOW );
    break;
  default:
    Serial.println("No valid random number");
    delay(1000);
    break;
}
 if(Serial.available() > 0) {
  // read the oldest byte in the serial buffer:
  incomingByte = Serial.read();
  Serial.print("Here");
  // if it's a A turn on the LED1:
  if (incomingByte == 'A') {
```

```
  Serial.print('A');
  digitalWrite(ledPin1, HIGH);
  delay(1000);
  digitalWrite(ledPin1, LOW);
}
// if it's a B turn on the LED2:
if (incomingByte == 'B') {
  Serial.print('B');
  digitalWrite(ledPin2, HIGH);
  delay(1000);
  digitalWrite(ledPin2, LOW);
}
//if it's a C turn on the LED3:
if (incomingByte == 'C') {
  Serial.print('C');
  digitalWrite(ledPin3, HIGH);
  delay(1000);
  digitalWrite(ledPin3, LOW);
}
//if it's a D turn on the LED4:
if (incomingByte == 'D') {
  Serial.print('D');
  digitalWrite(ledPin4, HIGH);
```

```
    delay(1000);
    digitalWrite(ledPin4, LOW);
  }
}
case 5:
    Serial.print('E');
    break;
  case 6:
    Serial.print('F');
    break;
  case 7:
    Serial.print('G');
    break;
  case 8:
    Serial.print('H');
    break;
  case 9:
    Serial.print('I');
    break;
  case 10:
    Serial.print('J');
    break;
  case 11:
```

```
      Serial.print('K');
      break;
    case 12:
      Serial.print('L');
      break;
    default:
      Serial.print('M');
      break;
    }
}
```

13.1.2 Final Arduino Demonstration Code:

```
// Set four LEDs to represent HOUSE appliances
int BUTTON = 4;
int sixthLED = 6;
int eighthLED = 8;
int tenthLED = 10;
int twelfthLED = 12;
//int thirteenthLED = 13;
int hasTurnedOn[4] = {0,0,0,0};
int buttonState = 0;
// if this is a zero, we do random, if this is one, we do
// BLUE PLUG SCHEDULING!
```

```
int buttonDebounce = 0;
// the setup routine runs once when you press reset:
void setup() {
  // button to debounce between different LED modes.
  pinMode( BUTTON, INPUT );
  // initialize the digital pin as an output.
  pinMode( 5, OUTPUT );
  pinMode( sixthLED, OUTPUT );
  pinMode( eighthLED, OUTPUT );
  pinMode( tenthLED, OUTPUT );
  pinMode( twelfthLED, OUTPUT );
 // pinMode( thirteenthLED, OUTPUT );
}
// the loop routine runs over and over again forever:
void loop() {
  int i = 0;
  int lightToTurnOn;
  digitalWrite( sixthLED, HIGH );
  digitalWrite( eighthLED, HIGH );
  digitalWrite( tenthLED, HIGH );
  digitalWrite( twelfthLED, HIGH );
  buttonState = digitalRead( BUTTON );
  /* we need to differentiate between a random, or actual comm;*/
```

```
if( buttonState == HIGH){
    /* RANDOM SCHEDULING*/
    if( buttonDebounce == 0 ){
        buttonDebounce++;
        /* replicating scheduling */
        toggleLight( 5 );
        while( i <  4 ){
            setLightsToZero();
            lightToTurnOn = getRandomAppliance();
            toggleLight( lightToTurnOn );
            i++;
        }
        /* the runnin of simultaneous apps! */
        setAppsToZero();
        i = 0;
        int L1, L2, L3, L4;
        while( i < 2 ){
            switch( i ){
                // turning on three lights.
                case 0:
                    setLightsToZero();
                    toggleLight( 5 );
                    L1 = getRandomAppliance();
```

```
    L2 = getRandomAppliance();
    L3 = getRandomAppliance();
    digitalWrite( L1, HIGH );
    digitalWrite( L2, HIGH );
    digitalWrite( L3, HIGH );
    delay( 2000 );
    digitalWrite( L1, LOW );
    digitalWrite( L2, LOW );
    digitalWrite( L3, LOW );
    L4 = getRandomAppliance();
    digitalWrite( L4, HIGH );
    delay( 2000 );
    digitalWrite( L4, LOW );
    setAppsToZero();
    setLightsToZero();
    i++;
// turning on two lights
case 1:
    toggleLight( 5 );
    L1 = getRandomAppliance();
    L2 = getRandomAppliance();
    digitalWrite( L1, HIGH );
    digitalWrite( L2, HIGH );
```

```
            delay( 1500 );
            digitalWrite( L1, LOW );
            digitalWrite( L2, LOW );
            delay( 1500 );
            runRemainingAppliances();
            i++;
        default:
        break;
      }
   }
}
/* BLUE PLUG SCHEDULING*/
else{
   setLightsToZero();
   delay( 5000 );
   digitalWrite( 5, HIGH );
   delay( 500 );
   digitalWrite( 5, LOW );
   delay( 500 );
   delay( 1000 );
   for( i = 6; i <= 12; i= i + 2){
      toggleLight( i );
   }
```

```
    }
  }
}
// gets an appliance to turn on
int getRandomAppliance(){
  int foundSuitableAppliance = false;
  int i = random( 0, 4 );
  while( ! foundSuitableAppliance ){
    if( hasTurnedOn[i] == 0 ){
      hasTurnedOn[i] = 1;
      if( i == 0 ){
        return 6;
      }
      if( i == 1 ){
        return 8;
      }
      if( i == 2 ){
        return 10;
      }
      if( i == 3 ){
        return 12;
      }
    }
```

```
      if( i == 3 ){
         i = 0;
      }
      else{
         i++;
      }
  }
}
/* helper function to flick a light on and off */
void toggleLight( int lightToTurnOn ){
   digitalWrite( lightToTurnOn, HIGH );
   delay( 1000 );
   digitalWrite( lightToTurnOn, LOW );
   delay( 1000 );
   return;
}
/* setting our global appliance list to zero*/
void setAppsToZero(){
   /* setting appliance counter to zero*/
   int z;
   for( z=0; z < 4; z++ ){
      hasTurnedOn[z] = 0;
   }
```

```
    return;
}
/* helper function to zero out lights */
void setLightsToZero(){
    digitalWrite( sixthLED, LOW );
    digitalWrite( eighthLED, LOW );
    digitalWrite( tenthLED, LOW );
    digitalWrite( twelfthLED, LOW );
    return;
}
void runRemainingAppliances(){
    int z;
    int x;
    for( z=0; z<4; z++ ){
        if( hasTurnedOn[z] == 0 ){
            x = mapper( z );
            toggleLight( x );
            hasTurnedOn[z] = 1;
        }
    }
}
/* turn array indices to arduino switches */
int mapper( int i ){
```

```
  if( i == 0 ){
      return 6;
   }
  if( i == 1 ){
     return 8;
  }
  if( i == 2 ){
     return 10;
  }
  if( i == 3 ){
     return 12;
  }
}
```

13.2 Android code

The Android code is broken up into layout code which are just XML files as well as the actual Java code.

13.2.1 Menu Code (Appliance Activity, XML):

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_settings"
        android:title="@string/menu_settings"
```

```
        android:orderInCategory="3"
        android:showAsAction="always" />
    <item android:id="@+id/add"
        android:title="@string/add"
        android:orderInCategory="1"
        android:showAsAction="always"
        android:icon="@android:drawable/ic_menu_add"/>
    <item android:id="@+id/remove"
        android:title="@string/remove"
        android:orderInCategory="2"
        android:showAsAction="always"
        android:icon="@android:drawable/ic_menu_delete"/>
</menu>
```

13.2.2 Menu Code (Specific Appliance, XML):

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_settings"
        android:title="@string/menu_settings"
        android:orderInCategory="100"
        android:showAsAction="never" />
</menu>
```

13.2.3 Menu Code (Splash Screen, XML):

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_settings"
        android:title="@string/menu_settings"
        android:orderInCategory="100"
        android:showAsAction="never" />
</menu>
```

13.2.4Layout Code (Appliance Activity, XML):

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:minHeight="?android:attr/listPreferredItemHeight"
        android:textSize="30sp"
        android:divider="@android:color/transparent"
        android:dividerHeight="10.0sp" >
    </ListView>
</RelativeLayout>
```

13.2.5 Layout Code (Specific Appliance, XML):

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TimePicker
        android:id="@+id/timePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" />
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignRight="@+id/timePicker1"
        android:layout_marginBottom="42dp"
        android:text="@string/submit" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_alignBaseline="@+id/button1"
        android:layout_alignBottom="@+id/button1"
        android:layout_alignLeft="@+id/timePicker1"
        android:text="@string/run_now" />
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/timePicker1"
        android:layout_alignParentTop="true"
        android:layout_marginTop="23dp"
        android:text="@string/run_next"
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="15dp"
        android:text="@string/next_time"
        android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>
```

13.2.6 Layout Code (Splash Screen, XML):

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"
        android:contentDescription="@string/splash_screen"
        android:src="@drawable/splashtitle" />
</RelativeLayout>
```

13.2.7 Android Manifest (XML):

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.seniordesign.theblueplug"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="15" />
```

```
<application
    android:icon="@drawable/iconhi"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".SplashActivity"
        android:label="@string/title_activity_splash"
        android:screenOrientation="portrait" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name=".LoginActivity"
        android:label="@string/title_activity_login" >
    </activity>
    <activity
        android:name=".AppliancesActivity"
        android:label="@string/title_activity_appliances" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
```

```
            </intent-filter>
        </activity>
        <activity
            android:name=".SpecificAppliance"
            android:label="@string/title_activity_specific_appliance" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

13.2.8 Appliance Dialog Builder (Java):

```
package com.seniordesign.theblueplug;
import java.util.ArrayList;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
public class ApplianceDialogFragment extends DialogFragment {
        //For sending back which dialog button was selected
```

```
        public interface ApplianceDialogListener {
        public void onDialogPositiveClick(DialogFragment dialog);
        public void onDialogNegativeClick(DialogFragment dialog);
    }


        @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
// Use the Builder class for convenient dialog construction
//        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
//        builder.setTitle(R.string.appliances_prompt)
//               .setItems(R.array.appliances_array, new DialogInterface.OnClickListener() {
//                   public void onClick(DialogInterface dialog, int which) {
//                       // The 'which' argument contains the index position
//                       // of the selected item
//
//                   }
//           });
                    final ArrayList<Object> mSelectedItems = new ArrayList<Object>();  // Where
we track the selected items
               AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
               // Set the dialog title
               builder.setTitle(R.string.appliances_prompt)
               // Specify the list array, the items to be selected by default (null for none),
```

```
// and the listener through which to receive callbacks when items are selected
      .setMultiChoiceItems(R.array.appliances_array, null,
                 new DialogInterface.OnMultiChoiceClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which,
                    boolean isChecked) {
               if (isChecked) {
                  // If the user checked the item, add it to the selected items
                  mSelectedItems.add(which);
               } else if (mSelectedItems.contains(which)) {
                  // Else, if the item is already in the array, remove it
                  mSelectedItems.remove(Integer.valueOf(which));
               }
            }
         })
// Set the action buttons
      .setPositiveButton(R.string.add, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
               // User clicked Add, so save the mSelectedItems results somewhere
               // or return them to the component that opened the dialog
            }
         })
```

```
                .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int id) {


                    }
                });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

13.2.9 Appliance Activity (Java):

```
package com.seniordesign.theblueplug;

import android.os.Bundle;

import android.app.ActionBar;

import android.support.v4.app.FragmentActivity;

import android.view.LayoutInflater;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.view.ViewGroup;

import android.view.Window;

import android.widget.BaseAdapter;
```

```
import android.widget.ListView;
import android.widget.TextView;
public class AppliancesActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Creating an action bar for the activity
        getWindow().requestFeature(Window.FEATURE_ACTION_BAR);
        setContentView(R.layout.activity_appliances);
        ActionBar actionBar = getActionBar();
        actionBar.setTitle(R.string.appliances);
        actionBar.show();
            //Sample appliances for demonstration
        final String[] words = {"Washer", "Electric Car", "Dishwasher"};
        //Setting up the list view and constructors
        ListView listView = (ListView) findViewById(R.id.listView1);
        listView.setAdapter(new BaseAdapter() {
            public int getCount(){
                    return words.length;
            }
            public Object getItem(int position) {
                    return words[position];
            }
```

```
        public long getItemId(int position) {
                return position;
        }
        public View getView(int position, View convertView, ViewGroup parent) {
                LayoutInflater inflater = (LayoutInflater)
getSystemService(LAYOUT_INFLATER_SERVICE);
                View view = inflater.inflate(R.layout.list_row,  null);
                TextView textView = (TextView) view.findViewById(R.id.TextView1);
                textView.setText(words[position]);
                return view;
        }
    });
  }
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_appliances, menu);
    return true;
  }
  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    // pop up menu for Add button, lists appliances to add
    switch(item.getItemId())
    {
```

```
        case R.id.add:
            ApplianceDialogFragment appliance = new ApplianceDialogFragment();
            appliance.show(getSupportFragmentManager(), "appliances");
            break;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

13.2.10 List Content Fragment (Java):

```
package com.seniordesign.theblueplug;
import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
public class ListContentFragment extends Fragment {
    private String mText;
    @Override
    public void onAttach(Activity activity) {
      // This is the first callback received; here we can set the text for
```

```
    // the fragment as defined by the tag specified during the fragment transaction
    super.onAttach(activity);
    mText = getTag();
  }
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    // This is called to define the layout for the fragment;
    // we just create a TextView and set its text to be the fragment tag
    TextView text = new TextView(getActivity());
    text.setText(mText);
    return text;
  }
}
```

13.2.11 Specific Appliance (Java):

```
package com.seniordesign.theblueplug;
import android.os.Bundle;
import android.app.ActionBar;
import android.app.Activity;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.View;
```

```
import android.view.ViewGroup;
import android.view.Window;
import android.widget.BaseAdapter;
import android.widget.ListView;
import android.widget.TextView;
public class SpecificAppliance extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Creating an action bar for the activity
       getWindow().requestFeature(Window.FEATURE_ACTION_BAR);
        setContentView(R.layout.activity_specific_appliance);
        ActionBar actionBar = getActionBar();
        actionBar.show();
}
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_specific_appliance, menu);
        return true;
    }
}
```

13.2.12 Splash Activity (Java):

```
package com.seniordesign.theblueplug;
import java.util.Timer;
import java.util.TimerTask;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
public class SplashActivity extends Activity {
        private long splashDelay = 2000; //5 seconds
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        TimerTask task = new TimerTask()
        {
            @Override
            public void run()
            {
                    finish();
                    Intent appliances = new Intent().setClass(SplashActivity.this,
AppliancesActivity.class);
                    startActivity(appliances);
            }
```

```
        };
        Timer timer = new Timer();
        timer.schedule(task, splashDelay);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_splash, menu);
        return true;
    }
}
```

13.2.13 Strings (XML):

```
<resources>
    <string name="app_name">The Blue Plug</string>
    <string name="loginblurb">Login with the hardcoded passcode found on your Blue Plug device.</string>
    <string name="login">Login</string>
    <string name="submit">Submit</string>
    <string name="done">Done</string>
    <string name="cancel">Cancel</string>
    <string name="remove">Remove</string>
    <string name="add">Add</string>
    <string name="appliances">Appliances</string>
```

```
    <string name="menu_settings">Settings</string>
    <string name="title_activity_login">LoginActivity</string>
    <string name="splash_screen">Splash Screen</string>
    <string name="run_next">Next Scheduled Run:</string>
    <string name="next_time">19:00</string>
    <string name="run_now">Run Now</string>
    <color name="blue">#2B4F81</color>
    <color name="black">#000000</color>
    <color name="white">#FFFFFF</color>
    <string name="title_activity_splash">SplashActivity</string>
    <string name="title_activity_appliances">AppliancesActivity</string>
    <string name="title_activity_specific_appliance">SpecificAppliance</string>
    <string name="appliances_prompt">Choose an appliance to add</string>
    <string-array name="appliances_array">
        <item>Dryer</item>
        <item>Air Conditioning</item>
        <item>House Lights</item>
    </string-array>
</resources>
```