Technical Support Remote Viewable Camera Device

September 2015

Author: Jamie Gilbertson (658405)

Project Dissertation Submitted to Swansea University in Partial Fulfilment for the Degree of Master of Science.

Department of Computer Science, Swansea University

Summary

This project was undertaken with the intention of creating a portable device that can broadcast a live video feed of a user's viewpoint. This device is intended to be reproduced hundreds of times to create a network of devices, and so a system to manage these devices was also to be created.

A suitable device capable of broadcasting this live video feed has been created, and includes remote control and voice chat features. A Registrar Server has also been created to track all of these devices and their locations. A local client has been created to allow a user to interact with and manage all of the devices registered to the Registrar Server, and gives access to view and modify a database of details specific to each device and the location the device is situated.

All of these sub-systems work together to function as a network of viewable portable cameras that can be monitored and managed from anywhere in the world (provided an internet connection is available) from an easy-to-use client.

The majority of the specifications set out in the project have been well met. The solution is potentially more expensive that originally intended, and the audio communications are limited to one-way, but every specification has been met to some degree. The final solution offered is well able to suit its purpose as a proof-of-concept system to justify further expenditure to create a fully-realised deployable system.

The software created during the development of this project was designed to suit generic purposes, and as a result has applications beyond those set out in the original project proposal.

Declarations/Statements

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

This dissertation is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by giving explicit references. A bibliography is appended.

I hereby give consent for my dissertation, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed: ..... Date: .....

Acknowledgements

I would like to express thanks to Dr Neil Harman, whose advice and steady guidance throughout the development of this project was very much appreciated. His patience and talent for constructive criticism was very helpful. I would also like to note his willingness to put aside time for meetings on very short notice, as well as his rapid response to communication, was also appreciated.

I would also like to thank the staff of the Welsh Video Network organisation for their consistent feedback and aid provided during the design, development, and testing stages of the project, and for their help in providing access to their equipment and services. Their feedback often provided a guiding light to give the project direction and purpose.

Table of Contents

Lists of Figures and Tables

Figures

Tables

Abbreviations Used

ALSA	Advanced Linux Sound Architecture
app	Mobile Application
APT	Advanced Package Tool
AR	Augmented Reality
FPS	Frames Per Second
GUI	Graphical User Interface
HD	High Definition
HTTP	Hyper Text Transfer Protocol
ID	Identification Number
IDE	Integrated Development Environment
IP	Internet Protocol
JAX-WS	Java API for XML Web Services
JDK	Java Development Kit
mAh	milli-Amp hours
MVC	Model View Controller
OS	Operating System
PBX	Private Branch Exchange
RPi	Raspberry Pi
SD	Secure Digital
SIP	Session Initiation Protocol
SSH	Secure Shell
UDP	User Datagram Protocol
UI	User Interface
USB	Universal Serial Bus
VoIP	Voice over Internet Protocol
WLAN	Wireless Local Area Network
WSDL	Web Service Definition Language
WVN	Welsh Video Network

Introduction

Proposal

The original proposal for this project was submitted by the Welsh Video Network **[1]** organisation, which provides video conferencing services via ~100 public sector sites across Wales. Not only do they provide and manage this service, they also provide technical support to those using the service, usually remotely. The support offered is available for troubleshooting issues with integrated video systems, involving both physical pieces of equipment and wiring issues. The person requesting support is usually not technically skilled themselves, which can cause some communication issues when discussing the problem over the phone with WVN staff. The proposal itself was to develop a piece of equipment that would allow the technical staff to see a live view of what the end-user can see, so that they might more easily understand and diagnose issues. This equipment would ideally be wearable and also allow easy verbal communication, and preferably be relatively low cost to produce.

Aims and Objectives

After further discussion with the Technical Support team at WVN, clear aims and objectives for the project were set. The solution should:-

- Be able to provide live video feed of user's location
- Allow two-way easy voice communication
- Require minimal user interaction to activate any on-location device
- Be wearable (and therefore portable) for easy use
- Speed up the troubleshooting process
- Have a desktop client to view and manage remote devices, and also to view information on remote locations
- Be able to remotely control the device to turn it off, or update software etc.
- Be scalable to work across more than 100 locations
- Be relatively low cost to produce and deploy

Existing Products

There are a variety of existing technologies that can satisfy some of these aims, though in most cases not satisfactorily or not completely.

The most obvious potential solution would be to take advantage of the popularity and power of smart phone devices. Most are easily capable of high quality video recording and transmitting video over the internet, and are inherently capable of verbal communication. All that would be required would be an app that handles the routing of the video call, which could even be an existing app like Skype **[2]**.

One of the main problems of this solution is that it requires any users wanting technical support to own a smart phone compatible with the application, which means creating an application that supports a wide range of devices on a variety of operating systems. It would also mean that the technician would have to walk the user through downloading the application before benefitting from its use, slowing down the troubleshooting process. Slowing down the troubleshooting process completely defeats the purpose of the project as a whole, and this makes it difficult to justify pursuing a course of action like this. An application that could range across these devices could also be prohibitively expensive due to the licensing fees involved with becoming a developer able to release and distribute applications on each major operating system's app store. A developer license on Apple's app store alone costs over £65 a year **[3]**.

The way around these problems would be to have already available a smart phone stored at each location with the app pre-loaded. This way the application would only need to support a single type of device and a single operating system. This would be potentially expensive, however, and discussions with WVN staff concluded with a preference against the idea of purchasing a smart phone for every location.

Another pre-existing and well established technology is that of the IP Camera **[4]**. Most come with software already installed that allow remote control of certain functions of the device, with many products even allowing panning and zooming of the camera for a better view, and some even offering voice communication. This offers the prospect of an ideal amount of interaction from the individual requesting assistance (i.e. none). However it would not be possible for it to properly show their viewpoint, nor would it be portable, as these devices are designed as static security cameras that are usually bolted onto a

surface and plugged into mains power. Most also don't come with software or methods to manage more than one of these devices, and due to their finished-product nature it is impossible to modify them to work in such a way that would allow a desktop client to monitor their activity or provide details about their location.

Interestingly, a somewhere similar product (though designed for a different overall purpose) exists in the form of Google Glass **[5]**. As a device it could satisfy nearly every requirement set by the project proposal. It has an embedded camera, in-built speakers and microphone, is inherently wearable, can connect to the internet, and would likely have been able to run apps just like a smart phone could. This could, ultimately, have been the perfect solution that could provide all of the requested functionality. However, disregarding the fact the product was discontinued due to the largely negative response it received from the general public, the cost of purchasing a Google Glass device for over 100 locations is well beyond the cost limitations set in the proposal.

The solution ultimately decided upon was to use a Raspberry Pi **[6]**. The Raspberry Pi is effectively a small, lightweight, low-cost computer board that has a variety of accessories available for it, including a specialised camera accessory, and USB and other ports to add generic accessories. A WiFi adapter would allow it to connect to the internet and a headset would allow voice communication. Due to the fact that the Raspberry Pi 2 is powered via a Mini-USB port, it can even be powered using a smart phone power bank, meaning it can be portable and even wearable, as it is a small device. It has low power requirements and so can be powered for a significant length of time by a relatively cheap power pack. Also, the fact that it is effectively a small computer means it is inherently capable of running applications with no extra development license costs, allowing for modification for management by a remote desktop client. In case WVN considers directing more resources towards the issue being addressed by this project, a solution built upon a Raspberry Pi would serve as an excellent proof of concept to justify allocating more resources to expanding upon this project to develop a potentially better system.

Of course, none of the hardware solutions listed above actually satisfy the criteria regarding a desktop based management client. There are a variety of existing pieces of software designed for similar purposes, such as security camera monitoring software, but again, it is very expensive to purchase or rent software like this for a device that may well

be very rarely activated. It is possible to develop a custom desktop application in a cross-platform compatible language (such as Java) to serve as a management client of all the remote devices registered to it. This also allows the creation of a custom database to store location details, or the integration of an existing one.

Specification

The specifications of the project were broken down into 3 tiers of features; base, standard, and ideal. In this way each feature could be categorised and prioritised, where base was an absolute requirement of the finished product, standard was an intended requirement, and ideal features are useful extras and would only be added if time allowed for their addition. Some features listed in higher tiered specifications (base being the 'lowest' and ideal being the 'highest') are iterative versions of a more basic version of that feature; for example 'high frame rate video feed' is iterative on 'high quality viewable live feed of user's viewpoint'.

Base Specification

- High quality and remotely viewable live video feed of user's viewpoint
- Desktop-based method of viewing camera feed
- Method of voice communication
- Operable with minimal end user interaction
- Portable
- Be relatively low cost

Standard Specification

- Scalable to at least 100 separate devices
- High frame rate video feed
- Low latency two way voice communication
- Stand-alone client that identifies location and lists site details
- Wearable device
- Remote control of the device
- Secure

Ideal Specification

- Self/Remotely updatable software
- Client that can access existing ticketing system database to display location-specific frequent issues and solutions

Design

Overall Design

To satisfy these specifications an overall sub-system structure was designed:-

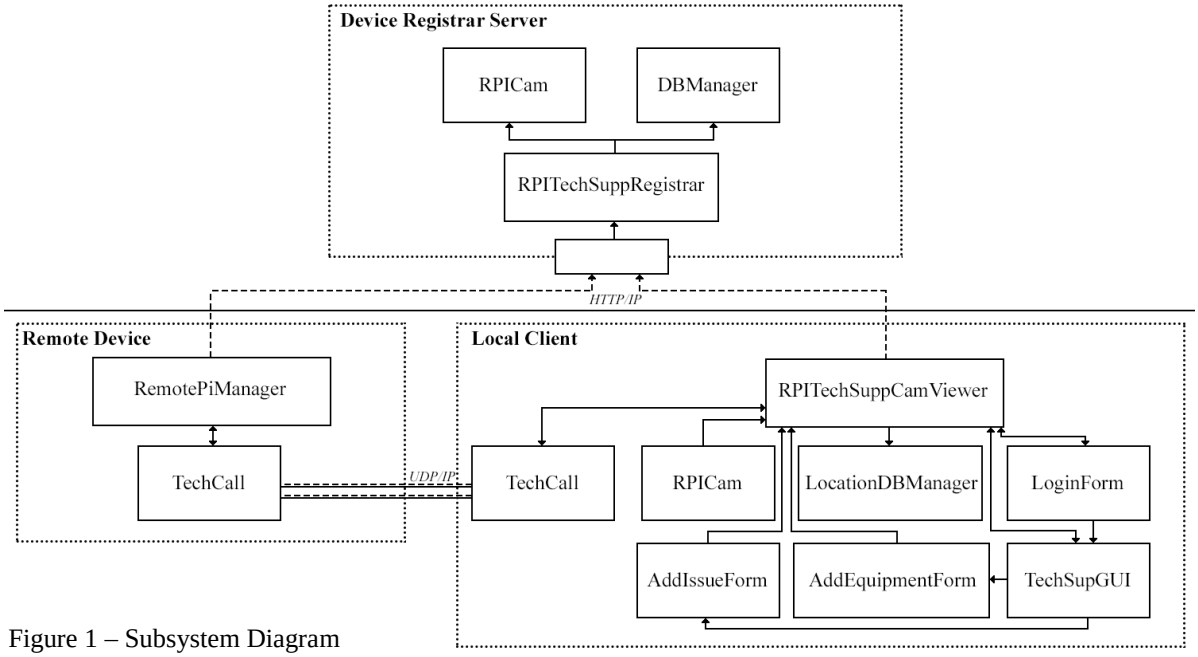

Figure 1 – Subsystem Diagram

As can be seen, the whole system was divided into 3 separate sub-systems.

Remote Device

The remote device should be responsible for providing:

- Live high frame rate/quality camera feed
- Portable/Wearable device
- Voice recording/sending and receiving/playback
- Self/Remotely updatable software

- Information about its status

To satisfy these requirements using a Raspberry Pi 2 as a base the following hardware was needed:-

- Camera
- Microphone and speakers (or a headset)
- Battery/power pack
- Wireless internet adapter

The software was also needed to:-

- Expose the video feed of the camera so that it was remotely accessible over the internet
- Record voice and send it to the technician's client as well as receive audio from the same and play it back
- Inform some other remote application of its status and how it can be accessed

Part of satisfying these requirements is connecting the device to the internet, and forwarding the necessary ports to allow it to actually perform its function. However, beyond the proof-of-concept demonstration, configuring internet access for each device deployed is beyond the scope of this project.

Device Registrar Server

To keep the project scalable, and to ensure the entire system isn't accessible via only one machine, it was decided that a server application running on a machine with a static IP address would be ideal. Without this, either every remote device would need a static IP address, or the machine running the client would need a static IP address. This way, remote devices can have IP addresses that change regularly (like is common practice with dynamic IP addresses) and the client can run on any PC. The remote devices can simply ping the server when they're active, and the server can register their IP address and make it available to the client so that the client can communicate directly with the device.

The server should ultimately be responsible for:-

- Serving as a registrar of all devices
- Providing information about registered and online devices

- Storing IP addresses of online devices
- Routing voice calls
- Block unauthorised access to information

To provide these services a server application would have to be developed, most likely as a Java JAX-WS **[7]** web service running on a GlassFish **[8]** server along with a database to store data. It is important to note that deploying to an actual server machine is beyond the scope of this project, as WVN will need to set up their own services to run the system.

Local Client

The client should serve as the interface through which the technician would view and interact with any devices. It should be responsible for:-

- Overall device management interface
- Displaying device information
- Displaying location-specific information
- Serving as call interface
- Voice recording/sending and receiving/playback
- Providing view of live camera feed
- Limited remote control of devices
- Block unauthorised access to management tools

To provide access to these functions a client would have to be developed, probably as a Java Swing Forms **[9]** application with a database to store location-specific data. It will also need to integrate the server's WSDL **[10]** information so that it can access the methods located on the server.

Class Design

Full Javadocs are available online for all classes and contain a full list of all methods and their purpose **[11]**.

GUI Design

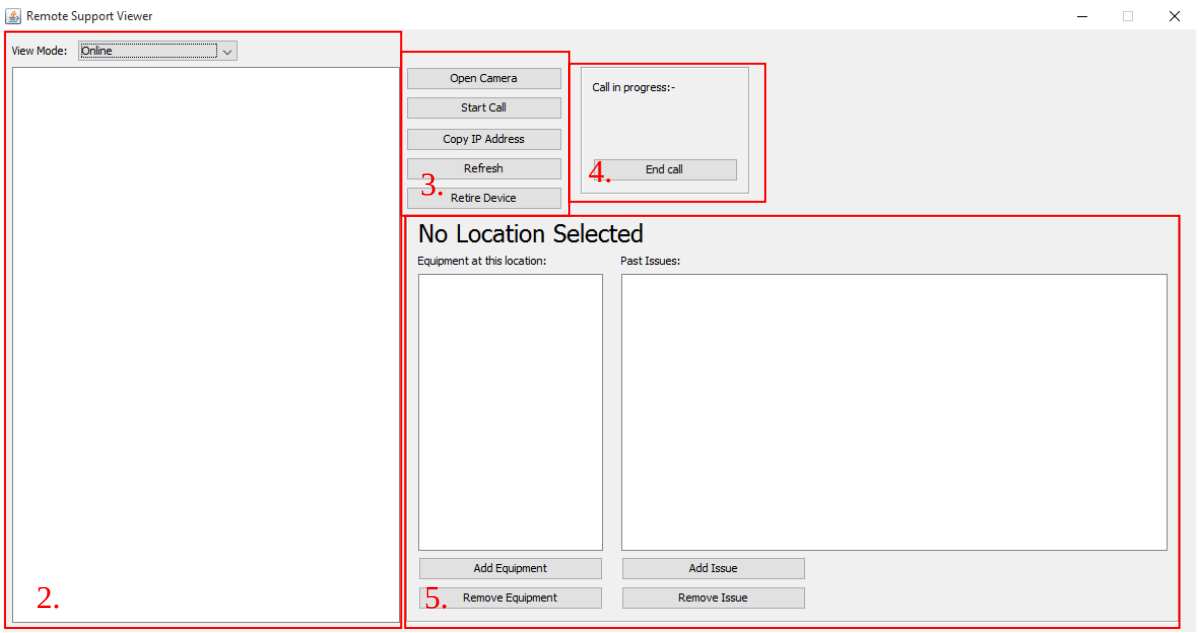

Figure 2 – First draft GUI design

1. Before being able to access the GUI, a password window must be completed
2. JList to display the list of devices. Combo-box above allows a user to switch between online devices view and all registered devices view
3. Provides buttons to access functions relating to the list of devices
4. Shows the details of an ongoing call and provides button to end the call
5. Shows the location-specific details of the currently selected device and provides controls to add to and remove from the issue and equipment lists for the location.
6. Adding issues and equipment is handled via separate forms that pop up when the respective buttons are pressed

Implementation

Hardware Implementation

A Raspberry Pi 2 Model B **[12]** along with the official Raspberry Pi Camera Module **[13]** and Raspberry Pi 2 Model B **[14]** Case were purchased to serve as the basis of the portable portion of the project that would reside at the site that required technical support. The following reasons were instrumental in making this decision:-

- The Camera Module is more than capable of capturing high resolution and high frame rate video
- The Raspberry Pi 2 Model B can be powered via mini-USB and has low power requirements
- The device is relatively cheap

Along with this device the following parts were acquired to serve all the required functions:-

- An EasyAcc Ultra 10000mAh **[15]** power pack was procured to serve as a portable power source
- A Microsoft Lifechat LX-3000 **[16]** was borrowed from WVN to provide audio support for the voice chat portion of the project
- A Dynamode Nano USB WL-700N-RXS WiFi Adapter **[17]** was bought to allow the Raspberry Pi to connect to Wireless Networks
- A Kingston 16GB Micro SD Card **[18]** was purchased to act as the Hard Drive of the Raspberry Pi

Raspberry Pi Software Setup

The first step to set up the Raspberry Pi was to select and install an Operating System. An OS called Raspbian **[19]** (based on the Debian Linux distribution [**20]**) was chosen due to its status as the officially supported OS for Raspberry Pi, and its widely adopted use by Raspberry Pi users. Once the image file for this OS was downloaded, it was written to the SD Card using a tool called Win32DiskImager **[21]**. Once written, the SD Card simply needed to be inserted into the Raspberry Pi and it could be powered and used.

While a Raspberry Pi is simple to configure for operation via SSH, first-time set up of the Raspberry Pi requires a screen and keyboard to provide the interface to configure SSH, wireless connections, and a handful of other settings. A small portable screen and keyboard were provided by the developer for the duration of the development process to serve this purpose, though these accessories are not required for use in the finished product.

Once powered, the Raspiconfig utility was loaded and settings to enable the Camera Module, to enable SSH, to expand the size of the partition accessible to Raspbian, and to

enable WiFi were set using the utility. Once these settings were enabled, the Raspberry Pi was rebooted and connected to a Router (again supplied by the developer) via Ethernet cable to allow for access via SSH (a technology that allows remote command-line control of devices). Once powered up again, an attempt was made to configure WiFi settings, which is performed by editing the WPA_Supplicant.conf file located in /etc/WPA_Supplicant. The following snippet was appended to the file and saved:-

```
network={
ssid="NETWORKNAME"
psk="PASSWORD"
}
```

Adding this snippet should have allowed the Raspberry Pi to connect to the router wirelessly, but this didn't work. After a brief investigation it was found that the Nano USB WiFi Adapter was faulty, and would quickly overheat and malfunction. A search revealed that this was a common problem with this adapter. To remedy this, an Edimax EW-7811UN Wireless Nano USB Adapter **[22]** was purchased as a replacement, chosen largely due to its purported reliability and widely adopted use in other Raspberry Pi projects.

Once the new WiFi adapter was installed, the Raspberry Pi quickly connected to the router, and SSH was possible over the Wireless Local Area Network provided by the router. SSH control was established using Putty **[23]**, a common tool used for this purpose. A similarly common tool called WinSCP **[24]** was used to transfer files to and from the Raspberry Pi.

Raspberry Pi Implementation

The first part of the device portion of this project tackled was providing a live video feed over the internet. It was decided due to time constraints that it would be far better to find and use existing software to provide this functionality rather than to design and build programs that probably already existed. After reading various documents (blogs, articles, and forum posts) regarding other Raspberry Pi projects used for IP Camera, WiFi Camera, and Security Camera purposes (similar to those specified by this project), it was found that the most commonly used package was Motion **[25]**. Motion provided a basic video feed accessible over the internet, and had a large variety of configuration options, including password protection and motion detection, among other features.

Installing packages to the Raspberry Pi is usually as simple as giving a single command to the Raspberry Pi, in this case: `sudo apt-get install motion`. Upon receiving this instruction, the APT **[26]** provided by Raspbian looks up the package in its stored Debian Sources file found in /etc/apt/sources.list (which is updated using `apt-get update`), and then calculates all dependencies of the package that weren't already installed, and downloads and installs the package and all its dependencies. However, this requires an internet connection to work, and it was proving impossible to connect the Raspberry Pi to the internet provided by the university, Eduroam **[27]**. This was mostly due to the access restrictions of the network, as Eduroam is a strictly administrated service. Unfortunately, the developer lived in university accommodation and had no other access to the internet. Instead, an attempt was made to install Motion and its dependencies manually by finding them in the Raspbian Repository Archive **[28]**, writing them to the SD card, and de-packaging them on the Raspberry Pi. This did not work, and progress was halted until internet access could be established. A full reinstall of Raspbian was then performed to remove the mismatched software and dependencies and to reset any configuration options that were potentially incorrectly changed in efforts to connect the Raspberry Pi to Eduroam.

Internet access was eventually achieved by registering the Raspberry Pi to another internet service provided by the university, Swis-lite, that is usually reserved for game consoles. Once connected, running the command `sudo apt-get install motion` correctly installed the program to the Raspberry Pi. It was at this time that it was discovered that since the program was designed to be used as Security Camera software, it deliberately delivered only low frame-rate video to remote viewers. Tests showed that this frame rate was unacceptably low (2-3 FPS) and could not provide higher FPS due to the limitations of the software, since it was originally designed to only provide an overview of the area live, and to record video for retrieval later only when motion was detected.

A deeper search of software available for the Raspberry Pi revealed a package called RPi Cam Web Interface **[29]**. Though it utilised Motion to provide some of its functionality, it was designed not as a Security Camera-like application, but was intended to provide a full HD high frame rate camera feed. As well as this, the package provides a comprehensive interface that allowed the user to change the desired frame rate, capture resolution, brightness and contrast settings, and a host of other configuration options, as well as

access to some system tools, including buttons to shut down or reboot the Raspberry Pi (this interface would serve well to allow limited remote operation of the Raspberry Pi without full SSH connection). All of this functionality was provided as a web page viewable in any browser and accessible by the IP address of the Raspberry Pi through port 80. This software seemed like it would well satisfy the video requirements of the project. To be clear, this package would provide a web browser-based interface to view a live feed of the camera view. The Jamie Gilbertson (hereafter referred to as 'the developer') had no hand in developing this package.

As it was not an archived Raspbian package, installation of RPi Cam Web Interface was not as simple as that of Motion, and required the source code to be cloned from GitHub **[30]** and installed using the following set of commands:-

```
git clone https://github.com/silvanmelchior/RPi_Cam_Web_Interface.git
cd RPi_Cam_Web_Interface
chmod u+x RPi_Cam_Web_Interface_Installer.sh
./RPi_Cam_Web_Interface_Installer.sh install
```

At this point an installer menu provided access to installing the program itself as well as a pre-configured Apache client to provide the web capabilities.

It was found that the application would automatically record video whenever it detected any motion. This was undesirable as it could quickly and inadvertently fill up all of the available memory space on the SD card. Since there was no option to directly deactivate this, the motion capture settings were tweaked so that motion had to be detected for over 10000 video frames before triggering recording, and 'motion' was changed to be defined as a difference of over 10000 (arbitrary units) between frames. This effectively made it impossible for 'motion' to be detected and impossible for it to last long enough to trigger recording if it was.

From this point on, RPi Cam Web Interface required no further configuration. It provided a sufficiently high quality video stream that was expandable to see more detail, as well as all the interaction utilities advertised.

Next, a lightweight application called RemotePiManager was created by the developer as a part of this project and installed to the Raspberry Pi stored in /RemotePiManager. This program accesses a file called details.conf stored in the same directory and registers itself to the server with the location specified on the first line of details.conf, and receives and stores a unique ID. Once registered, the application then pings the Registrar every second

to inform it that the Raspberry Pi is online and to allowed the Registrar to register the IP address by which the device could be accessed.

Once RemotePiManager was tested and was working as intended, focus shifted to the voice chat requirements. Since this function is common to both the remote device and the local client, the implementation of Voice Chat has been given its own section on page 24.

The last step of setting up the Raspberry Pi was to add the startup command to /etc/rc.local, which would instruct the Raspberry Pi to run the RemotePiManager program when it boots.

Device Registrar Server Implementation

The server is supposed to function as a Registrar, to offer the remote devices whose IP address may well change over time a static address to ping. Upon receiving the pings, the server stores the IP addresses of the devices and serves them to the Camera Viewer client upon request.

The server was implemented in Java as a Web Service using JAX-WS **[7]** libraries. The Netbeans **[31]** IDE was used for development, along with a GlassFish [8] server for testing deployment.

A database was also created to store a list of registered devices. This database has 1 table with 2 columns, ID stored as an integer, and LOCATION stored as a VARCHAR with a maximum of 30 characters.

The first version of the server was originally created to offer methods to register and retire devices, and to return lists of the devices currently online, and lists of all registered devices. To do this the server program utilises three classes, RPITechSuppRegistrar, RPICam, and DBManager, all of which were designed and written by the developer as a part of this project.

The main RPITechSuppRegistrar class is the port of call for devices and clients, and contains the methods for registering and retiring devices, and for returning lists of devices.

RPICam serves as an Object to store details of individual devices, and is used in both the server and the local client. RPICam stores the location and ID number of a given device,

as well as its IP address and a Date variable containing the precise time the last ping from the device was received. It also offers timeout methods to check whether the time since the last received ping is greater than the timeout parameter passed in its constructor. The RPITechSuppRegistrar class stores instances of RPICam in an ArrayList to keep track of the online devices. RPICam also offers methods for converting its contents into Strings for easy transfer, and contains constructors that accepts those strings.

DBManager is the class that manages the device database, and offers methods to add and remove entries, to search for specific entries based either on an ID or a location String, and for returning the entire database in String format that was the RPICam constructors can parse.

As the managing class, RPITechSuppRegistrar is where most of the processes occur.

Device-Registrar Process Flow

When registering, the device sends its location to the server. Upon receiving this information the server checks to see if a device is already registered at that location, as only one device is allowed to be at each location. If another is already registered at the location the server sends back an error code. If the location is open, the server assigns the device a unique ID number (and sends this pack to the device) and registers it in the database. The `retire()` method offers the reverse of this, and removes devices from the database.

Once registered, the device can then call the `pingAlive()` method to tell the server that it's online. When receiving a ping, the server uses the ID passed to it to grab the device from the database, and extracts the IP address of the device from the MessageContext of the HTTP request. Using this information it then constructs a new instance of RPICam with the IP address and adds it to the list of online devices (`onlinePis`), and then updates the lastPing variable of the object to the current time. If the device has already pinged and has not timed out and been pruned from the list, then the entry in the online device list is updated with the new time. The following (refactored for easy viewing) code was written to extract the IP address from the MessageContext:-

```
1| MessageContext pingContext = context.getMessageContext();
2| HttpServletRequest pingRequest =
(HttpServletRequest)pingContext.get(MessageContext.SERVLET_REQUEST);
3|
4| String temp = pingRequest.getRemoteAddr()
```

Local Client-Registrar Process Flow

A number of methods were also designed to be used by the local client to display information to the Technician using it. The `getOnlineList()` method can be used to retrieve the list of all the online devices. Calling this method causes the server to run through the list of online devices, checking all of them to see if they have timed out and pruning them from the list if they have, before sending the list back to the client as a parse-able String.

The method `getRegisteredPiList()` allows access to the list of all devices registered in the database. Much like `getOnlineList()`, it returns the list in a parse-able String format.

Further Development

At this point the server was deployed to a temporary Glassfish server running on Netbeans. The server saw very little modification beyond debugging after this point.

One of the additions made was to support passing of IP addresses to devices so that a two-way VoIP call could be established between the Technician's machine and a remote device, but was deprecated in favour of the device waiting to receive a UDP packet and extracting the IP address to return packets to from that. These methods and their related member variables remain implemented in the RPITechSuppRegistrar class in case voice communication is expanded to a more complex program in the future that may require methods like these.

The only other addition was to add password support. For a user to access any functions from the client a password must be entered. The password should be embedded into the HTTP request, and would then be extracted by the server and checked each time a method is called. A `login()` method was added so that controls in the client could be locked until receiving a confirmation of the password entered from the server. Beyond this, if calls are made to other server methods that don't contain the correct password the server returns "PASSWORDNOTFOUND" strings to the requester. The following code was written to extract the password from the HTTP header.

```
1| private boolean checkPass(){
2|       MessageContext authContext = context.getMessageContext();
3|
4|      Map headers = (Map)
                  authContext.get(MessageContext.HTTP_REQUEST_HEADERS);
5|
6|       List pw = (List) headers.get("Password");
7|       String pass = "";
8|       if(pw != null){
9|           pass = pw.get(0).toString();
10|     }
11|
12|     if(serverPassword.equals(pass)) //pass is global String variable
13|         return true;
14|     else
15|         return false;
16| }
```

Local Client Implementation

The client was split into a variety of classes as can be seen in Figure 1. These classes are AddEquipmentForm, AddIssueForm, LocationDBManager, LoginForm, RPICam, RPITechSuppCamViewer, TechCall, and TechSuppGUI. All of these classes were designed and created by the developer.

The GUI was the first part of the client that was actually designed and implemented. The GUI was designed using the built-in Netbeans form designer by the developer as a part of this project. The following controls were added (see Figure 2):-

- A list view to display the list of devices (online and registered)
- A combobox to allow the user to switch between online devices view and all registered devices view
- Buttons to open camera views, to start a call, to copy the IP address of the selected device to the system clipboard, and to refresh the list
- A panel to display lists of the past user issues and equipment at the selected location

The GUI itself is contained in its own class, TechSuppGUI, as MVC principles were borne in mind while designing the application. This class not only handles all the required events thrown by the GUI controls, but offers a few methods to allow the controller class to enable and disable parts of the UI, as well as update them with new information as needed. A method to return the index of the device selected in the device list is also

available, as the index of the device list corresponds to the index of the same device stored in an ArrayList in the controller class. This GUI is also responsible for validating user input before sending it to the controller class.

The controller class, RPITechSuppCamViewer, offers a variety of functions and acts as the access point to the server and the database that stores location specific issues and equipment. This class stores a local list of devices that changes depending on whether the user is viewing online or all registered devices, as well as the web service and GUI references. The methods it offers include:-

- A refresh method to update the device list with devices dependant on the selected view
- A method to open a webpage to view the camera feed of a device
- A method to copy the stored IP address of an online camera

The following code was written to open an instance of the system default web browser pointing to the IP address of the selected device:-

```
1| if(Desktop.isDesktopSupported()){
2|      try{
3|          Desktop.getDesktop().browse(new URI("http://" +
                                camList.get(index).getIPAddress()));
4|
5|     }
6|     catch(URISyntaxException | IOException e){
7|          System.out.println(e);
8|     }
9| }
```

And the following code was written to copy the IP address of a device to the system's clipboard:-

```
1| StringSelection stringSelection = new StringSelection
                                    (camList.get(index).getIPAddress());
2|
3| Clipboard board = Toolkit.getDefaultToolkit().getSystemClipboard ();
4| board.setContents (stringSelection, null);
```

This list was later expanded as more features were added. The first of these features was the ability to perform calls to devices. Methods for initiating and ending calls as well as dealing with call timeouts are offered by the TechCall class, and related UI objects to

perform these functions are available in the GUI. Further discussion of this class is available in the dedicated Voice Chat Implementation section.

The next addition was that of the location database. It was originally intended that an existing ticketing database filled by WVN's own ticketing system would be integrated and the panel for view location details would simply pull data from this database, but unfortunately WVN was unable to provide a copy of the database in time for it to be integrated into the application before the end of the development phase. Instead, a custom database was designed by the developer as part of this project and a class called LocationDBManager was created by the developer to access it, as well as methods added to RPITechSuppCamViewer to serve as an interface between the GUI and LocationDBManager. This database is made up of 2 tables, EQUIPMENT and ISSUES. EQUIPMENT has 2 columns, LOCATION which stores a VARCHAR with a maximum limit of 30 characters, and EQUIPMENT which stores a VARCHAR with a maximum of 50 characters. ISSUES also has 2 columns, LOCATION which stores a VARCHAR with a maximum limit of 30 characters and ISSUES which stores a VARCHAR with a maximum limit of 500 characters. It later emerged that the delay in receiving WVN's database was partly because it was due to be decommissioned shortly after the delivery of this project (for reasons unrelated to this project), which meant that the generic database approach was actually preferable.

Buttons were added to the location information panel in the bottom right of the GUI to allow a user to add and remove issues and equipment from the database. The buttons to add entries to the database open instances of the AddEquipment and AddIssue form classes when activate. These classes actually provide the interface for typing and adding new database entries as well as validate user input before submitting it for entry into or removal from the database. The reference to these instances are stored in the TechGUI class and are simply set visible and invisible when needed instead of new instances being created and destroyed.

Lastly, the requirement of a password was added to access the user interface to add a basic level of security. This was implemented as a new GUI class called LoginForm that appears before the full GUI is loaded. The user has to enter the correct password which is then sent to the server (via the login() method of the RPITechSuppCamViewer class which adds the password to the header of every HTTP request sent to the server in future)

for verification. If the server confirms the password, then the full GUI is loaded. The following code was written to add the password to the HTTP request header:-

```
1|      Map<String, Object> reqContext =
            ((BindingProvider)port).getRequestContext();
2|
3|      String temp = password;
4|      Map<String, List<String>> header = new HashMap<>();
5|      header.put("Password", Collections.singletonList(temp));
6|      reqContext.put(MessageContext.HTTP_REQUEST_HEADERS, header);
```

Voice Chat Implementation

During the course of development, and during the research for original specifications document, it was expected that a VoIP **[32]** program would be installed onto the Raspberry Pi and integrated into the local client running on the Technician's computer. Some existing programs offered the ability to complete calls using the IP address of each caller, which was ideal. When it came to actually implement voice chat, further inspection was carried out and found that most VoIP solutions required installing several extra applications, such as a soft-phone application and an IP PBX, among other applications, none of which could be properly integrated into the client. Even using Java SIP libraries **[33]** to create a client that could be integrated was more complex than needed, since these libraries were designed to handle the handshaking between devices to determine what media and devices they had in common to communicate with. This seemed especially true when considering the fact the developer had direct control over both applications and could simply decide in advance exactly how they would communicate. Both of these solutions suffered from the same issue, installing or creating the applications and somehow integrating them into the system seemed too complicated to properly implement given the limited time available to complete the project

Ultimately it was decided a very simple class could be created and added to both the client and device that would allow very simple two-way audio communication by recording small frames of audio and sending them using UDP packets and then playing them back on the receiving end. Each caller would run two threads, one that would record audio to a Byte Array and send it to the other caller, and another that would receive incoming Byte Arrays and play them back. At its core, the class that was created to manage these calls, called TechCall, relied on Java DataLines **[34]**. The class requests a DataLine from the system's audio mixer that matched a specific audio format (in this

case, 8KHz, 16-bit, mono-channel, signed, and little-Endian). For recording, a TargetDataLine is required, and a SourceDataLine is required for playing back audio. Both of these objects natively write and read Byte Arrays, perfect for use with standard Java UDP socket libraries.

There are two constructors for the TechCall class. One of these requires no arguments to be passed to it, and is used by the Raspberry Pi to create an instance of TechCall at startup. The other constructor requires an IP address passed as a String. When the startCall() method is called it checks to see if an IP address has been set by the constructor, and if it is, it initiates the call to the given IP address. If no IP address was set by the constructor the class waits for an incoming UDP packet before starting the playback and recording threads.

During testing, it was found that a problem arises when setting up the DataLines on the Raspberry Pi. The following snippet of code written to request a TargetDataLine throws a LineUnavailableException once it reaches the indicated line:-

```
1| AudioFormat format = new AudioFormat(8000, 16, 1, true,
                              false);
2|
3| DataLine.Info targetInfo = new DataLine.Info(TargetDataLine.class,
                               format);
4| targetLine = (TargetDataLine) AudioSystem.getLine(targetInfo);//error
5| targetLine.open(format);
```

The targetLine object is used to record audio, and there are no problems with this when running on Windows machines or when running on the Linux machines available in the Swansea University Linux Lab. This problem is present only on the Raspberry Pi, regardless of whether it is running Oracle JDK or OpenJDK. The problem is believed to be related to the interaction between Java's AudioSystem **[35]** libraries and the Raspberry Pi's hardware, or perhaps with the ALSA **[36]** audio mixer that manages Raspberry Pi audio in Raspbian.

Considering the popularity of Debian, and of Raspbian, and the fact that the libraries used to implement this TechCall class are included in the standard JDK, the fact that this wouldn't work was completely unforeseen. Research carried out upon identifying the problem revealed that it is a fairly common issue and no solution could be found that was currently available to developers. Due to the nature of the problem and the fact that it was

completely unforeseen meant that no research was carried out into whether a library included in the standard Java libraries might not work.

As stated earlier, using Java SIP libraries to implement VoIP would probably have encountered the same problem when trying to access any microphone connected to the Raspberry Pi.

This problem has been discussed further in the Retrospective Analysis section.

Suffice it to say, the Raspberry Pi would not allow recording of microphone input, but it could play back the sound frames it received. This allows one-way communication from the Technician to the end-user, which would be sufficient for the Technician to relay instructions, but doesn't really allow the end-user to ask questions or explain anything.

In reality, the process of using this device would usually begin with the end-user calling Tech Support on the phone for help with their problem, and only after exhausted other options would the Technician request that the user activate the camera and put it on. Ultimately, it's likely that the Technician and end-user could and would maintain the phone call they were having, the exception to this being if the original call had been made on a wired phone. If they are already on the phone with each other, they could simply continue the phone call while the end user just wears the Raspberry Pi without a headset. Failing this, one-way communication is still possible using the Raspberry Pi and computer client, but the end-user would have to communicate with hand gestures, or by writing down more complicated questions and showing them to the camera, a cumbersome process but still a functional one.

Deployment

An actual set up of the system is relatively simple. The first step of setting up a new system using this project should be to deploy the server application to an actual server. It is recommended to deploy to a GlassFish server using Netbeans, as this is how the project has been developed and Netbeans offers useful tools for integrating the web service with the other applications used in this project.

Before deploying any software, the 2 necessary databases should be set up. If a user wishing to deploy this system wishes to changed any database, table, or column names,

these should be changed in the DBManager and LocationDBManager classes. Once these classes are adjusted, no other classes need to be modified to accommodate these changes.

The RegisteredRPIDB should contain 1 table called REGISTEREDPIS that has 2 columns, ID stored as an integer, and LOCATION stored as a VARCHAR with a maximum of 30 characters. The RemoteLocationDB should contain 2 tables, EQUIPMENT and ISSUES. EQUIPMENT must have 2 columns, LOCATION which stores a VARCHAR with a maximum limit of 30 characters, and EQUIPMENT which stores a VARCHAR with a maximum of 50 characters. ISSUES must also have 2 columns, LOCATION which stores a VARCHAR with a maximum limit of 30 characters and ISSUES which stores a VARCHAR with a maximum limit of 500 characters.

Once the server is deployed, the web service WSDL needs to be integrated into the RPITechSuppCamViewer project and RemotePiManager as a package called CamRegistrar. This can be done in Netbeans by selecting the project, right clicking the project name, clicking new, then clicking Web Service Client, and then following the instructions on screen.

For RPITechSuppCamViewer, the project simply needs to be cleaned and built and the RPITechSuppCamViewer.jar found in the ‘dist’ folder of the Netbeans project can be used as a standalone client.

For RemotePiManager the setup is more difficult. Before trying to deploy this application, the Raspberry Pi SD card should be written using the cloned Raspbian image available on the GitHub project page **[37]**. This image includes all the settings and commands necessary to run the project. The login username is ‘Pi’ and the password is currently ‘Raspberry’. It is recommended to change this password while setting up the Raspberry Pi.

Once the Raspberry Pi has been set up, the RemotePiManager project needs to be cleaned and built now once web service has been added. Once done, all you need to do is drop the RemotePiManager.jar found in the Netbeans project’s ‘dist’ folder into the RemotePiManager folder of the Raspberry Pi (when logged in as user ‘Pi’) using a program such as WinSCP.

At this point, the location (or name) of the device should be written to the first line of RemotePiManager/details.conf stored on the Raspberry Pi using nano **[38]**, or a text

editor of your choice. If the location is already registered, a new line informing you of the error will be written to details.conf next time you run the RemotePiManager program (which starts automatically when the Raspberry Pi boots).

Once these steps are completed, the system should be ready to use.

Testing

Methodology

There were 3 practical tests carried out to evaluate the suitability of the program. One was designed to assess the UI and its ease of use. The second was designed to assess how well the overall system worked between a user and a technician. Another test was also performed to test the power requirements of the device and the stability of the battery pack.

UI Ease of Use

This was a very simple test. Anonymous users who had never seen the interface before and had no technical support experience – but were computer literate - were placed in front of the UI for the first time and asked to perform a short series of tasks relating to the UI. Each task was assigned a time limit, and if users consistently took longer than the limit to carry out a task, then the UI was deemed badly suited to the task. The time between being asked the question and completion of the task was then measured. There were 2 categories of tasks, one where the user would simply point out where controls were, and another where they would actually have to carry out an operation. For the first category a special version of the UI (see Figure 2) was created in which all controls were enabled at all times, as some controls are not enabled (and therefore greyed out) unless a certain process is followed, and this may confuse the user in tests, while in practical use this would not pose an issue. For the latter category a live version of the client was used. The following tasks were set (the time limits for each are laid out in brackets next to the task):-

1. Point out the button to open a camera view (3s)
2. Point out the button retire a device (3s)
3. Point out the button to start a call (3s)

4. Point out the button to end a call (3s)
5. Point out the button to refresh the device list (3s)
6. Enter the password and log into the client (10s)
7. Switch the camera view to ‘all registered’ mode (3s)
8. Select the device at Cardiff (3s)
9. Add a piece of equipment called ‘computer’ to the list for this device (15s)
10. Remove the piece of equipment from the list for this device (3s)

Performing physical tasks

In this test 2 users worked together to perform a simple task in a room they were both familiar with. One user (hereafter referred to as the technician) sat in front of a computer running the client the Technician would have access to, and had already been familiarised with. This user would also have been shown how the device works. The other user (hereafter referred to as the remote-user) was to stand in the room with the device next to them, but they did not know how the device worked. The technician was the same individual throughout the tests, while the remote-user changed for each test, to simulate the technician who would be used to the process and a remote-user who wouldn’t know what was happening.

The task allocated was to turn on a small lamp in the corner of the room. The technician knew where the lamp was and what it looked like, but was not allowed to tell the remote-user. The remote-user was to call the technician, and the technician would walk them through using the device, and then finding and turning on the lamp. Since only one-way communication was possible, remote-user was advised to use hand signals to communicate if they needed.

Power Consumption and Supply Reliability

This was a simple test to assess the viability of using a battery pack to power the Raspberry Pi as it rendered video, broadcasted it to the internet, and ran the required software. Every 10 minutes for 4 hours the Voltage and Current being supplied to the device was measured using a small USB Voltage and Current meter **[39]**, and the power supplied (in Watts) calculated. This was designed to give a reliable idea of whether the device could operate in a stable manner over a period of time long enough to complete a troubleshooting process. 4 hours would be more than enough time, and if the device

functioned reliably and the power consumption was stable during this time, then it would be known that the battery pack was a viable power supply.

Results and Analysis

UI Ease of Use

1. Point out the button to open a camera view

Table 1 – Results of task 1 in UI Ease of Use test

User	**1**	**2**	**3**	**4**	**5**	**Average**
Time(s)	2.60	1.12	0.66	1.25	1.61	1.45

Time limit: 3 seconds

Average time: 1.45

Max time: 2.60

Result: Users found button with ease, placement deemed suitable

2. Point out the button to retire a device

Table 2 – Results of task 2 in UI Ease of Use test

User	**1**	**2**	**3**	**4**	**5**	**Average**
Time(s)	1.91	1.01	0.71	0.96	1.14	1.15

Time limit: 3 seconds

Average time: 1.15

Max time: 1.91

Result: Users found button with ease, placement deemed suitable

3. Point out the button to start a call

Table 3 – Results of task 3 in UI Ease of Use test

User	**1**	**2**	**3**	**4**	**5**	**Average**
Time(s)	1.51	2.01	1.21	1.62	1.43	1.56

Time limit: 3 seconds

Average time: 1.56

Max time: 2.01

Result: Users found button with ease, placement deemed suitable

4. Point out the button to end a call

Table 4 – Results of task 4 in UI Ease of Use test

User	**1**	**2**	**3**	**4**	**5**	**Average**
Time(s)	1.57	0.40	0.75	0.84	0.91	0.89

Time limit: 3 seconds

Average time: 0.89

Max time: 1.57

Result: Users found button with ease, placement deemed suitable

5. Point out the button to refresh the device list

Table 5 – Results of task 5 in UI Ease of Use test

User	**1**	**2**	**3**	**4**	**5**	**Average**
Time(s)	1.02	0.70	2.51	1.63	1.42	1.46

Time Limit: 3 seconds

Average Time: 1.46

Max time: 2.51

Result: Users found button with ease, placement deemed suitable

6. Enter the password and log into the client

Table 6 – Results of task 6 in UI Ease of Use test

User	**1**	**2**	**3**	**4**	**5**	**Average**
Time(s)	6.45	4.52	4.81	4.93	5.17	5.18

Time Limit: 10 seconds

Average: 5.18

Max time: 6.45

Result: The limit was set high for this test, as the password entry is designed to be an obstacle. Even so, users still completed task well within time limit.

7. Switch the camera view to 'all registered' mode

Table 7 – Results of task 7 in UI Ease of Use test

User	**1**	**2**	**3**	**4**	**5**	**Average**
Time(s)	2.34	2.53	2.64	2.32	2.63	2.49

Time limit: 3 seconds

Average time: 2.49

Max time: 2.64

Result: Users completed task with ease, interface deemed suitable

8. Select the device at Cardiff

Table 8 – Results of task 8 in UI Ease of Use test

User	**1**	**2**	**3**	**4**	**5**	**Average**
Time(s)	2.78	1.16	1.16	1.83	1.71	1.73

Time Limit: 3 seconds

Average time: 1.73

Max time: 2.78

Result: Users complete task with ease, interface deemed suitable

9. Add a piece of equipment called ‘computer’ to the list for this device

Table 9 – Results of task 9 in UI Ease of Use test

User	1	2	3	4	5	Average
Time(s)	8.63	9.31	10.16	10.02	9.51	9.53

Time Limit: 15 seconds

Average Time: 9.53

Max time: 10.16

Result: Limit was high as users had to find a button that opens a form, then fill out the form, then press another button to complete the entry. Users all performed this task within the specified time limit. Interface deemed suitable.

10. Remove the piece of equipment from the list for this device

Table 10 – Results of task 10 in UI Ease of Use test

User	1	2	3	4	5	Average
Time(s)	2.17	5.83	4.22	3.62	6.12	4.39

Time limit: 3 seconds

Average time: 4.39

Max time 6.12

Result: Only 1 out of 5 users were able to complete the task within the time limit. The reasons for this and the adjustments made to compensate are discussed below.

Most of the tests showed that the UI would perform well with inexperienced users, let alone technicians who would be familiar with the interface. The biggest issue was found in task 10 where users failed to properly understand how to remove a device. Doing so required the user to select the piece of equipment from the list by clicking on it, then clicking the ‘Remove Equipment’ button. Every single user who tested the interface clicked on ‘Remove Equipment’ first, and when nothing happened they realised they had to select the equipment first before clicking the button. The feedback from the users

indicated that they expected to click the button and then select the equipment to remove from a pop-up menu.

To solve this issue, the text on the button was changed to ‘Remove Selected Equipment’. The same changed was made to the ‘Remove Issue’ button (to ‘Remove Selected Issue’), as this test was designed to serve as the test for both of these functions, since they are very similar.

Only one other issue cropped up that is not evident from the times measured, but was fed back by 3 out of the 5 users. They stated that the ‘Call Device’ button was confusing as it was not contained within the call interface panel, next to the ‘End Call’ button. The ‘Call Device’ button was actually located next to the device list, as its function was directly related to the currently selected device in the device list.

The ‘Call Device’ button was moved to above the ‘End Call’ and the text displayed on it changed to ‘Call Selected Device’ and is only enabled when a device has been selected.

The changes made can be seen in the Figure 3 below.

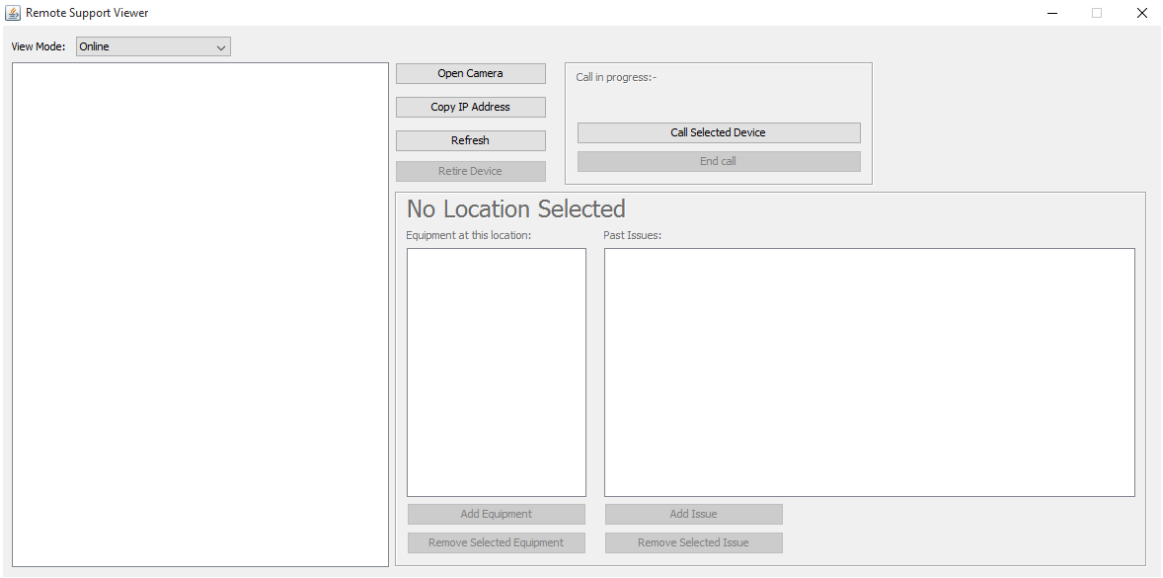

Figure 3 – Final GUI design

Performing Physical Tasks

The tests carried out to assess the suitability of the device in a pseudo-real world scenario were successful. The technician was easily able to guide the remote-users in performing their tasks. Due to the low latency of the audio and video visual feedback of instructions was immediate. It was found that instructions given had to be kept simple, as the user

could not ask questions to clarify. For example, instead of asking a user to go to the back right corner of the room, the instructions were more easily followed when broken down into turn around, go to the corner of the room on your left. The only feedback the user ever gave was when asked ‘can you hear me?’ to which most responded almost immediately with a thumbs up in front of the camera.

Power Consumption and Supply Reliability

The raw results of the tests are as follows:-

Table 11 – Results of Power Consumption and Supply Reliability test

Time	Voltage (V)	Current (A)	Power (W)
13:00	5.03	0.43	2.16
13:10	5.04	0.40	2.01
13:20	5.04	0.39	1.96
13:30	5.04	0.39	1.96
13:40	5.04	0.38	1.91
13:50	5.04	0.39	1.96
14:00	5.04	0.39	1.96
14:10	5.04	0.42	2.11
14:20	5.04	0.35	1.76
14:30	5.04	0.39	1.96
14:40	5.04	0.35	1.76
14:50	5.04	0.36	1.81
15:00	5.04	0.39	1.96
15:10	5.04	0.42	2.11
15:20	5.04	0.41	2.06
15:30	5.04	0.38	1.91
15:40	5.04	0.40	2.01
15:50	5.04	0.40	2.016
16:00	5.04	0.44	2.2176
16:10	5.04	0.39	1.9656
16:20	5.04	0.44	2.2176
16:30	5.03	0.46	2.3138
16:40	5.04	0.48	2.4192
16:50	5.04	0.49	2.4696
17:00	5.04	0.45	2.268

The results are also displayed overleaf as a line graph to display the change over time of the Voltage, Current, and Power consumption.

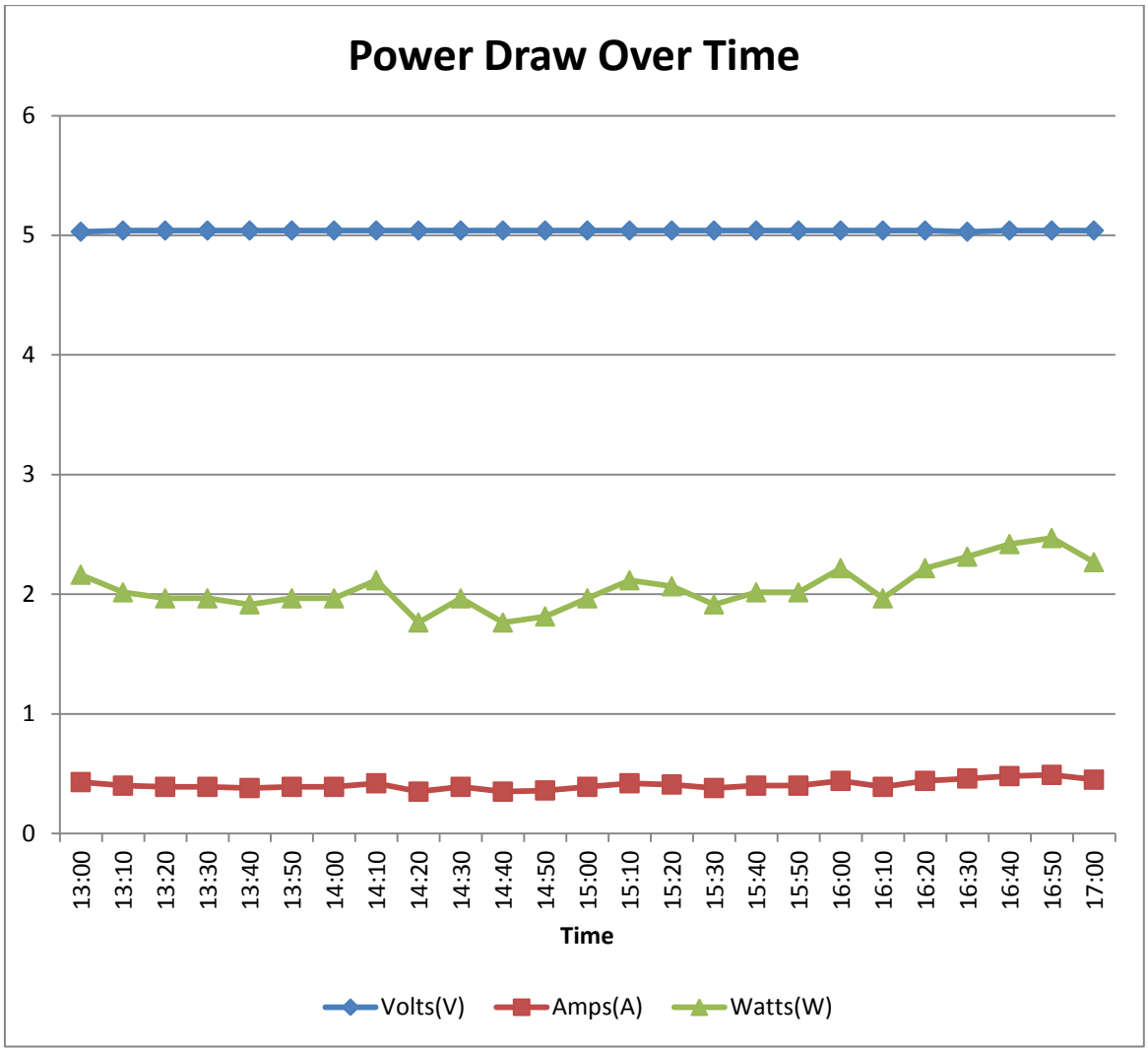

Figure 4 – Graph of Power Draw Over Time

The main issue this test anticipated to find was that over time the power being supplied by the battery would drop, but as can be seen from these results there was very little variation in power draw during the 4 hour period tested.

The largest variation showed an increase in power draw towards the end of the test, indicating that the battery pack was still well able to meet the higher demands of the device even after 4 hours of testing. We can see due the fact that the small variations evident are both increases and decreases that changes in power draw are more likely to be due to processes taking place within the Raspberry Pi rather than because of the battery pack's ability to provide power.

This is further supported by the fact that the current draw was the varying factor, while the voltage remained nearly constant. If the issue was with the battery, we would have

observed a drop in the voltage rather than the current draw, since the current is dependent on the load being supplied. However, we can see that the voltage only varies by 0.01 Volts during the entire duration of the test, while it is the current draw (which is dependent on the device) that is changing over time.

Feedback from Welsh Video Network

The project was demonstrated to WVN upon its completion. The feedback from them was generally positive; they were pleased with the quality of the video stream and the ease of use. They were disappointed with the one-way audio issue, but were happy to work around it.

The only issues they had was with the volume of the audio that was available being too low, which as explained is adjustable via settings on the Raspberry Pi itself. The battery pack was also an issue, as it was too large and cumbersome to be practical. It would be fairly simple, and actually is recommended later on in this report, to purchase smaller battery packs that would take up less space and would be cheaper.

The only suggestion for addition made was to program the device so that when a call is established a small noise or jingle is played over the speakers to let the remote-user know that a call has started.

Achievements

Satisfying Specifications

Base Specification

- There is a high quality live video feed of the user's viewpoint offered via a Raspberry Pi and a camera module
- The camera feed can be viewed directly from an internet browser window thanks to RPI Cam Web Interface package installed on the Raspberry Pi
- One-way voice communication is available via the management client and speakers connected to the Raspberry Pi device
- The only interaction with the device required by the remote-user is to plug in the power supply and wear the device

- The Raspberry Pi is powered by a portable battery pack that can be easily carried
- Device can be recreated for ~£90

The only part of the base specification that has perhaps not been satisfied is the specification regarding low cost. Using cheaper alternatives to recreate the device's functions (as most equipment used during development has been spare and already available to the developer or WVN in small quantities, but sets of components would need to be purchased to roll out the solution to more locations), the cost comes to just under £90 **[40-45 & 22]**. This would cost nearly £9000 to roll out the device to the ~100 locations serviced by WVN. It is important to note, however, that no hard limit was ever set on the cost of the device in the project proposal.

Standard Specification

- System is scalable to well over 100 devices, and is in theory scalable up to billions of unique devices thanks to the management client and server
- The video feed is high frame rate and can be adjusted using the interface provided by the RPI Cam Web Interface package
- Two way voice communication is not considered possible using current technologies due to issues with Raspberry Pi and recording audio with Java
- A stand-alone client to view and manage devices and their location specific details is implemented and user friendly
- The device is portable and has a small clip attached to it to allow it to be clipped to a remote-user's lapel
- The interface offered by the RPI Cam Web Interface package offers limited functions to record videos and images, as well as shutdown and reboot the system; further remote control can be established through SSH
- Basic security is offered through the requirement of a password to access many of the server functions

One concern of these features is the low-level of security. The security is limited in that nothing is encrypted, and any device is accessible as long as you know its IP address. However given that that the devices should only be active for a very short time, it would be difficult to accidentally (or even purposely) discover the IP address of a device without

use of the management client. Even if a malicious user did discover the IP addresses, they wouldn't necessarily know when the devices were active.

In consideration of the scale of the two-way audio issue, discussion of this problem has been explored in detail in the Retrospective Analysis section of this report.

Ideal Specification

- The software is remotely updateable via SSH using a program such as WinSCP, but will require activation of the device and manual updating
- Client cannot access existing database, as the existing database is due to be decommissioned shortly after the delivery of this project (for unrelated reasons); instead a custom database has been implemented to offer the same functionality along with tools to manage this database

Unique Achievements

This project is similar in many ways to a variety of other devices already available, but it differs in some key respects.

- The system is scalable and still manageable up to many hundreds of devices, and can be used in a variety of ways.
- The Registrar Server allows the devices to be located anywhere in the world (that has internet access) and still be manageable.
- The individual devices are portable and yet maintain reliability, video stream quality, and have relatively low power requirements that allow for long operation times.
- Once first time set up of the server has been performed, it is incredibly easy to expand the system with more devices, and management of the system remains simple.
- The system features an easy-to-use UI that offers a level of security.
- The software is open-source, and the system does not require any further costs after purchase of the hardware to run.

Potential Uses

This project was designed to be relatively generic, so that it could be applied to other situations and used in different ways to perform different goals. Some suggested potential uses are:-

- Use in disaster relief zones, either attached to robots or humans, to aid in search and rescue operations.
- Use as pet or baby monitors.
- Live streaming events from an individual's perspective, or even as full streaming cameras for temporary events that require fast set up.
- Portable security applications instead of hard-wired security cameras that can't be moved once installed.

With the addition of a 4G dongle or router **[48]** to provide wireless internet without needing a Local Area Network router these uses can be expanded even further to include:-

- Remote control drone/car/vehicle monitoring and viewpoint streaming
- Live streaming of an activity
- Use in police force to help monitor officer's activities and situation, and to aid in assessing situations and assistance needed in high-risk scenarios, as well as to help settle legal disputes between officers and potential plaintiffs.
- Use in fire departments to help monitor and combat fires, as well as monitor the situation of individuals combating the fire.

Retrospective Analysis

There are several limitations to this project, and there were a significant number of obstacles overcome to create the final working system.

Unfortunately, achieving two-way audio communication was unsuccessful. While one-way audio communication is possible, the fact that the remote-user cannot respond in kind to a technician giving them instruction has possibly made the usage of the device more difficult than it could have been.

Ultimately, this problem was caused by an unexpected problem when using standard JDK libraries when trying to record audio from a Raspberry Pi running Raspbian. As stated previously, this problem was completely unexpected considering the fact that Raspbian (which is a modified version of the Debian Linux distribution) is the officially supported OS for Raspberry Pi, and the fact that the JDK library used is included as a standard library. The developer ended up spending a significant amount of time attempting to find and implement a solution to the problem before it became clear that no solution could be found and implemented within the allotted time.

In reality, this limitation should have been identified much sooner than it was. If it had been indentified sooner, there may have been time to explore other solutions. The reasons for this lie in the fact that the approach finally taken to implement voice chat was itself the fallback position of the initial plan. Originally it was intended that VoIP applications, potentially using Java SIP libraries, would be installed to the device to facilitate communication, but the sheer complexity and number of programs necessary to implement such a system was deemed to be too great. As a result, development of the second attempt to achieve voice communication did not start until much later than it had been intended. It may have been wiser to instead have continued down the VoIP route despite its unnecessary complexity, but it is entirely possible that the exact same error would have been encountered, as the program would still have been in Java and audio would still need to be recorded by the device, but there would perhaps have been more time to find a solution.

There are other potential solutions to this issue. If there was more time, the option that would be explored would be to re-write the program in a different language (e.g. C# or more likely C++). However, it is possible a similar issue may have been encountered when writing the program in another language, and since it would require re-writing large portions of the project, it was deemed that this would simply take too long. It would perhaps be possible to only write a small program in a different language that would only record and send audio, then start it as a separate process from RemotePiManager and pass it the IP address to send the packets to, but this would likely be difficult to manage, and might well require a reboot of the device to restart a dropped or ended call.

For now, the options available to any user of this system are to either use only one-way communication to complete the task, or to continue the original phone call that started the

support request (since a remote-user has to make a call to technical support to even start this process).

According to feedback from WVN the occasion where the initial phone call is made using the land-line phone at the user's location it relatively common, and this land-line phone is wired onto the wall. In this scenario the best solution is probably a combination of the previous two. If the remote-user and the technician were to leave the original call running and use the device itself to communicate while the remote-user was out of reach of the land-line phone the user could simply pick up the land-line phone if they really needed to communicate anything particularly complex. Failing this, the remote-user could simply be requested to call again using a mobile telephone.

Given the problems with audio, and factoring in the overall cost of reproducing the device hardware, it may have been better to have built the system around a piece of physical hardware other than the Raspberry Pi. The potentially best solution would have been to instead use a relatively cheap Android phone. The Vodafone Smart First 6 **[47]** phone would only cost £50 per handset, which is significantly cheaper than the device produced by this project, and still has a high quality camera and obviously has the ability to host a voice chat. As stated earlier in the Existing Products section of this report, one of these devices would have to be purchased for each location due to the cost of and issues with supporting multiple smart phone platforms and getting users to install the application developed.

This avenue was considered at the beginning of the project, but the Raspberry Pi was used at the request of WVN. When using an Android phone at each location was suggested it was thought it would be too expensive. WVN also already had a Raspberry Pi available, so it was decided that it would be suitable enough to use the Raspberry Pi as a proof of concept rather than purchase extra equipment.

Future Recommendations

There are a number of features the developer would suggest as future additions to be explored, in no particular order:-

- A gallery of photographs of equipment, set ups, and floor plans of the locations devices are situated at, to be viewable alongside the location-specific issues and

equipment. This should help a technician understand the layout of the room and be more able to instruct a remote-user in navigating it.

- The addition of a small tune to be played when a voice call is established.
- Exploring the addition of a directed light or laser pointer so the remote-user knows where the camera is currently pointing.
- Automatic refreshing of the online list of devices; this was explored but abandoned due to the current selection being cleared when the list is refreshed, but does warrant further inspection.
- Online/Offline indicators added to the All Registered view of the management client's device list; this was briefly explored (as evidenced by the inclusion of icon1.png and icon2.png in the Netbeans project files) but abandoned as the system was not designed to handle such a feature and would require too many additions for a feature not defined in the original specification.
- Further exploration of a solution to the two-way audio issues; it is suggested to perhaps attempt writing a patch in a different language to serve as the audio recording part of the TechCall class.
- It has been suggested that Augmented Reality could be a useful addition in the far-flung future, allowing the Technician to show the user exactly what to do using an AR interface, however the technology for this does not exist in an available form at the present time.
- It would perhaps be advisable to explore the addition of *more* interaction from the remote-user. In the developer's experience, frustration and feelings of uselessness can often cause the remote-user distress and discomfort, and the idea of being 'led around by the nose' as it were could dissuade them from wanting to call support in the first place. It may be worth considering adjusting this specification in the future and give a remote-user more methods to help in the diagnosis and solution process.
- Addition of a 4G wireless internet dongle or router **[48]** to expand the potential uses of the project.
- Development of a dedicated device that would likely be cheaper to manufacture and wouldn't include unused features of a Raspberry Pi that contribute to its higher cost or prohibit certain features (i.e. two-way voice communication).

Conclusion

In terms of the original proposal, there now exists a device that allows a technician to actually see the problems a user is facing from their own viewpoint. There is a Registrar Server available to keep track of these devices wherever they are located, provided they are connected to the internet. All of the devices are manageable from a custom client that can be installed on any machine with an internet connection. The system offers a method for the technician to give instructions verbally, even if the listener cannot respond. It can be easily scaled up to every single location managed by WVN, and the devices themselves can be easily managed by use of a client running on the technician's own machine. Beyond that, the device can be remotely updated with software, and the system is capable of storing the common issues and the equipment at each individual location. The solution is potentially more expensive than originally intended, but despite its failings, serves as a suitable proof of concept to warrant further expenditure in developing a deployment ready product.

List of References

[1] Welsh Video Network. *About the WVN.* [Online]
Available at: http://www.wvn.ac.uk/en/aboutthewvn/
[Accessed September 2015].

[2] Microsoft, 2015. *About Skype.* [Online]
Available at: http://www.skype.com/en/about/
[Accessed September 2015].

[3] Apple, 2015. *How it works - Apple Developer Program.* [Online]
Available at: https://developer.apple.com/programs/how-it-works/
[Accessed September 2015].

[4] TechTarget, 2015. *What is IP camera?*. [Online]
Available at: http://whatis.techtarget.com/definition/IP-camera
[Accessed September 2015].

[5] TechRadar, 2015. *Google Glass 2: Everything you need to know*. [Online]
Available at: http://www.techradar.com/news/wearables/google-glass-2-release-date-price-features-1300484
[Accessed September 2015].

[6] Raspberry Pi Foundation, 2015. *What is a Raspberry Pi?*. [Online]
Available at: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/
[Accessed September 2015].

[7] Oracle, 2015. *JAX-WS Reference Implementation – Project Kenai.* [Online]
Available at: https://jax-ws.java.net/
[Accessed September 2015]

[8] Oracle, 2015. *Glassfish Server.* [Online]
Available at: https://glassfish.java.net/
[Accessed September 2015]

[9] TechTarget, 2015. *What is Swing?.* [Online]
Avilable at: http://whatis.techtarget.com/definition/Swing
[Accessed September 2015]

[10] W3Schools, 2015. *Introduction to WSDL.* [Online]
Available at: http://www.w3schools.com/webservices/ws_wsdl_intro.asp
[Accessed September 2015]

[11] Jamie Gilbertson, 2015. *PiTechSuppRemote apidocs.* [Online]
Available at: http://jagilbertson.GitHub.io/PiTechSuppRemote/apidocs/index.html
[Accessed September 2015]

[12] Raspberry Pi Foundation, 2015. *Raspberry Pi 2 Model B.* [Online]
Available at: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
[Accessed May 2015]

[13] Raspberry Pi Foundation, 2015. *Camera Module.* [Online]
Available at: https://www.raspberrypi.org/products/camera-module/
[Accessed May 2015]

[14] Amazon, 2015. *Raspberry Pi Case for Model B+ - Black.* [Online]
Available at: http://www.amazon.co.uk/Raspberry-Pi-Case-Model-Black/dp/B00LMDE57S
[Accessed September 2015]

[15] Amazon, 2015. *EasyAcc Classic 10000mAh External Battery...* [Online]
Available at: http://www.amazon.co.uk/EasyAcc-10000mAh-Brilliant-Smartphone-Bluetooth/dp/B00H9BEC8E
[Accessed September 2015]

[16] Amazon, 2015. *Microsoft LifeChat LX-3000 Headset* [Online]
Available at: http://www.amazon.co.uk/Microsoft-JUG-00014-LifeChat-LX-3000-Headset/dp/B000JSDOMO
[Accessed September 2015]

[17] Amazon, 2015. *Dynamode WL-700N-RXS 11n WiFi Nano Adapter.* [Online]
Available at: http://www.amazon.co.uk/DYNAMODE-WL-700N-RXS-Wifi-Nano-Adapter/dp/B008FZO1A2
[Accessed September 2015]

[18] Amazon, 2015. *Kingston Technology Micro SDHC 16GB Memory Card.* [Online]
Available at: http://www.amazon.co.uk/Kingston-Technology-Micro-16GB-Memory/dp/B003WIRFD2
[Accessed September 2015]

[19] Raspberry Pi Foundation, 2015. *Welcome to Raspbian.* [Online]
Available at: https://www.raspbian.org/
[Accessed September 2015]

[20] Debian. *Debian – The Universal Operating System.* [Online]
Available at: https://www.debian.org/
[Accessed September 2015]

[21] Ubuntu Wiki, 2015. *Win32DiskImager.* [Online]
Available at: https://wiki.ubuntu.com/Win32DiskImager
[Accessed September 2015]

[22] Amazon, 2015. *Edimax EW-7811UN 150Mbps Wireless Nano USB Adapter*.[Online]
Available at: http://www.amazon.co.uk/Edimax-EW-7811UN-150Mbps-Wireless-Adapter/dp/B003MTTJOY
[Accessed September 2015]

[23] Putty. *Download Putty - a free SSH and telnet client for Windows.* [Online]
Available at: http://www.putty.org/
[Accessed June 2015]

[24] WinSCP. *Introducing WinSCP.* [Online]
Available at: https://winscp.net/eng/docs/introduction
[Accessed June 2015]

[25] Motion. *Motion – Web Home.* [Online]
Available at: http://www.lavrsen.dk/foswiki/bin/view/Motion
[Accessed June 2015]

[26] Debian. *Apt – Debian Wiki.* [Online]
Available at: https://wiki.debian.org/Apt
[Accessed July 2015]

[27] Eduroam, 2015. *What is Eduroam?.* [Online]
Available at: https://www.eduroam.org/
[Accessed September 2015]

[28] Raspberry Pi Foundation, 2015. *Raspbian Repository.* [Online]
Available at: https://www.raspbian.org/RaspbianRepository
[Accessed September 2015]

[29] Unknown, 2015. *RPi-Cam-Web-Interface.* [Online]
Available at: http://elinux.org/RPi-Cam-Web-Interface
[Accessed July 2015]

[30] GitHub, 2015. *GitHub – Where software is built.* [Online]
Available at: https://github.com/home
[Accessed September 2015]

[31] Oracle, 2015. *Welcome to Netbeans* [Online]
Available at: https://netbeans.org/
[Accessed September 2015]

[32] iNet Telecoms, 2015. *What is VoIP.* [Online]
Available at: http://www.voipfone.co.uk/What_Is_VoIP.php
[Accessed September 2015]

[33] Oracle, 2015. *JSIP: JAVA API for SIP Signalling – Project Kenai.* [Online]
Available at: https://jsip.java.net/
[Accessed July 2015]

[34] Oracle, 2015. *DataLine (Java Platform SE 7).* [Online]
Available at:
http://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/DataLine.html
[Accessed July 2015]

[35] Oracle, 2015. *AudioSystem (Java Platform SE 7).* [Online]
Available at:
http://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/AudioSystem.html
[Accessed July 2015]

[36] AlsaProject, 2015. *Advanced Linux Sound Architecture (ALSA) project homepage.* [Online]
Available at: http://www.alsa-project.org/main/index.php/Main_Page
[Accessed July 2015]

[37] Jamie Gilbertson, 2015. *PiTechSuppRemote Repository*[Online]
Available at: https://github.com/jaGilbertson/PiTechSuppRemote
[Accessed June 2015]

[38] Nano. *GNU nano.* [Online]
Available at: http://www.nano-editor.org/
[Accessed September 2015]

[39] Ebay, 2015. *PortaPow USB Power Monitor Multimeter Ammeter Voltmeter.* [Online]
Available at: http://www.ebay.co.uk/itm/PortaPow-USB-Power-Monitor-Multimeter-Ammeter-Voltmeter-Meter-Solar-Charger-/221242144139?hash=item33830f358b
[Accessed July 2015]

[40] Raspberry Pi Foundation, 2015. *Raspberry Pi 2 Model B.* [Online]
Available at: http://swag.raspberrypi.org/products/raspberry-pi-2-model-b?utm_medium=cpc&utm_source=googlepla&variant=3303107905
[Accessed September 2015]

[41] Amazon, 2015. *Deep Bass Foldable HD Headphones.* [Online]
Available at: http://www.amazon.co.uk/Foldable-Headphones-Tangle-iPhone-Smartphone/dp/B00SRJZEIE
[Accessed September 2015]

[42] Amazon, 2015. *Kingston SD4/8GB SDHC Memory C*[Online]
Available at: http://www.amazon.co.uk/Kingston-8GB-Micro-SD-HC/dp/B001CQT0X4
[Accessed September 2015]

[43] Amazon, 2015. *Raspberry Pi Camera Module.* [Online]
Available at: http://www.amazon.co.uk/Raspberry-Pi-100003-Camera-Module/dp/B00E1GGE40
[Accessed September 2015]

[44] Amazon, 2015. *Raspberry Pi Case for Raspberry Pi 2.* [Online]
Available at: http://www.amazon.co.uk/Raspberry-Pi-Case/dp/B00ZS26ZJA
[Accessed September 2015]

[45] Amazon, 2015. *MAK POWER BANK Cylinder 2600mAh.* [Online]
Available at: http://www.amazon.co.uk/MAK-POWER-Cylinder-External-Ultra-Compact/dp/B00YCKSH7C
[Accessed September 2015]

[46] Amazon, 2015. *Vodafone Smart First 6 Smartphone.* [Online]
Available at: http://www.amazon.co.uk/Vodafone-Smart-First-Handset-Smartphone/dp/B00XJRI0EO
[Accessed September 2015]

[47] Amazon, 2015. *TP-LINK TL-MR3020 Portable 3G/4G Wireless N Router.* [Online]
Available at: http://www.amazon.co.uk/TP-LINK-TL-MR3020-Portable-Wireless-Router/dp/B00634PLTW
[Accessed September 2015]

Appendices

Please find included with this report a DVD containing the Netbeans project files, and the raw .java code files organised by package.