University Of Auckland

Solar Monitoring Panel

BTech 451 Mid-Year Report

Author:
David Armstrong
5636534
darm230

Supervisor:
Dr. Ulrich Speidel

August 2014

Abstract

This report contains information about all my project-related work on my BTech 451 project up to Sunday 10th August 2014. This BTech451 project is with Vector Limited. The project involves producing a solution to answer a research question. This research question is based on an optimisation challenge that the global solar industry is facing. The solution will consist of a device that can turn a load on in the house to consume power when it is in excess, with the goal of providing additional value to the customer. The hardware component of the solution will consist of a Raspberry Pi connected to a mains-switching relay. The software component of the solution will involve a program running on the Raspberry Pi that is capable of making intelligent decisions based on the state and recent history of a solar panel system.

The first chapter of this report introduces and outlines my project. It first gives an introduction to Vector and the solar panel system that I'm working with. This chapter then addresses the research question and the solution I am aiming to provide that will enable me to answer the research question. There is also a brief introduction to some key people who are involved with this project.

The second chapter provides a deeper understanding of the solar panel system that I'm working with. It looks at the hardware components of the system, how they interact with the house, as well as the software interface for the system.

The following chapter, chapter three, looks at the research I did into technologies. I needed to do this research as Vector wasn't specific about what device or programming language(s) I should be using to create my solution. This chapter lists options and explains the choices I made.

Chapter four then looks at the design of my solution that I will be creating. It looks at both the hardware component of my solution and the software component of my solution.

The fifth chapter then looks at how much of the solution I have currently created and implemented. It looks at both the hardware and software components of my solution.

The following chapter, chapter six, looks at the future work of my project. This future work is based on the idea of the project I have at this point, however it may change due input from Vector.

The final chapter is the conclusion of this report and looks at the current work of the project and what will be done in the future.

Following the conclusion I have bibliography which lists some resources and references that I have used during this project.

Contents

Chapter 1

Project Introduction

The BTech (IT) degree is a four year degree offered by the University of Auckland. The courses that are taken as part of this degree are mostly Computer Science and Information Systems papers but include various others. As part of this degree, there is a year-long project in the final year of the degree. This project is known as the BTech 451 project and is worth 45 points, or 3 papers. This project usually involves an industry sponsor that specifies a research question for the student to work on and answer.
My BTech 451 project is formally known as the "Solar Monitoring Panel" and is sponsored by the company Vector Limited. This chapter gives an introduction to Vector, the background behind my project and an overview of my project.

1.1 The Company

Vector Limited [1] is an Auckland based multi-network infrastructure company. Vector was formed in 1999 from Mercury Energy when new government regulations were introduced, splitting up electricity distribution and electricity generation businesses. Electricity distribution is Vector's primary role. They own and operate the electricity distribution network in the greater-Auckland area and they are also the largest power distribution company in the country, however they don't produce the power themselves. Vector also operates a natural gas network, they supply gas to over 150,000 customers across the North Island of New Zealand, using their 9,300 kilometre-long gas network. Vector owns their own fibre-optic telecommunication networks in Auckland and Wellington. They also have a nation-wide fibre network that has access points in other major cities, such as Hamilton, Tauranga and Christchurch.
Recently Vector, like other distribution networks, has begun to investigate an optimisation challenge that is based on over-supply with solar panel systems. The particular solar panel

system that Vector is allowing me to work with for this project consists of a collection of components that are installed in a house or business. The system allows power to be generated from the installed solar panels on a building's roof. This power can be directly used by the building, or stored in an onsite battery for future use. The battery can be used at the same time as power is being produced by solar panels, or later during a time when no solar power is being produced. The home/business is still connected to the power grid like normal buildings, and whatever power the system requires that the solar panel system is not providing on its own will be brought in from the grid like normal. Excess power produced by a system can be exported back into the grid. This occurs when the battery is fully charged and the system is producing more power than the home/business requires. The owner of the home/business is reimbursed for the power exported, however this rate is sometimes less than what it would cost to bring the same amount of power in from the grid. There is also an HTTPS-based API for interacting with each solar panel system.

1.2 The Research Question

A current challenge that the global solar industry faces is optimising the use of solar panel systems to provide customers with the maximum value that the system can offer. The research question for this project is "Can the strategic use of excess power provide additional value to the customer".

As mentioned before, excess power generated by a system is exported back into the grid, provided the battery is fully charged and the solar panels are producing more power than the house is consuming. While in principle this is the desired behaviour of the system, there are situations where there is optimisation potential. An example for such optimisation potential for the customer is when their system exports excess power during the day, but later buys in power from the grid to power a power-hungry appliance. It would be in the customer's interest and more efficient to use the appliance when the excess power is available. Another example of optimisation potential is when excess power is exported to the grid while the export reimbursement rate is low. Depending on how much power has been exported into the grid, the rate that a customer is reimbursed for exporting power can vary. It would be a benefit to the customer to consume excess power with a power-hungry device rather than exporting to the grid when the export rate is low.

Each solar panel system needs to make effective use of the excess power that it has generated, however many systems have no control over loads in the house. It is also too much work for the customer to monitor the status of a system to find out when it would be a good time to use the excess power.

1.3 The Solution

To explore and answer this research question, Vector would like me to look for potential added benefits that could arise from having a home/business strategically consume excess power in situations instead of exporting it to the grid. One way of effectively using excess power would be to have an appliance like a hot-water pre-heater run automatically when power is in excess. Pre-heating the water when power is in excess will mean that the normal hot-water heater won't have to work as hard later as the water that enters the hot-water heater will already be hotter than normal. If this device can run at times when the export reimbursement rate is low then this will provide additional value to the customer. Another way of effectively using excess power would be to have a power-hungry appliance run when power is in excess, rather than exporting current power and buying in power later in the day to power the appliance. An example of this appliance could be a dryer. If it is possible to run the appliance earlier when power is in excess then it will consume the excess power and there may be no need to import any power for it. In this case the customer could be alerted that it is an appropriate time to use the appliance.
As part of this project, my job is to explore and answer the research question. To answer this research question I will produce a solution that will enable an optional power-hungry appliance in the house to be turned on to consume the excess power in a useful and efficient way. Vector wants the solution to be in the form of a device that runs locally in the home/business. The device will contain a program that will look at the current power state of a solar panel system, recent power history of the system, and make a decision to turn an appliance/load on or off accordingly.
Vector would also like information about decision outcomes and power states sent back to them as otherwise they would have a limited view of what's going on.

1.4 People Involved

There are several people involved with this project, from either Vector or The University of Auckland. Here are some of the key people that have been involved with this project.

- Academic supervisor: Ulrich Speidel
  My academic supervisor for this project is Ulrich Speidel. He has been assisting me and supporting me throughout this project. I have kept in regular contact with Ulrich during the project.
- BTech Coordinator: Sathiamoorthy Manoharan (Mano)
  Mano is the BTech (IT) Coordinator and is the one who manages the BTech 451 project course. He was the one who assigned me this project.

- Previous Industry supervisor: Anthony Thornton
  Anthony Thornton was my initial Industry supervisor, however he has recently left Vector.

- Current Industry supervisor: Steve Muscroft-Taylor
  After Anthony Thornton left Vector, Steve Muscroft-Taylor took his place and became my new Industry supervisor. He was able to attend my mid-year seminar.

Chapter 2

Solar Panel System

The solar panel system that I'm working with is a complicated system that coexists with existing standard power components of a building. The system contains both hardware and software components. This chapter looks at the system and its components that are installed in and outside the building.

2.1 Hardware

Several hardware components make up the system. They are mostly installed in a fridge-sized cabinet that is located on the outside wall of the building. These components interact with the existing components of the house. All the components are connected to an onsite computer than connects to the internet via the pre-existing home/business internet connection. Figure 2.1 refers to a logical diagram that displays the major components of the system. As this is a logical diagram, the placement of components in this diagram are not accurate or to scale, but it gives an idea of how they work together.

2.1.1 Solar Panel Array

Solar panels are an essential part of the system. An array of solar panels are installed onto the roof of a house. They are placed either just on one side of the roof, or both sides depending on how many the resident wants. If they are installed on just one side of the roof then they are placed in the optimal position to generate the most power from the sun. Power produced by the solar panels is in Direct Current (DC). Power from the panels are fed into the Maximum Power Point Tracker (MPPT).

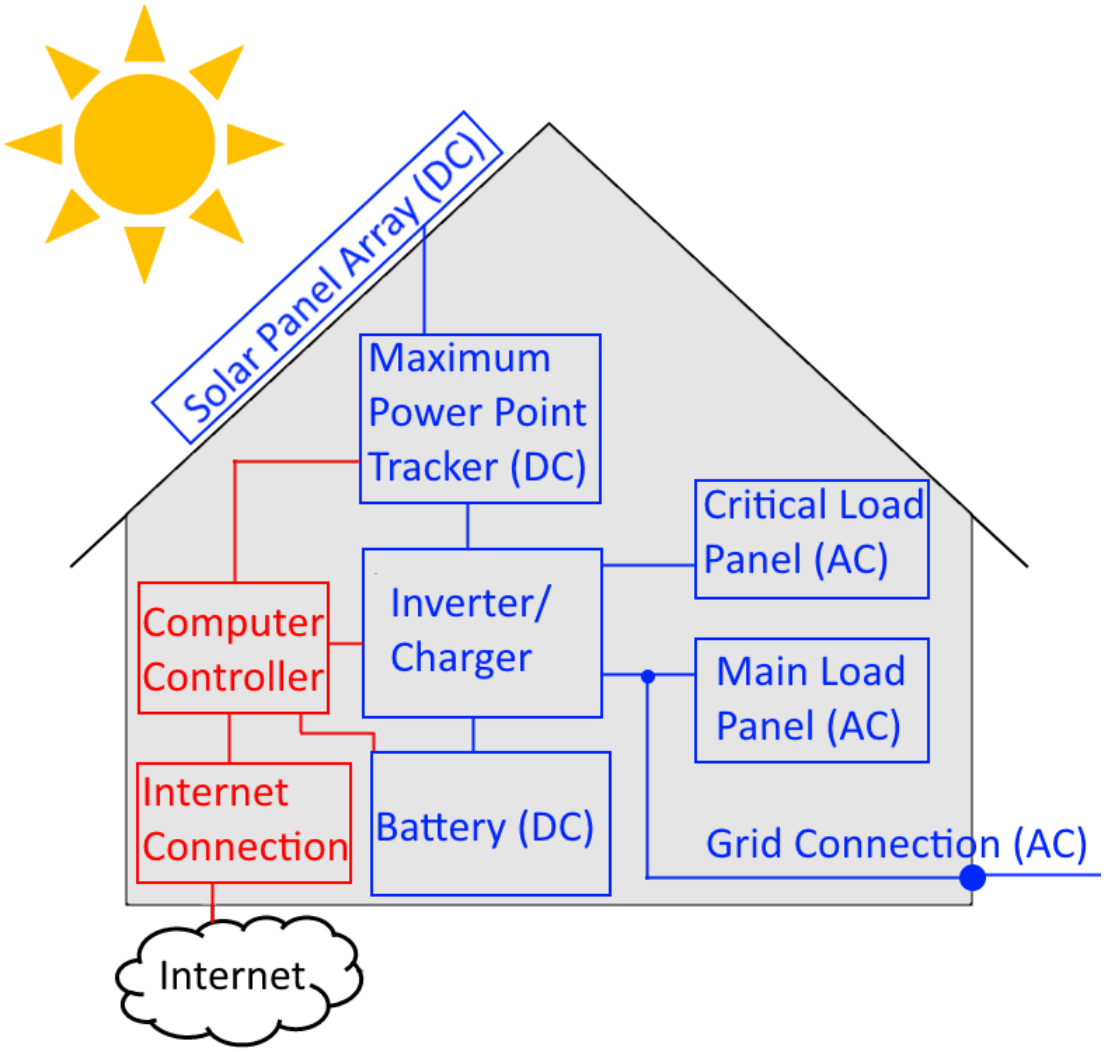

FIGURE 2.1: Solar Panel System - Logical Diagram

2.1.2 Maximum Power Point Tracker (MPPT)

The Maximum Power Point Tracker (MPPT) is a device that aims to optimise the power being produced by the solar panels at any given point in time. The MPPT does this by varying the resistance of a circuit to maximise the Current-Voltage product, and therefore get the maximum amount of power available from the solar panels. The optimally generated power is fed from the MPPT into the Inverter/Charger

2.1.3 Inverter/Charger

The Inverter/Charger is at the heart of the system. Its main job is to convert between DC and AC power when power flows exist between components that use a mix of AC and

DC power. If a power flow between components is in the same form the whole time – e.g. flowing from one AC component to another AC component, it will just pass through the Inverter/Charger without being converted. DC power is used by the MPPT (originating from the solar panels) and the battery. AC power is used by the Critical Load Panel (critical appliances in the house), Main Load Panel (other appliances in the house) and the grid connection.

2.1.4 Battery

The battery stores power for later use. It uses DC power. The power used to charge the battery can come from solar panels (via MPPT), or the grid. Having a chargeable battery means that power can be stored while it is sunny, and used later when no power is produced by solar panels. An example of this would be at night time or while it is very cloudy. Batteries can also be used in the event of a power cut, so essential devices can still run.

2.1.5 Critical Load Panel (CLP)

The Critical Load Panel (CLP) has the "critical" appliances and devices in the house attached to it. It uses AC power. The CLP is connected directly to the Inverter/Charger. In the event of a power cut, the CLP can still be provided with power from the battery (via Inverter/Charger), or if the power cut is during day time it can also be provided with power from the solar panels (via MPPT and Inverter/Charger). Power from the grid can flow to the CLP like with standard houses (not using the solar panel system) via the Inverter/Charger.

2.1.6 Main Load Panel (MLP)

The Main Load Panel (MLP) has the remaining (lower priority) appliances in the house attached to it. It uses AC power. The MLP either receives its power from Inverter Charger (sourced from the solar panels or the battery) or from the grid (like with standard homes).

2.1.7 Computer Controller

The computer controller doesn't have any large amounts of power flowing through it, but instead has data connections to components in the system. It is also connected to the internet via the home internet connection. The computer looks at several bits of

information from around the system, including the battery charge levels, power being produced by the solar panels, and the flow of power between each component. Based on the information of the system and the current power plan settings, it makes decisions about whether the battery should be charged, left alone, or if power from the battery should be used to power the CLP/MLP/grid.
This computer connects to the internet using the home/business internet connection. This connection allows each system to be managed remotely. The current power plans and rules being followed by the system can be modified. An example of remote management would be before an expected storm, the system could be told to charge the battery and keep it charged in case the storm causes the local power to go out.

2.1.8 Measuring Point

The measuring point isn't a component as such, it is just the point where the Inverter/Charger, MLP and the grid are all connected to each other. If the MLP is requiring more power than the Inverter/Charger is providing (if any), then the remaining required power will be brought in from the grid. Power from the grid is also required if the Inverter/Charger is requiring power from the MP. In this case the grid will power the MLP, and supply some power to the Inverter/Charger. In the case that the Inverter/Charger is sending more power to the MP than the MLP needs, power will be sent back into the grid. In this case the owner of the house will be reimbursed for the power sent back into the grid.

2.2 Sofware - The API

There exists an HTTPS-based API for communicating with the solar panel systems. It uses a RESTful architecture. By writing custom POST, GET and DELETE HTTP requests, a system can be communicated with remotely. The API doesn't connect to a system directly, but instead connects via the system provider of the solar panel system.
The primary function of the API that I will be using during this project will be to query the state of a solar panel system to get information about battery charge levels and important power flows. These power flows include:

- The amount of power being generated by the solar panels (PV)
- The current power rate that the battery is charging/discharging
- How much power is being taken in from (or exported to) the grid

- The amount of power that the appliances in the house are using (measuring the CLP and MLP)
- And several other flows

The API also allows power plans to be set and viewed. These plans include rules such as how quickly the battery should be charged, and at what charge level power should be exported to the grid.
A new version of the API was introduced last semester. This brought new features and is an overall benefit to my project. However it broke some earlier code and I needed to rewrite it. This caused a minor setback.

Chapter 3

Technology Research

There are several design decisions to be made in this project. Vector wasn't specific about what technologies or devices I should use as long as it does what they want it to, does it efficiently, and is appropriate. This chapter covers some of the choices I made while I was researching and developing.

3.1 Device

The first and main choice was to decide on what hardware I would run my software on. Vector had said it needs to be a device that runs locally in the house/business. The device needs to meet certain requirements. The device has certain requirements, these include:

- Being relatively cheap
- Supporting a programming language that has an HTTPS library
- Has enough processing power to make decisions
- Has an electrical interface to send a control signal to a relay

3.1.1 Arduino

The initial idea as suggested by Vector was to use an Arduino as the device to implement the decision making program on. Arduino is an open-source single board microcontroller [2]. The processor on an Arduino consists of either an 8-bit AVR CPU, or 32-bit ARM CPU. Arduino models vary quite significantly, but are mostly all are used for hobby hardware projects with basic computations. Programs on Arduino are written in C or C++.

Although it has the electrical capability of sending a control signal and the computational power to make decisions, it doesn't support any HTTPS libraries. This is the main reason why I couldn't use Arduino as there would be no way use the API. Without being able to use the API there would be no way to query a solar panel system and make decisions.

3.1.2 Raspberry Pi

After realising Arduino would not be a suitable choice, the next device I looked into was the Raspberry Pi. Raspberry Pi is a credit-card sized single board computer [3]. Raspberry Pi's have a 700MHz ARM CPU and either 256 or 512 MBs of RAM depending on the model. Raspberry Pi's are significantly more powerful than Arduinos. Raspberry Pi's can run a full version of Linux that is designed to use run on ARM CPUs. Within a Linux distribution, programs can be written on various programming languages, such as Java, Python or C++. Having an operating system running means more overhead when compared to Arduino, but it makes software development much easier. Raspberry Pi's have several General-purpose input/output pins that can be used for sending control signals. Raspberry Pi's can connect to the internet/networks using an inbuilt 100Mb/s Ethernet port, as well potentially using a Wi-Fi dongle in on one of its USB ports.
I chose to use Raspberry Pi as the device as it fulfils all the technical criteria, is easily purchasable, and relatively cheap.

3.2 Raspberry Pi Operating System

Once I had chosen Raspberry Pi, I had to decide on the operating system I would install on it. Unlike Arduino which had a basic operating system built into it, Raspberry Pi needs an operating system to run. There were certain requirements for this Operating system, it needs to be:

- Stable
- Easy to install
- Support a programming languages capable of HTTPS requests

3.2.1 Raspbian

Raspbian is a custom version of Debian Linux that is designed to run on a Raspberry Pi [4]. It is currently the most popular and supported operating system. Raspbian comes

pre-installed with many programming languages and development environments.
I chose to use Raspbian as it is the most supported operating system, stable, and supports many programming languages.

3.2.2 Pidora

Pidora is a custom version of Fedora Linux that is designed to run on a Raspberry Pi [5]. Like Raspbian, Pidora comes pre-installed with various programming languages.

3.2.3 Arch Linux

There is also a version of Arch Linux that runs on Raspberry Pi's [6]. Arch Linux requires considerable more setup than Raspbian and Pidora, and comes installed with less programming languages.

3.3 Programming languages

Once I had decided on using a Raspberry Pi as the device to run the program, the next decision was what programming language and platform to choose from. The language needs to :

- Support HTTPS queries for the using the solar panel system API
- Be able to run for a long time without crashing or suffering memory leaks
- Be relatively efficient when performing computations
- Have libraries for using the Raspberry Pi's GPIO ports (sending the control signal)

3.3.1 Python

Python [7] is a multi-paradigm programming language that runs on a Virtual Machine. The use of Python is encouraged on Raspberry Pi's as it is pre-installed on the device and many libraries for raspberry pi specific features (e.g. GPIO) are pre-installed too. Python would have been a good choice for the project, but due to my limited experience with it I chose not to use it.

3.3.2 Java

Java [8] is a multi-paradigm programming language that runs on a Virtual Machine. Java is also encouraged for use on Raspberry Pi's, however not as strongly as Python. Java is pre-installed on Raspberry Pi's, however some libraries (e.g. GPIO libraries) need to be installed from external sources. I chose Java over Python for this project as I feel it is functionally-equivalent with Python for the software that I need to create, however I have far more experience and confidence to code with it. One downside when compared to using Python is that I will need to find and reference external libraries (which are easily available) for using the GPIO interface. I feel this downside is more than out-weighted by the benefits of using a language that I am more confident in.

3.3.3 C++

C++ [9] is multi-paradigm programming language that runs natively on hardware. Like the Python and Java, C++ is supported on raspberry Pi's and it is pre-installed. Like Java, external libraries would need to be found and referenced to use the GPIO ports, or code could be written in a C++ class to use them without too much effort. Code written in C++ generally runs faster when compared to Python or Java code, but I would need to put more effort into memory management to avoid memory leaks. I chose not to use C++ as I don't have too much experience in it and I would have had to put more effort into writing the program. The speed advantages I would get from writing the program in C++ would be small as my program is not computationally demanding.

3.4 Status Query Result format

Originally with the v1 API, XML was the only format that the retrieved system status data would be in. However with the release of the v2 API, there is now the option of using JSON as the format for the retrieved data to be in, as well XML. I needed to make a decision between using XML or JSON as for retrieving system status data. I needed to pick the format that:

- Uses the least amount of bandwidth
- Is computationally efficient

3.4.1 XML

Extensible Markup Language (XML) is a data encoding language that uses opening and closing tags. An example of data entry returned for how much power being produced by the solar panels would be: <pvWatts>890</pvWatts>. This requires 22 characters. I sampled the returned status in XML format at certain times (both day and night) and I found on average the format used around 1920 characters.

3.4.2 JSON

JavaScript Object Notation (JSON) is also a data encoding language, however it uses name-value pairs instead of tags. JSON is more light-weight than XML and this will reduce data transmitted each query. An example of data entry returned for how much power being produced by the solar panels would be: "pvWatts":890, this requires 14 characters and represents the same amount of data as in the XML sample for the same value. I sampled the returned status in JSON format at certain times (both day and night) and I found on average the format used around 916 characters. This is less than half the size of XML's average of XML's average of 1920 characters.
I chose to use JSON over XML as JSON uses less characters and therefore less internet bandwidth will be consumed. It also has been found to be less computationally intensive to parse JSON when compared to XML [10].

Chapter 4

Solution Design

This chapter looks at the overall design of my solution that will enable me to answer the research question described earlier. It also describes design decisions that have been made during my project. These designs include both hardware and software designs.

4.1 Hardware

Once I had decided on using a Raspberry Pi as the platform for making decisions and sending a control signal, the next step was to design how the Raspberry Pi will be physically installed with other components. The device that we are currently looking at using to switch a mains load on or off is a solid-state relay. These are capable of switching AC loads for standard appliances in a home/business.
There are two main design choices about how to connect the Raspberry Pi and relay together. At this point in time I'm not sure which will be used, potentially both could be used. I will know more about this once I meet with Vector again to discuss the project.

4.1.1 Combined components

One hardware design is to combine both the Raspberry Pi and the relay together in one box. There would be mains power going into it for both the Raspberry Pi and the relay. There would be a standard mains power socket on the box that an appliance could plug into. There would also be a Ethernet connection for the Raspberry Pi to connect to the internet. This hardware design would be simpler to transport and demonstrate as it is all in one box. Figure 4.1 is a basic logical diagram of a combined design.

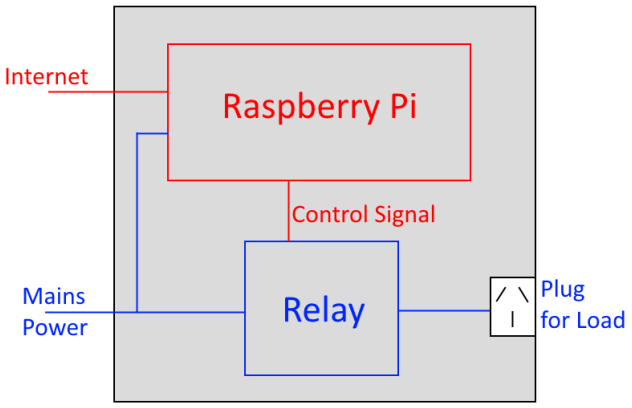

FIGURE 4.1: Combined - Logical Diagram

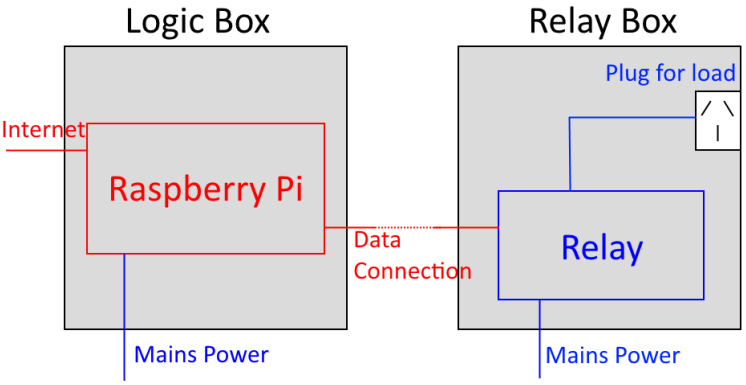

FIGURE 4.2: Separate - Logical Diagram

4.1.2 Separate components

The other hardware design is to separate both the Raspberry Pi and relay into separate boxes. The "logic" box would contain the Raspberry Pi. The relay box would contain the relay and have a mains power plug embedded into it. The two boxes would be linked so the control signal can be sent from the Raspberry Pi to the relay. This link could be either a basic wire, or potentially a type of wireless signal. Figure 4.2 is a basic logical diagram of a separate design.

4.2 Software

There is a lot more flexibility with the design of my software compared to flexibility with the design of the hardware. With most programming, there are often several ways of producing the same outcome.

4.2.1 Program Overview

The software that I will be developing as part of the solution will consist of a single program that runs on the device with the sole purpose of making decisions about switching a load on or off. The software will be written in Java and run on Raspbian Linux. This decision-making program will be launched when the Raspberry Pi boots up as this will allow it to initiate and run without human interaction. I am choosing to implement my decision-making program as a command line program. There was no point making this a GUI program as the basic decision-making algorithm needs to run discretely on the device without displaying visual information or requiring human interaction. There is the possibility of producing an optional GUI in the future to allow users to remotely interact with the program, however this is not necessary for now and will only be needed if Vector requires a new feature. A command line program will allow many optional input parameters when the program is launched. This is handy for development and testing, but will have limited benefits as the program will be auto-run without interacting.

4.2.2 Logic Overview

The core algorithm of my program involves a sequence of steps that will repeat at certain intervals. Initially I will set the algorithm to repeat every 30 seconds, however this can easily be changed. The core algorithm steps to be repeated at regular time intervals include:

1. Using the API to fetch the currently power status of a solar panel system
2. Logging the power status information for use in the near-future
3. Using the power state information along with recently power history to make a decision about whether or not to turn a load on or off
4. Send a control signal via a GPIO port to a relay based on the result of that decision
5. Find how long the above steps took, subtract this time from the interval amount (e.g. 30 seconds) and sleep for the resulting amount of time

There will also be certain calculations performed on logged data to enable it to be used by future iterations of the core algorithm. These calculations will be performed at the end of each day, or potentially other times when needed.

4.2.3 Solar Panel System API

The program will need to be able to use the solar panel system API in order to fetch the power status of a system. As this is an HTTPS based API, I will need to manipulate HTTPS queries to match the requirements of the API, this includes modifying HTTPS headers. I will develop methods that perform these HTTPS query manipulations to simplify the development of this program. Being able to use a single method to perform functions like querying the status of a system will simplify the algorithm logic.

Chapter 5

Current work

This chapter looks at the work I have completed up to the end of the 2nd week of the 2nd Semester (ending Sunday 3rd August).

5.1 Raspberry Pi configuration

I have obtained a Raspberry Pi Model B and installed Raspbian onto it. I have left most of the settings as default due to the fact it is already mostly configured to how I want it to be. One thing I noticed that was the default DHCP configuration wasn't working with my network setup so I configured it with a static IP address. Once it was connected on my network I can SSH onto it to run my program, and remotely control files using FTP. Java 7 was already pre-installed with Raspbian so there is no need to install it.

5.2 Program Structure

As mentioned in the design chapter, I have implemented my program as a command line program in Java that runs on the Raspberry Pi (running Raspbian). The basic structure of my program is in place, however some parts are lacking. The program currently consists of a basic core-algorithm as described earlier. This core algorithm consists of a loop that repeats at certain time intervals, currently 30 seconds by default. The current logic for each loop includes querying the state of a solar panel system using the API, making a very basic decision about whether or not to turn a load on or off and then sending a control signal to the GPIO. This power-state information is also currently logged for future use as it will be used to help develop the improved decision logic. By default, the core-algorithm loop will continue to forever. Upon launching the program a command line parameter

can be used to specify how long it should loop for (currently for development purposes). For each iteration of the loop, the algorithm times how long these previously mentioned steps took to execute (including network delay when querying the server) and then sleeps for the corresponding amount of time. For example with the default setting of looping every 30 seconds, if these steps took 2.5 seconds it will sleep for 27.5 seconds to ensure it loops every 30 seconds. If there is a problem connecting to the network then it will just log that no information has been made and wait for the next loop iteration.
The decision making logic is currently very basic and at this point it only looks at the grid import/export amount. It decides to switch a load on if the system is exporting power to the grid and switch it off if the system is importing power from the grid. This current logic is more of a placeholder as I haven't had the time or sample data to work on this logic yet. This will comprise a large amount of the future development of this program.

5.3 Solar Panel System API use

I have currently created 4 basic methods for using the solar panel system API. These include:

- Initiating a session with the API, this needs to be done first or else no other methods will work
- Querying the status of a system in JSON format, I plan to use JSON as the return format when querying the server due to reasons given earlier
- Querying the status of a system in XML format, although I plan to use JSON as the return format I will leave the XML code in the class for now
- Disconnecting from the API session, this is optional but good practice

5.4 GPIO output

I am currently able to successfully send a control signal using the Raspberry Pi's GPIO ports. I installed the Pi4j library to do this. The Raspberry Pi isn't hooked up to a relay yet, however I have tested the GPIO output with an LED. The LED successfully lights up when my program wants it to. Later when the Raspberry Pi is connected to a relay, I will just send the same signal as I am doing currently.

5.5 Classes

I currently have 5 Java classes that I have created as part of the decision making program.

5.5.1 DecisionMaker.java

This class contains the main method of my program and is called when the program is launched. The main purpose of this class is to define logon information about the solar panel system that will be interacted with, initiate some program variables and then launch the computation. This currently creates an instance of an APIConnection then launches a LoopTask thread with the APIConnection instance as a parameter. This class also will be able to control the LoopTask thread and cancel its execution early.

5.5.2 LoopTask.java

LoopTask.java is a subclass of thread. This class contains the higher level logic of my program – the core algorithm. It is designed to loop at given time intervals (currently 30 seconds by default). The LoopTask thread is created with a handle to an APIConnection instance that it will use to query information about a solar panel system.
At times I have noticed a loop iteration can take more than 30 seconds to execute. In the event of this happening the thread will not sleep at all, it will immediately being the next loop iteration. The program keeps a track of how many loop iterations have occurred, and at which time each iteration should sleep. By doing this the program can correct its timing and return to a regular pattern in the event that 1 or more iterations take more than 30 seconds.

5.5.3 APIConnection.java

This class contains the 4 wrapper methods mentioned earlier for interacting with the HTTPS based API. This class enables the LoopTask thread (where the main high-level logic is) to interact with the API using simple methods, rather than having several lines of manipulating and parsing HTTPS queries each time the class wants to use the API.

5.5.4 Logger.java

This class contains wrapper methods for writing information to a .csv file. It was made to simply file writing for logging. Currently the class separates its logs into 1 .csv file

per day for simplicity, in such a way that they are nicely ordered. At the moment this class is currently being used by a LoopTask instance to log information about the system at regular intervals. This information will be used to work on and improve my decision-making algorithm. At the moment I am logging every part of information about power flows in case they will be required later. In the future I will only be keeping recent logs about relevant information due to storage constraints on the Raspberry Pi.

5.5.5 LoadController.java

This is a simple class that simplifies GPIO output on the Raspberry Pi. It contains methods for switching the GPIO output on or off. These methods will be used for controlling the load that will consume the excess power. An instance of the LoopTask class will use these methods after it has finished making a decision. This class uses one of the external libraries that I am currently using, Pi4j, as Raspbian doesn't include a Java library for GPIO interfacing.

5.6 External Libraries

I tried to limit the amount of external Java libraries used during this project. I only used an external library when it enabled my program to do something it couldn't do with stand Java libraries, or I felt it brought significant benefits.

5.6.1 GSON

GSON is a Java library for JSON parsing [11]. It is created by Google. I needed to use a JSON parsing library as Java surprisingly doesn't include one with the standard libraries. I chose GSON as it is easy to use and fairly efficient when compared to other external Java JSON libraries.

5.6.2 Pi4j

Pi4j is a Java library for using the GPIO ports on a Raspberry Pi [12]. I needed a library for interacting with the GPIO ports as Raspbian doesn't include one for Java by default. This is the only downside I have found when I chose to use Java over Python, as Raspbian includes a simple Python GPIO library. I chose Pi4j as was the only GPIO library I could find for Java.

Chapter 6

Future work

The first semester has seen a fair majority of the effort put into this project go into understanding the system and communication with Vector. In the second semester of my project, there will be a bigger focus on the development of my solution. This chapter looks at the future work of the project

6.1 Decision making logic algorithm

I will need to develop and vastly improve the decision making algorithm logic for my program. This algorithm will be potentially the most computationally intensive part of my program, however I don't feel it will be intensive enough to justify using a native language like C++. At the moment I haven't implemented any intelligent design making logic; decisions are currently made just based on whether the system is currently exporting power or not. This current algorithm was just a placeholder of the algorithm to be developed.

A rough idea of the algorithm to be implemented is as follows. I will create logs of certain power values at regular time intervals throughout each day. The values will include the battery charge level, current power produced from solar panels (PV), and potentially other values. At the end of each day I will perform some computations, these will include noting what time sunrise occurred by looking for the first solar/PV value higher than 0 and what time sunset occurred by looking for what time after midday that the solar/PV values dropped back down to 0. Then for each time interval I will calculate how much power will be produced from this point in time until sunset. I will also look at how much power will be charged into the battery from this point in time until sunset occurs. There will most definitely be other calculations at this point. These calculations will be used for future days to try to estimate how much power will be produced from a certain point

in time. For example at 11am on the current day the algorithm will look at data and calculations from previous days. It will look at the state of the system at 11am on other days and try to estimate how much excess power will be produced this current day. When this has been intelligently estimated, the algorithm will make a decision about whether to turn the load on or off and if it is turned on how long it should be on for.
This is just a rough idea for now and I will give a more in-depth description in my final report when I have developed and implemented this algorithm. The development of this algorithm will be an iterative process and I am expecting to make many modifications and refinements to make this as accurate as possible.

6.2 Connecting the Raspberry Pi to a relay

Currently the only hardware I have is just the Raspberry Pi and its basic components. These components include a power source, microSD card, Ethernet cable, and LED for testing GPIO function.
As mentioned before, there are 2 current hardware designs for connecting the Raspberry Pi for a relay. A decision will need to be made about whether to keep all the components together in one box, or separate the Raspberry Pi into one box and send a control signal to separate box that contains a mains-switching relay. Vector will need to give me input about this. I will give a more in-depth description of the choices made and the resultant box/boxes in the final report.
Irrelevant of design, the relay we are currently looking at is a solid-state relay that's capable of switching a mains AC appliance. My academic supervisor, Ulrich Speidel, will be able to set up the mains wiring of a prototype as he is qualified to do so.

6.3 Sending information back to Vector

Vector has mentioned before that they would like to know information about the load-switching decisions that will be made by the device. They don't want to just know the outcome of the decisions, but also they would like to know the conditions of the system and the reason the decision was made to be what it is. Vector wasn't too specific about the method of sending the information back to them, but it could be in the form of a database that I store entries in. More information about sending information back to Vector will be covered in the final report where I will give a more in-depth description of this.

Chapter 7

Conclusion

This small chapter is just to conclude and sum up the report.
I am currently making steady progress on the solution that I am creating as part of this project. This solution will help me to answer the research question. My solution will be in the form of a decision-making device that will be able to switch an optional load on or off via a relay. This optional load will consume excess power produced by a system. The current progress in this project has involved learning about Vector and the solar panel system, researching technologies to be used for the solution, designing the solution and creating the start of the solution.
There is still a lot of work to be done on this project as mentioned in the future work section. Although plenty of effort and time has currently gone into the project so far, there has been a lot of overhead with communication, technology changes and understanding the system. Last semester I had some communication issues with Vector which restrained project progress slightly, however this semester the project is back on track with communication. A setback occurred during the introduction of the new version of the API as it broke some earlier code and I needed to rewrite it. I have also spent a lot of time trying to understand the solar panel system that I'm working with, and its' relationship with XML/JSON outputs. Due to these multiple forms of project overhead, not a large amount of time has been put into producing the solution. However with the remaining weeks of the project there will be a considerable increase in the time and effort put into the development of the solution.
The remaining weeks of this project will see me complete and evaluate my solution. I will look at the impact it has and how well it answers the research question.

Bibliography

[1] Vector. Vector - about us. http://vector.co.nz/about-us.

[2] Arduino. Arduino - home. http://www.arduino.cc/.

[3] Raspberry Pi. Raspberry pi. http://www.raspberrypi.org/.

[4] Raspbian. Raspbian: Frontpage. http://http://www.raspbian.org/.

[5] Pidora. Pidora - raspberry pi fedora remix. http://pidora.ca/.

[6] Arch Linix ARM. Raspberry pi - archwiki - arch linux. https://wiki.archlinux.org/index.php/Raspberry_Pi.

[7] Python. About python — python.org. https://www.python.org/about/.

[8] Oracle. Learn about java technology. http://java.com/en/about/.

[9] cplusplus.com. A brief description. http://www.cplusplus.com/info/description/.

[10] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: A case study. 2009.

[11] Google. Google-gson. https://code.google.com/p/google-gson/.

[12] Pi4J. The pi4j project. http://pi4j.com/.