ECE 4600 – Group Design Project

Final Report

Wireless Utility Meter Reading

Prepared For: Dr. Udaya Annakage, Ph.D., C.Eng.
Mr. Paul Card

Advisor: Dr. Robert D. McLeod, Ph.D, P. Eng

Submitted by Group 09

Marc Soiferman
Andy Tang

March 8, 2007

Abstract

This report details the development, design and implementation of a remote meter monitoring system using a camera and wireless ZigBee nodes. Currently, remote meter sensing systems exist, however these are not deployed in older homes do to the prohibitive replacement cost of installing a new system. This system was designed to be a low cost prototype solution, capable of performing remote reading. The major hardware components of this design are a Logitech Quickcam webcam and a pair of Integration ZigBee USB dongles. Most of the physical hardware has been emulated in order to reduce the cost of the prototype. The embedded system used to run the camera and a single ZigBee USB dongle has been emulated on a Linux 2.4 kernel using VMWare. The camera system is implemented in Java using an extra package known as the Java Media Framework to interface with the drivers.

The system is a one-click application that will capture a picture of the meter and wirelessly transmit to the ZigBee nodes from up to 10m away. The device with the camera initializes the ZigBee network and waits on standby until a reader connects. Upon a connection, the camera device transmits an image of the meter to the handheld reader. The handheld reader then displays the image on its screen. The system is designed to allow for extensions to be made by students interesting in pursuing a similar project at a later time.

Contributions

For the project to be completed successfully, team work and a properly divided list of tasks were both needed. The tasks for the project were divided between the two members as outlined below.

Tasks	Members Involved
Research and Literature Review	Andy, Marc
Hardware Design	Andy, Marc
Hardware Emulation and Driver Setup	Andy
Linux Install, Setup and Troubleshooting	Andy
Automate Image Capture	Marc
Graphical User Interface for Receiver	Marc
Initial Transmission Protocols	Marc
Interfaces between Java and C++	Marc
ZigBee Communication	Andy
Performance Analysis	Andy, Marc

Acknowledgements

We would like to thank Dr. Robert McLeod for his assistance with the design procedure and pointing us in an initial direction so that the project could be started.

We would also like to thank the ECE Tech Shop staff for their help in ordering the parts required to assemble our project into a working design. We're especially thankful to Mr. Gordon Toole for acting as a liaison between our group and the staff.

Table of Contents

1. Introduction **1**

1.1. Project Introduction **1**

1.2. Motivation **2**

2. Wi-UR System **4**

2.1. System Description **4**

2.1.1. Physical System **4**

2.1.2. System Specifications **6**

2.2. System Design **7**

2.2.1. System Overview **7**

2.2.2. Design Procedure **8**

2.2.2.1. Stage 1 .... **8**

2.2.2.1.1. Hardware **9**

2.2.2.1.2. Software **10**

2.2.2.2. Stage 2 .... **10**

2.2.2.2.1. Hardware **11**

2.2.2.2.2. Network Accessibility **11**

3. Stage 1, a Wired Solution **13**

3.1. Camera **14**

3.1.1. Hardware **14**

3.1.2. Software **15**

3.1.2.1. Java Media Framework **16**

3.1.2.2. Software Implementation **17**

3.1.2.3. Algorithm **20**

3.1.3. Communication **23**

3.1.4. Graphical User Interface **26**

4. Stage 2, a Wireless Solution 30

4.1. ZigBee USB Dongle **31**

4.1.1. Hardware **31**

4.1.2. Zigbee **32**

4.1.3. RFMessenger **32**

4.2. Camera Device **33**

4.2.1. Camera Algorithm **34**

4.2.2. Java to C++ **36**

4.3. Reader Device **37**

4.3.1. Reader Algorithm **37**

4.3.2. C++ to Java **39**

4.4 Tweaking and Limitations **40**

5. Performance Analysis 42

5.1. Camera **42**

5.2. ZigBee Communication **43**

6. Extensions 45

6.1. Communication Link **45**

6.2. Database **46**

6.3. Optical Character Recognition **46**

6.4. Security **46**

7. Conclusions 48

8. References 49

9. Appendices 50

Appendix A. Installer Guide for JMF on Linux 50

Appendix B. Using JMFRegistry on Linux 52

Appendix C. Emulation Guide **54**

Appendix D. Knoppix Installation Guide **58**

Appendix E. Camera Source Code **59**

Appendix F. Communication System Source Code **64**

F–1. Client **64**

F–2. Server **70**

F–3. States **75**

F–4. Message Routines **76**

Appendix G. Graphical User Interface Source Code **78**

Appendix H. Camera Device ZigBee Link Source Code **80**

Appendix I. Vitas **86**

List of Figures

Figure 2-1. Block Diagram of the System **5**

Figure 3-1. JMFRegistry, run in Linux **20**

Figure 3-2. Camera Operational Flowchart **21**

Figure 3-3. Camera Initialization Routine **22**

Figure 3-4. Communication Diagram for Transfer Protocol **24**

Figure 3-5. Initial GUI for the Wi-UR System **27**

Figure 3-6. Wi-UR System GUI following an Image Request **28**

Figure 4-1. Initialization Routine **34**

Figure 4-2. ZigBee File Transfer, Camera Side **36**

Figure 4-3. PAN Selection **38**

Figure 4-4. ZigBee File Transfer, Reader Side **39**

Figure 4-5. Revised Transmission Algorithm **41**

Figure B-1. JMFRegistry, Initial GUI **52**

Figure B-2. JMFRegistry **53**

Figure C-1. Virtual Machine Configuration **54**

Figure C-2. Select Virtual Machine Format **55**

Figure C-3. Select a Guest Operating System **55**

Figure C-4. Choose a Virtual Machine Name **56**

Figure C-5. Select I/O Adapter Type **57**

List of Tables

Table 2-1. Laptop Receiver Specifications **4**

Table 2-2. Wireless Utility Reader Specifications **6**

Table 2-3. Camera System Hardware Specifications **9**

Table 2-4. ZigBee Device Requirements **11**

Table 3-1. Camera Technical Specifications **14**

Table 3-2. JMF Objects for Multimedia Interaction **18**

Table 4-1. Technical Specifications of the ZigBee USB Dongle **31**

Table 4-2. Commands Executed for Camera Device Initialization **35**

Table 5-1. Performance of the Camera System **42**

Table 5-2. Performance of the ZigBee Communication Link **44**

Nomenclature

List of Acronyms

AES	Advanced Encryption Standard
FFD	Full Function Device
GUI	Graphical User Interface
IP	Internet Protocol
JMF	Java Media Framework
MAC	Media Access Control
OCR	Optical Character Recognition
PAN	Personal Area Network
PDA	Personal Digital Assistant
RFD	Reduced Function Device
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
WDM	Windows Driver Model
Wi-UR	Wireless Utility Reader

1 – Introduction

1.1 – Project Introduction

The design problem posed by the Wireless Utility Meter Reader (Wi-UR) system consists of automating the process of capturing an image and transmitting it to a remote handheld device located out of line of sight of the reader. To design a working solution for the Wi-UR problem, several different tasks had to be performed. Research had to be conducted on key topics, including the ZigBee protocol and how to interface external devices with different operating systems. As well, implementation of transmission over ZigBee and communication through external devices and software was done to complete the solution.

The Wi-UR project involved the design of the reader system and automation of several different processes. Capturing an image from a webcam connected to a remote machine, as well as setting up and dismantling ZigBee networks was required for the Wi-UR system to be a fully automated complete solution. As well, the problem involves writing software capable of interfacing the hardware components as well as creating communication protocols between devices. The completed prototype design has the potential to be a low power, low cost solution to the problem of requiring wireless methods of reading the legacy utility meters currently installed in households.

On top of the Wi-UR system, several extra features could be added to project. These include a database management program, an off-site image processor to allow number recognition of values without human interaction, and an automatic transmitter that sends data periodically without requiring prompting along with various security options to protect the privacy of the clients and guarantee the accuracy of the measurements. While these extra features were outside the scope and timeline of our project, they could easily be made the objective of a project for future years. These extra features will be discussed and elaborated on in Section 6. As well, the Wi-UR device can serve as a prototype for any periodic on-site remote image capturing system where between requesting images, the solution requires a low power, low maintenance solution.

The Wi-UR system has a straightforward system design that will be discussed further in Section 2.

1.2 – Motivation

The Wi-UR device was designed to provide a solution to an existing problem involving legacy utility meter readers located in existing households. This problem is a complex problem composed of individual issues that must all be dealt with for an acceptable solution to be developed for the existing situation. The problem involves safety and time concerns for both of the involved parties, the worker for the utility company (or the *user*), and the customer (or the *client*) who owns the meter that must be read.

The safety concerns for the worker and the customer are similar in nature. Both are based around the fact that the worker needs to enter into the household of the customer to read the meter. For the worker, this poses safety concerns as they are entering into an unknown environment with an unknown person, a person that could be potentially dangerous. The customer must allow the worker entry into their household, and this poses significant threats; people can pose as workers, the workers can steal from the homeowner, or even threaten the homeowner. As well, the worker can cause property damage for the homeowner, either directly or indirectly. Being granted access to a household also allows a worker access to the internals of the homeowner's personal life, a situation which has a potentially frightening effect for individuals. Combined with the safety concerns present with the current system of reading legacy utility meters, there is also the additional time that is required by the customer.

The problem of time is imposed only on the customer, as they must remain in the household waiting for the worker to arrive. While estimates of the arrival time can be given, unforeseen delays can cause the time frame to be large, on the order of hours, which requires the homeowner to be present at their household during the entire time. The time constraints imposed on the worker are non-problematic. The time that the worker puts in is being compensated by his salary and is not an issue. As well, the time required for the worker is the time to arrive at the house, wait for the homeowner to grant entrance, finding the meter and then acquiring the reading. A problematic situation that could arise for the worker is if the homeowner is not present at the household during the expected hours. The worker would have then wasted the time required to arrive at the house, as well the time required to return at a later date.

Wi-UR can provide solutions to these problems by being an effective remote system that is capable of allowing readings to be taken from outside the household. By allowing this, the safety concerns of having to enter into the house are eliminated, as well as the concerns of the homeowner about granting a stranger access to their house. There is no longer a need for the customer to be at home during the reading, since the worker can take a reading from outside the house. This component eliminates any time issues for the customer, as well as negating the scenario where the worker must leave without acquiring a reading due to the homeowner being away at the time.

The method in which Wi-UR is employed to solve these problems will be discussed in Section 2, on the Wi-UR System.

2 – Wi-UR System

2.1 – System Description

The Wi-UR system is split into two systems, physical and software, which will form the Wi-UR system. Both systems are subject to design specifications; however the physical system has much stricter constraints.

2.1.1 – Physical System

The Wi-UR system requires two physical components to function. The first component is a free-standing, independent system equipped with the webcam and a ZigBee transceiver. This component is in charge of the section of the project that will interface with the legacy utility meter. The ZigBee transceiver on this component will be in charge of making the transmission from this device to the other half of the system; this is a fundamental concept to allow for remote reading of the legacy utility meters. The system being used for this is a personal computer system running an emulated Linux environment to simulate an embedded processor. Design decisions will be discussed further in Section 3.

The second physical component of the Wi-UR system is the receiver device. The remote system will be required to automatically take a picture and then forward it to the receiver. The receiver software is implemented on a laptop computer running Windows XP. The laptop computer being used has the specifications described in Table 2-1, which are well above the required specifications for a system to act as a receiving device for the Wi-UR system. Once again, design decisions will be discussed in Section 3. The components listed are fairly straight forward; however these do not represent at all the minimum standards for successful operation of the system.

Table 2-1. Laptop Receiver Specifications

Component	Specifications
Processor	2.0 GHz
RAM	1 GB
Hard Disk Space	107 GB
USB Ports	2

A block diagram for the hardware system is shown in Figure 2-1. As is shown in this diagram, the system is a fairly linear system that accepts user input at the interface, transmits this signal throughout the system to the camera, which then captures information on the environment and returns the information through the same path back to the user interface for interpretation by the user.

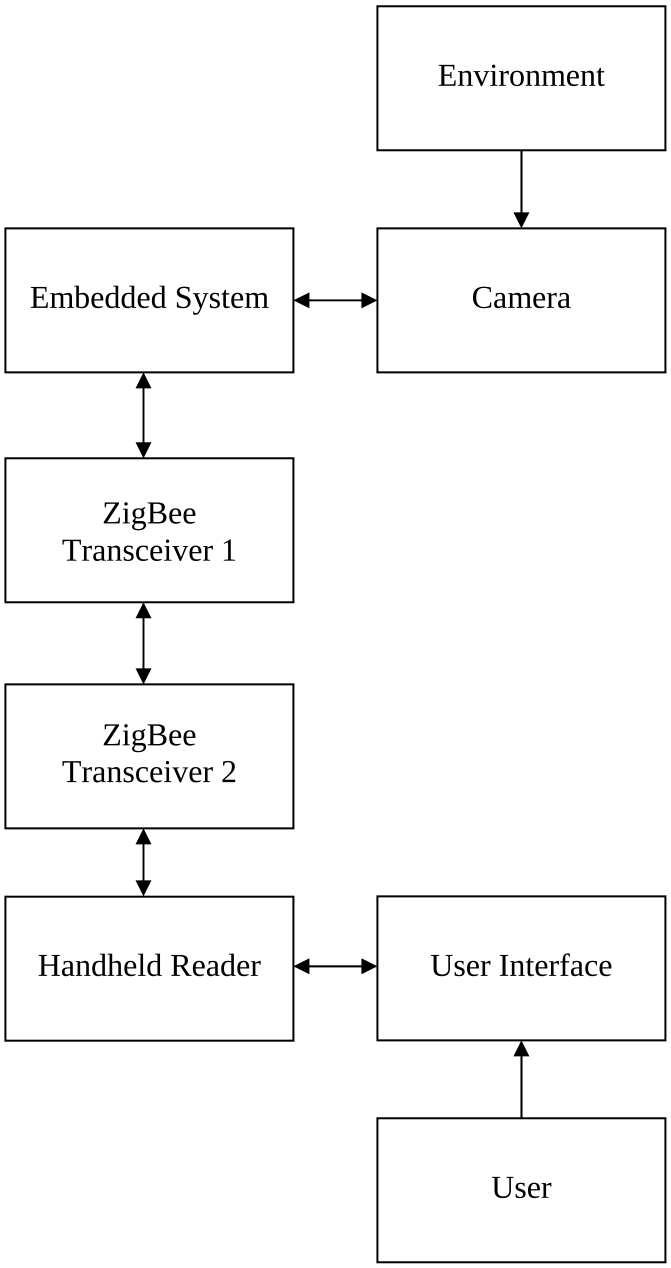


Figure 2-1. Block Diagram of the System

The nodes shown in Figure 2-1 are direct representations of the different components of the Wi-UR system. Each component only has access to the component directly before and after it in the chain. The components are labeled according to their intended implementation, even though this is not how the system was implemented. This is done because even though the system is not implemented as it was initially proposed, the existing components were chosen to emulate the actual components. If the design was advanced out of the prototype stage, then the proposed components could go in place of the emulated ones in the final design. Further examination of Figure 2-1 also shows where the communication link is. This link can easily be replaced by any form of communication link. A different implementation of a communication link, involving Ethernet is explored in Section 6, on extensions to the design.

2.1.2 – System Specifications

The final specifications of the completed system are summarized in Table 2-2. These are compared to the proposed values in the same table.

Table 2-2. Wireless Utility Reader Specifications

Feature	Final Value	Proposed Value
Uptime	5.79×10^{-4}%	<10%
Time Delay	10s - 15s	<30s
Output Power	1mW [1]	<50mW
Image Resolution	320x240 pixels	640x480 pixels
Range	30m [1]	>20m

In Table 2-2, there are numerous parameters. These can be defined as follows; the uptime is the percentage of time that the device is active and consuming normal operating power. The time delay is the amount of time between when a request for a reading is made and the image is received at the device. The output power is defined as the power required by the device to transmit the image to the reader. The image resolution is the number of pixels in the image. The range is how far the Wi-UR device can transmit messages. The uptime is difficult to measure for this application, as it is dependant on the frequency of picture capturing. However, using an industry standard of one reading per month, the following calculations were conducted to estimate the uptime.

$$30\,\frac{\text{days}}{\text{hour}} * 24\,\frac{\text{hours}}{\text{day}} * 60\,\frac{\text{minutes}}{\text{hour}} * 60\,\frac{\text{seconds}}{\text{minute}} = 2\,592\,000\,\text{seconds per month}$$

Using a transmission time of 15 seconds, this means that the device is active for 15 seconds out of every 2 592 000, or an uptime of $5.79\text{x}10^{-4}$%.

As summarized in Table 2-2, the final specifications match up with the proposed specifications except in one category. First, the Image Resolution in the final design is only 320x240 pixels. While the device supports resolutions of 640x480, the quality of the image and the readability of the numbers on the meter are higher when the image is captured at 320x240. After experimentation, it was discovered that this resolution was more than sufficient for the requirements, and since it helped with faster transfers it was decided that 640x480 was an unnecessarily high specification.

2.2 – System Design

The system was split into two devices due to the very specific roles each device had to play. This allowed for the design of the system to be more systematic and less tightly coupled between the two devices.

2.2.1 – System Overview

As mentioned earlier, the Wi-UR system is composed of two major components, a webcam and a ZigBee transceiver network. These two components are split into two different devices. The webcam is installed on site in the house that possesses the meter; this device will be referred to as the *camera* for the remainder of the report. A ZigBee transceiver is placed on both the device containing the webcam, and the device used by the utility worker. The device used by the utility worker, called the *reader*, is a handheld portable reader that is capable of communicating with the different camera devices.

The proposed system design was to have a low-power embedded system that would run a Linux-based operating system and would be able to interface with the camera and ZigBee transceiver. This system would be installed at the meter and would be a low power solution to providing a permanent system that could be placed on site for practical purposes. However, due to budgetary issues this was not possible and the project was instead changed into a proof of concept design. As a result of this, the embedded system was chosen to be converted to a personal computer system running a Linux kernel that is similar to the one that would have been installed on the initial embedded system. This allows the development and devices to be almost identical, while allowing a cheaper alternative of a personal computer system to provide a more cost effective prototype design solution.

The second component of the proposed system was to have a handheld reader that the worker would be able to walk from house to house and use in gathering readings. Due to additional budgetary concerns, this was decided not to be a practical approach for a prototype design. As a result, the choice of a proof of concept was further enforced by choosing an alternate solution. Rather than using a handheld device such as a Personal Digital Assistant (PDA), a laptop computer is being used as the separate system. This allows for development on multiple platforms, which is a design goal for the system.

2.2.2 – Design Procedure

Due to the clear separation of the system into two individual components, the design process for the Wi-UR system was split into its two pieces. Both of these components were able to be treated independent of each other, and could be developed in parallel with no need interact until the final assembly. This design system resulted in a two-stage approach being implemented.

2.2.2.1 – Stage 1

The first stage of the system was to design the software and hardware support to enable the camera to function independently of human interaction. The goal of this stage was to have a fully automated camera system that could transmit a file over a network to another program which would reassemble the file and display it on the screen. The requirements of full automation of this system are as follows. The camera must be able to start up, acquire an image, write this image to a file, and then go back to standby without a user needing to interfere. The startup signal for the camera must be transmitted through software, as well as the termination signal.

For Stage 1, a network was implemented as a concept to allow simpler substitution of the ZigBee network once it was completed. This network could run over any protocol, and initial design discussions selected USB and Ethernet as the two most straight-forward options. Investigation into the USB Protocol [2] revealed that USB drivers are not simple to create and thus Ethernet was chosen as the final design solution. This simple network allowed for the main focus of the stage to be the camera, and not the communication between the components. As well, if the entire system was to be extended to be fully automated without requiring any human interaction, a switch from ZigBee to Ethernet would be required. This will be further discussed in Section 6, on possible extensions to the project.

2.2.2.1.1 – Hardware

The first design step taken was to define the requirements for the hardware emulation system that would allow it to interface with the camera. These specifications that were developed are summarized in Table 2-3. These hardware specifications are considered to be minimal, and higher specifications were used.

Table 2-3. Camera System Hardware Specifications

Part	Additional Comments
USB Port	2
Linux Based Operating System Support	Kernel 2.4
RAM	32MB
Hard-disk space	30MB-50MB

Examining Table 2-3 provides insights into several key design concepts towards construction of the Wi-UR system. Two USB ports are required due to the need to interface a camera and a ZigBee transceiver with the system. Support for a Linux kernel 2.4 operating system is required to correctly emulate an embedded system. This kernel is the one that would have ran on the original embedded system, had it been used in the project. A major design decision reached was to still emulate the system exactly as if the embedded system was being used, while not using one so that the concept would be able to be immediately transferred to the embedded system if one became available. The RAM and Hard-disk requirements are in place to allow for the functioning of drivers, run time environments and developed software. In the final system, these design requirements were exceeded for the purposes of RAM and hard-disk space, allowing for more liberal initial designs to be completed before optimization had to be considered as a factor.

As the system design for the Wi-UR system requires a camera, two alternatives were considered, a webcam and a digital camera. The webcam was chosen to be the superior choice for numerous reasons. It is a cheaper solution than a digital camera, as well as providing smaller resolutions and thus smaller image files. For the applications that Wi-UR is required, having a large camera resolution on the order of megapixels would be detrimental; it would cause the file transfer to be slower as well as requiring extra hard-disk and memory space to store the image. As well, a USB webcam can be interfaced and automated easier than a digital camera, and this provided extra incentive to use one. Implementation details will be discussed further in Section 3.

2.2.2.1.2 – Software

The design specifications for the software for Stage 1 are based on enabling the functionality of the camera as required for the system. The software must be able to start up the camera, take a picture and then put the camera back on standby, to minimize power consumption. The software was also to allow for transmission of a file across a network while remaining transparent to the network as much as possible. Development for the camera software had to be completed so that it could be run in Linux. The other half of the system, the part that is composed of the human interface, was to be developed for a Windows based platform, as discussed earlier. This system was to be designed to have a usable, simple, straight-forward graphical user interface that would acquire an image at a command from the user, as well as displaying the image for the user to see. Implementation details will again be discussed later in Section 3.

2.2.2.2 – Stage 2

The second stage of the project is to design and implement the ZigBee communication link between the two devices. Once again, this requires development for multiple platforms as well as potentially interfacing multiple programs together in order to successfully interface the ZigBee communication link with the process of automating the image acquisition. The difference in platforms arises from the differences in the platforms being implemented on the two different ends of the system. However, due to the nature of ZigBee, it naturally abstracts the communication link layer from the underlying operating system; this results in less emphasis on the message format and content. The ZigBee requirements are described as follows:

- The reader must be able to automatically detect and connect to an existing ZigBee network.
- The camera requires the ability to start a network as well as accept incoming connections.
- Both devices must have the capability of sending messages to the network as well as receiving messages from the network.

The second stage is only concerned with developing the source code to run the ZigBee link; both the camera software as well as the GUI are developed in the first stage. The ZigBee code developed will replace the Ethernet transmission and will need to interface with the code developed in the initial stage. In order to expedite the process of code creation, example applications supplied by Integration, the vendor for the ZigBee-USB dongle, were studied. These were then reused and modified to fit our application. Investigation into the ZigBee documentation and software discovered the various network protocols for detection, connection and transmission. In order to provide extendibility to future projects, the project will combine to have the camera and reader behave as a full function device (FFD).

2.2.2.2.1 – Hardware

Working on multiple PC platforms requires a suitable interface for the ZigBee connection. For this design, USB is the obvious choice as it is ubiquitous for computer interfaces. The important of this decision is that device drivers were needed for the two different platforms used; Windows and Linux. Using drivers abstracts the control of the ZigBee device so it is easier to manipulate and use. The ZigBee device requirements are shown in Table 2-4.

Table 2-4. ZigBee Device Requirements

Requirement	Additional Comments
2 Transceivers	One per device; more needed for testing
Drivers	For Windows XP & Linux Kernel 2.4
FFD Capabilities	Required to start and connect to networks
USB Connectivity	Plug & Play capabilities

By inspecting Table 2-2 and Table 2-4, the constraints imposed on the system limit the number of devices that would be useful for the project. The devices that were decided on come included in a development kit from Integration (IA DAUB-DK1 2400). This kit contains a CD with several reference programs, drivers and various documents. These documents concern the reference programs, device capabilities and the ZigBee protocol. The main part of the kit is the two ZigBee USB dongles (IA OEM-DAUB1 2400) that were chosen to implement the design.

2.2.2.2.2 – Network Accessibility

The design of Stage 2 requires that software be capable of finding, creating, joining and sending data across a ZigBee network. This requires that the role of each device, the camera and the reader, must be clearly defined. The camera must be able to start networks, accept incoming connections, and send data to a connected party. The reader must be able to actively search for, connect to, and send data to the camera. The creation of network can occur with either device due to the FFD capabilities of the dongle; however it is more energy efficient to have the camera device start the network. In addition, this leaves open the option of ad-hoc networking; this will be discussed further in Section 6. The device code created in here must also be able to communicate with their Java counterparts. This stage merely replaces the Ethernet connection code with ZigBee code.

3 – Stage 1, a Wired Solution

The wired solution required two stages for the total design to be accomplished. First, the separate pieces of the hardware had to be able to communicate with each other. Secondly, the destination computer was required to be able to issue control commands to the hardware to start the processes of picture taking and transmission. The communication interface selected to be run with the wired interface solution was the Ethernet protocol [3]; this allows for fast and simple communication between the two devices. The wired solution is a mid-point in the design; upon completion of this stage, a functional wired prototype was complete and fully operational. This might sound silly, as the project requires a wireless solution. However, despite the fact that at this stage the system is connected via a wired interface, due to the identical Ethernet development protocols for wired and wireless communication, the wired connection could easily be replaced by a set of wireless Ethernet transceivers. This would allow the design to be converted into a remote utility meter reader that is fully functional.

The wired solution is composed of three components, the camera, the graphical user interface and the communication between the two devices for Wi-UR. The camera is further composed of a hardware and a software component. The camera is programmed in Java using a supplemental package known as the Java Media Framework. This allowed for interfacing of the device to the system without many major issues. At this stage, the communication protocol used between the two devices is the User Datagram Protocol, or UDP. The GUI for the reader side of the system is also programmed in Java. Java GUIs are simple to construct due to them using frameworks instead of libraries for construction.

3.1 – Camera

The camera is the key component of Stage 1, and the major contribution the stage contributes towards the final design. The component involving the camera requires automating the image capture. For the scope of the project, the definition of full automation is simply that it can operate after the initial power up and starting of reception software without further user intervention. This means that the software and the machine do not need to initialize themselves as both can be started by a user. The camera software is coded to interface with Windows drivers supplied by the vendor as well as Linux drivers that were downloaded online.

3.1.1 – Hardware

The camera selected to be used for the project is the Logitech QuickCam for Notebooks Deluxe. The specifications of the camera are listed in Table 3-1.

Table 3-1. Camera Technical Specifications

Technical Specifications
Image Sensor: Colour VGA CMOS
Colour Depth: 24-Bit True Colour
Video Capture: Maximum Resolution of 640x480
Manual Focus
USB 1.1 and 2.0 Compatible

This camera was found to meet the requirements of the Wi-UR system.

A potential problem was discovered when it was found out that Logitech only advertised compatibility with Windows based operating systems. However, additional research found a driver that was reported to be able to work with Linux. This driver is called SPCA5xx and has versions for both Linux kernels above 2.6 and below 2.6. Since the kernel being used is kernel 2.4, SPCA5xx version 0.60.00 was used. While this driver was said to work, there was some problems actually getting it functioning properly.

The camera was initially tested with the manufacturer supplied drivers for Windows. This was done to permit easy testing of the camera qualities and the different types of operations and file formats it would support. After rigorous testing, the camera was found to be able to perform according to the needs for image acquisition and transfer. This testing provided several key informational components about the differences between the specifications provided by Logitech and the actual operation of the camera. A surprise was that the images became distorted if a capture resolution of 640x480 was used. Standard logic would imply that the quality of the image should increase if the resolution is increased; however the output images appeared to be simply rescaled versions of their lower resolution counterparts. This produced a pixelation effect that made the images more difficult to view. Since the clarity of the acquired image is a more important aspect of the design than the size of the picture, the lower resolution of 320x240 appeared to be a better choice for the application. There is a function in the camera that allows for a toggle of resolutions between 320x240 and 640x480, however it is currently locked and only used internally. The resolution of 320x240 is an adequate resolution for reading the digits from an image of a utility meter, and since the images are of high quality this was chosen to be the optimal resolution.

3.1.2 – Software

The software component of the camera is programmed in Java. There are several reasons why Java was chosen for the development of the software. Java is promoted as a cross-platform solution to programming problems. This allows for development of an initial prototype in a Windows environment. Since Windows environments are more familiar and easier to deal with than a Linux based system, this was a large consideration for using Java. As well, when conducting research on the subject of interfacing software with webcam drivers, numerous sources were found to be using a Java package known as the Java Media Framework. This will be discussed in further detail in the next subsection. Research indicated that the Java Media Framework would be easy to use and apply to the problem, and so it was chosen for the project.

The software for controlling the camera is isolated into a Java class on its own. This class has multiple useful functions, including an ability to modify the capture resolution to any value, the ability to test if the camera is operational, the ability to capture a frame, initialize the camera if it’s not operational, take pictures, write the picture to a file and disable the camera. The camera class can be initialized and called in any java program, including a stand-alone script that captures an image embedded into the class itself.

Because Java was used, the software was initially developed in Windows XP. The result of this meant that a closer relationship to the vendor supplied drivers was established and kept, enabling a much quicker guarantee that any problems with using the camera was related to bugs in the developed software and not bugs in the installation or use of the drivers. This helped isolate the different errors that could occur by separating the different implementations to prevent compounding of software and installation bugs. After development on the Windows system, a port to Linux was simple, with the most time consuming part being the installation of the Java Media Framework.

3.1.2.1 – Java Media Framework

The Java Media Framework (referred to as JMF) is an additional library provided for free by Sun Microsystems to help Java interface with media devices. JMF is an extension on the standard Java Platform and can be used for effective cross-platform multimedia development. While code written to be compatible with JMF can be run on any platform, the actual framework does not need to be platform independent. Sun Microsystems has made available different package versions that are tailored specifically for different operating systems, including Windows, Linux and Solaris SPARC. The package is offered in these different versions to give the Java Virtual Machine more direct access to the native multimedia drivers and video/audio devices that are native to the different platforms. Despite this difference however, written code functions the same with the different versions of the Java Media Framework.

The Java Media Framework functions by abstracting devices as their respective data steams. These streams can be a video stream or an audio stream. Java then talks to the drivers for the device and allows these streams to be manipulated as data streams in user created programs. The simplicity at which this can be programmed eliminates the need for finding drivers which allow importing into C or C++ programs. This provides a large programming advantage, as knowledge and research of the underlying structure of individual devices and drivers are not required to be known. Another large advantage of this approach provided by Java is that it allows for programs to be written independent of the drivers or devices that are currently being used in the system. The importance of this aspect is that drivers can be updated or modified without requiring additional software development time. As well, a much more surprising consequence of this approach is that the device can also be switched out requiring only minimal changes to the software, rather than additional large scale software development.

In order to properly use the Java Media Framework, the package must be downloaded from the Sun Microsystems website. Once this is done, it must be installed properly on the machine of interest. In order to install JMF, it requires 32MB of RAM and less than 10MB of hard disk space. If these requirements are met, the software installs very easily on a Windows based operating system. However, in order to properly install JMF on a Linux machine, a lot more effort had to be made. Because there is no self-extracting installer executable supplied for the Linux platform, the installation must be done manually. Research from an installer guide provided the following outline on how to install JMF on a Linux operating system. An installation guide will be supplied in Appendix A.

Before the Java Media Framework can be used in a program, the device in question must first be registered to be used with JMF. This is done through a supplied program called JMFRegistry. This is a program provided by Sun Microsystems with the JMF download, and aids in management of the JMF registry file. It is written in Java, runs with the Java Virtual Machine and works through a simple graphical user interface. The program is able to detect the devices that are currently installed on the machine and allows the user to verify different information about them, including the resolutions supported, the type of device and the name of the device that must be used to reference it inside developed software. Sun Microsystems also supplies a program called JMFInit that will initialize the Java Media Framework for operation on a computer. While using these two programs is simple in a Windows environment, when used in Linux a problem arose. Since only the user “root” can write to system files or edit anything related to system operation, such as the JMF registry, attempts at running the registry editing program proved difficult. The root was only able to be logged in through the command line, not the graphical user interface, which posed significant problems with running the graphical interface driven editor. When attempting to run the registry, the command line produced errors related to the inability for the interface to find a proper graphical display device. This was fixed by reinstalling the Java Media Framework. Once the registry editor was run while logged in as the root, it could be updated and the Java Media Framework worked perfectly, allowing proper interaction with the connected multimedia devices.

3.1.2.2 – Software Implementation

The software for the camera, written in Java for the Java Media Framework, relies heavily on library calls and objects to provide the services required for the system to function. The objects required are summarized in Table 3-2. Examining Table 3-2 has an indication of the complexity of learning the required procedure towards constructing operational code using JMF. However, once this initial learning curve is conquered, the programs are simple to generate and work very well.

Table 3-2 – JMF Objects for Multimedia Interaction [4]

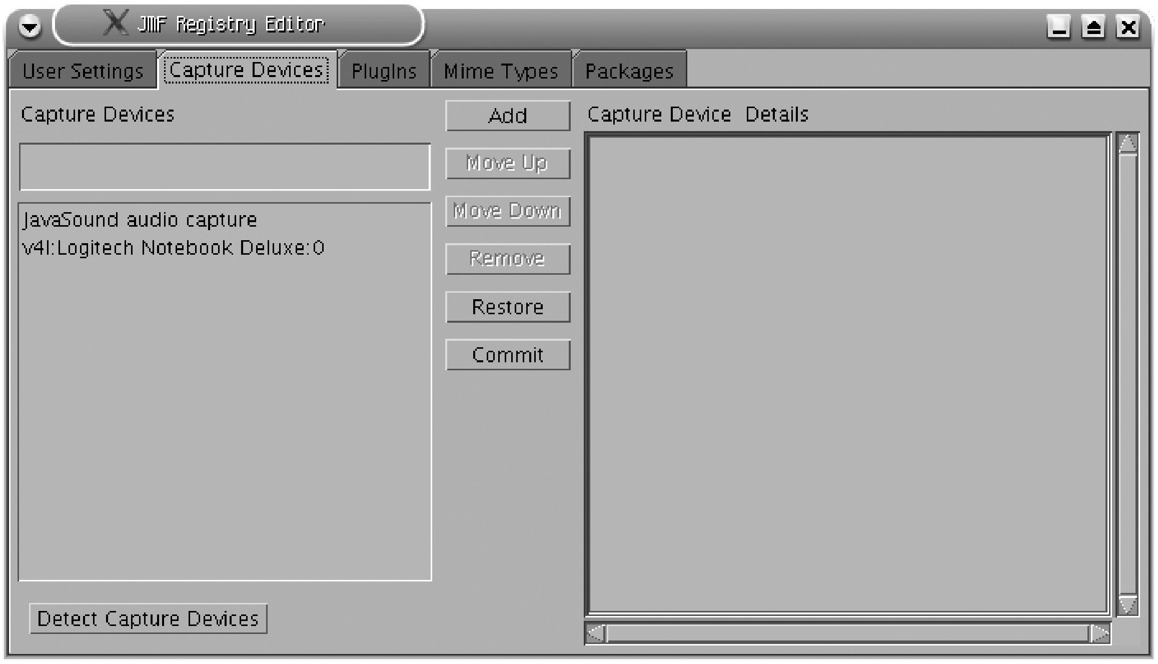
Object	Purpose
Player	Controls and renders the multimedia device output data.
FrameGrabbingControl	This allows access to the ability to grab a single image frame from a video feed.
MediaLocator	This finds the multimedia device based on supplied information like the device name.
Manager	An access point that enables abstracting devices by obtaining system resources to start players and data sources.
Controller	Stores state descriptions that can be used to check on the current state of the multimedia device in use.
CaptureDeviceManager	A structure-like object that keeps track of all available capture devices.
CaptureDeviceInfo	A structure-like object that keeps track of the information of the current capture device being used.
Buffer	Takes data from the video stream and stores it for access
VideoFormat	Allows for checking of the format of the stream currently stored in a Buffer.
BufferToImage	Takes a VideoFormat and initializes a converter that allows a Buffer object to be converted into an Image object.
Image	Takes the data stored in a Buffer object and allows it to be treated as an image.
BufferedImage	Creates a buffer for the data that allows an image to be output to a file
Graphics2D	Scales an image and formats it for output to a file

In addition to these objects, the methods performed on them are equally important. The structure of the calls and their relationships to the objects listed in Table 3-2 will be explored further in the section dedicated to the algorithms employed in the camera system.

The software is simplistic in its use and tailored specifically for this problem. It relies on there only ever being one server to make a request to the camera; this is a safe assumption for the Wi-UR system because sending multiple workers out to read the same meter is a waste of both time and resources. Other assumptions were made to simplify the design process for the software and the algorithm. One such assumption, that the camera only needs to take one picture each time it is started, allows for conversation of power. This is accomplished by allowing the camera to turn itself off as soon as it is done taking a picture. This makes the program run slower, since a start-up of the camera each time increases execution time. However, this increase in execution time is worth the decreased power consumption required by the Wi-UR device. A second assumption is that the only goal of the camera is to take a picture, and thus a timeout is not acceptable and it is given as long as it needs to get the image. Additional development of the software after the initial completion led to several key functions being optimized, the most important of this being the function that checks if the camera is initialized yet. As well, memory usage was optimized by removing additional objects that were not required for the algorithm but aided in debugging and testing.

In the software, the camera needs to be referred to by name. For Windows XP, the name of the camera is "vfw:Microsoft WDM Image Capture (Win32):0", and for Linux, the name is "v4l:Logitech Notebook Deluxe:0". These names are used with the MediaLocator object from the JMF library; using them allows for the devices to be located by name of the driver they have installed. A corollary to this is that if you switch the device or the driver that is used for the device, the only change required in the software is changing the device name that you are looking for. These device names were found by using the JMFRegistry program provided by Sun Microsystems.

Figure 3-1 shows a screenshot of the JMFRegistry program being executed. By observing Figure 3-1, in the list of Capture Devices, "v4l:Logitech Notebook Deluxe:0" is clearly visible. This shows that the camera is properly installed in the operating system, and that the Java Media Framework is properly configured to allow access to it through developed software. If the camera is not present in this list, then it cannot be used in a program until it is placed on the list. A short guide on using JMFRegistry will be provided in Appendix B.



JavaSound audio capture
v4l:Logitech Notebook Deluxe:0

Figure 3-1 – JMFRegistry, run in Linux

3.1.2.3 – Algorithm

The algorithm for the camera is designed to be both robust for errors and provide reliable services. The camera system is represented as a finite state machine, and the algorithm defines the proper transitions between the states. A flowchart of the algorithm for an image capture is shown in Figure 3-2. As the flowchart shows, the program starts with the camera closed and non-operational. The algorithm commences by the initialization of the camera, and then following the completion of this step, the camera is started. After this is done, an attempt is made to take a picture using the camera. This is tried repeatedly until a picture is returned by the camera; the reason for this is that the software can run faster than the device interface to the camera.

Since the interface is slower than the flow of the software, program execution can easily assume that the camera is not returning any data. Meanwhile, the camera is transferring data through the interface at a slower pace, and this data arrives too late. By making the software pause until the camera is responsive, it eliminates this latency. By repeatedly reading from the camera until the camera data is written into a buffer, it guarantees that a picture will be available for the next step. After the image is taken, it is written to a file and then the algorithm terminates by shutting down the camera.

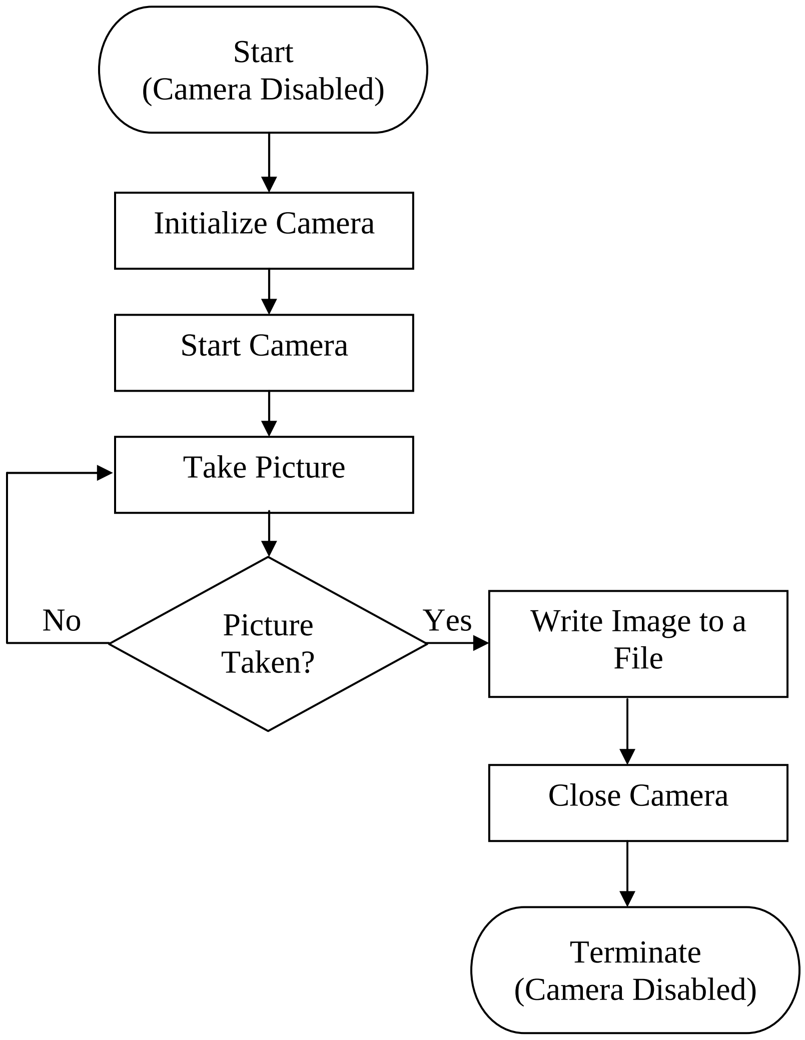


Figure 3-2. Camera Operational Flowchart

The initialization step can be further expanded, as this step involves much more detail than any of the other steps. A flowchart for initializing the camera using JMF can be found in Figure 3-3. Examining this figure and comparing it to Figure 3-2 shows that the initialization routine is more complicated than the entire rest of the algorithm required to capture an image and write it to a file. The two loops present in the flowchart indicate positions where the camera could be non-responsive due to slow recognition by the system, or slow starting of the camera's functionality after it has been found by the system. The camera cannot be considered to be fully initialized until the video streams that it captures are usable for the acquisition of images from the environment.

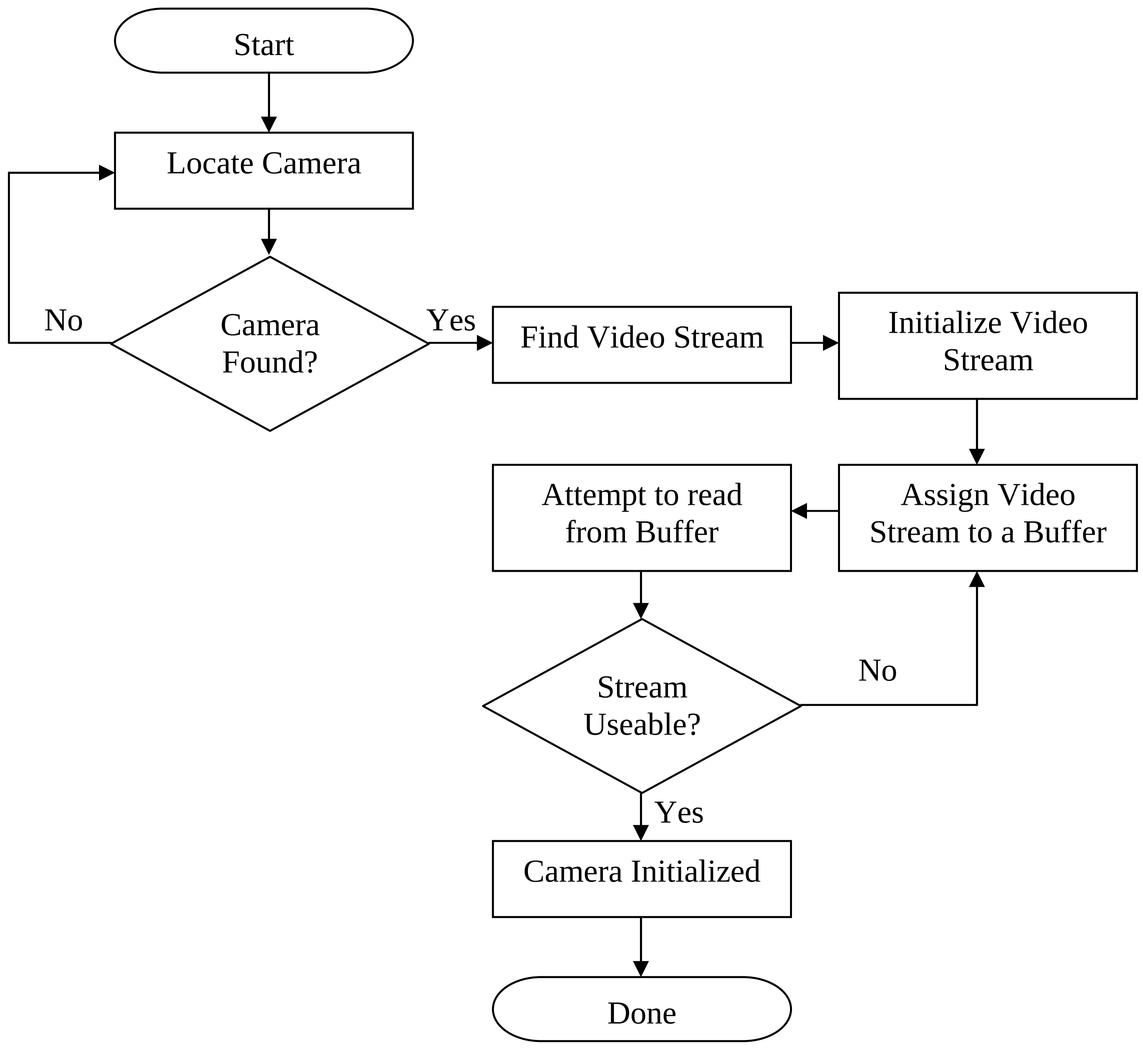


Figure 3-3. Camera Initialization Routine

All stages in the algorithm require the use of library functions that are applied to the objects listed earlier in Table 3-2. These functions are often tightly coupled between different objects, as they require these objects as parameters, or return new objects from the list to be used in the next step as a parameter. This creates a flow of objects that can be observed to map the progression through the algorithm.

Source code for the Camera System can be found in Appendix E.

3.1.3 – Communication

The communication layer for Stage 1 of the Wi-UR device is done over Ethernet. The control protocol that was chosen to be implemented over Ethernet is the User Datagram Protocol (UDP). This was chosen over the more standard Transmission Control Protocol (TCP) for numerous reasons. Using UDP allows for more specification of the different parts of the protocol, along with allowing for potentially higher bandwidth transfers. This is because UDP is a less defined protocol when compared to TCP; the major differences between TCP and UDP are that TCP offers the following features that UDP does not:

- Ordered Data Transfer
- Error-free Data Transfer through retransmission of lost packets
- Discarding of duplicate packets
- Automatic Congestion Control

These parameters give some unique functionality to TCP that is not present in most transmission protocols available to be used. However, this additional control built into TCP also carries with it some increased restrictions. The speed of TCP is reduced due to guaranteed packet delivery; this is because it requires acknowledgements to be transmitted back to the sender. Due to the relatively small number of packets involved in the Wi-UR system, transmission over UDP was found to be faster, easier to implement and control as well as reliable enough for the purposes.

While Ethernet communication is not fundamentally important to the Wi-UR system for the scope of this project, having a transmission that functions over Ethernet is useful for a possible extension to the project, discussed further in Section 6. Having a working communication in Stage 1 allowed for it to be a completely functional prototype of the Wi-UR system upon completion. This goes with the design goals submitted with the project proposal that listed functional sub-systems as an important design decision. Having the Ethernet connection functional before the ZigBee connected was complete allowed for a fallback of using wireless Ethernet as the transmission in the design rather than having an incomplete design due to issues with ZigBee. The transmission protocol used to moderate the transfer over Ethernet is given in Figure 3-4. This protocol is needed due to the usage of UDP over TCP, which would handle most of the protocol by itself.

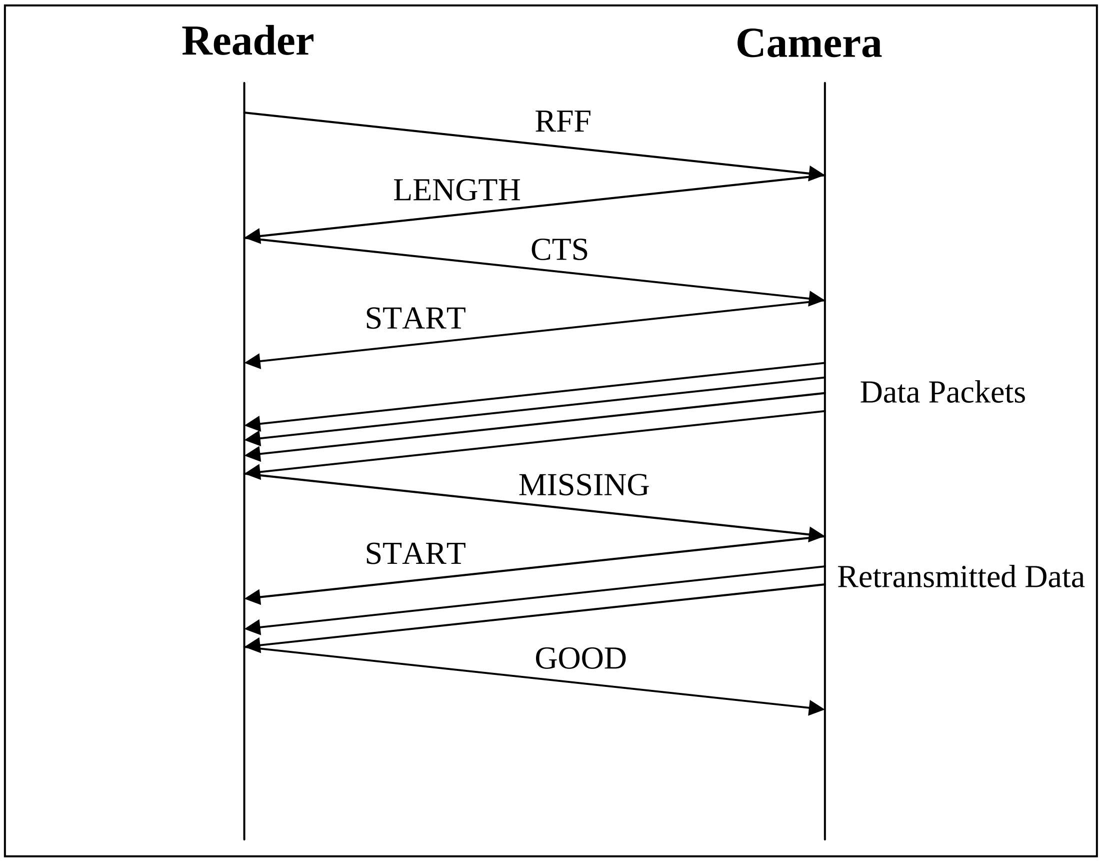


Figure 3-4. Communication Diagram for Transfer Protocol

Figure 3-4 shows the communication diagram that occurs between the two devices. The messages are passed in the shown sequence and the two devices are transitioned through internal states dependant on the message that is sent. The message names and content are described as follows:

- **RFF – Request For File**. This message is sent with a file name appended at the end of it. This lets the server know which file the user wants to get. This tag is a communication starter as well as letting the camera know what file to save the image as. Sending this message will cause the camera system to capture an image.

- **LENGTH – Length of file**. This message is sent by the camera with the length of the captured image file appended at the end of it. This lets the reader know what size of a data array needs to be generated.

- **CTS – Clear to Send**. The reader sends this message when it has set up its data array, and is ready to receive the file.

- **START – Start transmission**. The camera sends this after it receives the CTS to allow the client to know that data is going to start being transferred.

- **Data Packets** – The data packets are sent with a Sequence number appended to the start. This lets the reader know the order to reassemble them in.

- **MISSING – Missing packets**. This message is sent after the data transfer is finished if there are one or more missing packets.

- **GOOD – File is sent**. This message lets the camera know that the file has been successfully transferred to the reader.

Sequence numbers are very important in the functionality of the protocol as they guarantee that the receiver will properly interpret the files based on the sequence number attached to the start of every packet.

This communication software can be used to implement a reliable file transfer method for use with an Ethernet implementation of the Wi-UR system. Once again, the importance of having a transfer method that functions over Ethernet is discussed in Section 6.

Source code for the communication system can be found in Appendix F.

3.1.4 – Graphical User Interface

The final component of Stage 1 is the graphical user interface (GUI). The half of the system that is implemented on the hand held reader requires user interaction. While the command line is a way to receive this input, the use of a graphical user interface makes the process much more user friendly. The goal of any user interface is to aid the user in using the tools that are needed to fully explore the functionality of the given program. In order to properly understand what is required in the user interface for the Wi-UR system, the functional requirements of the program must be examined.

The important functional requirements related to how the user sees the program operation are as follows:

- The program must allow a picture to be requested when specified.
- The program must allow a picture to be received from a remote location.
- The program must display the picture for the user to view.

By examining these requirements, the second requirement, that the picture must be able to be received, should not be displayed on the interface. The user should not be able to have access to the inner workings of the communication, as this would open a hole in the design that would allow potentially unrestricted access to the system, allowing for users with malicious intent to exploit the system.

The interface must be simple and straight-forward to use, so as to prevent problems from occurring rather than attempting to solve the problems that occur. The major functionality that the user needs is the ability to let the system know that a picture is required; this triggers the communication and operation of the remainder of the system. To allow for this functionality, a single button can be used. This button, appropriately labeled “Acquire Image”, is the starting block for any interface that will be used with the Wi-UR system.

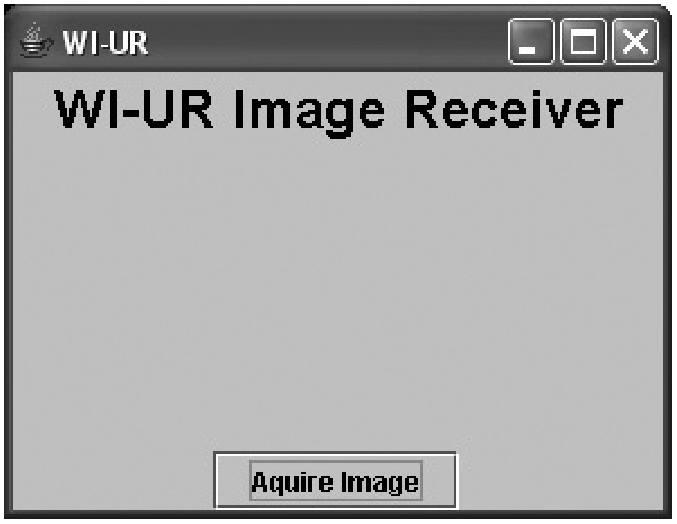


Figure 3-5. Initial GUI for the Wi-UR System

Figure 3-5 shows a screenshot of what the interface will look like when it is initially started by the user. Upon inspection of this screenshot, it is clear that the "Acquire Image" button dominates the interface, and the lack of any additional forms of interaction with the user streamlines the usage of the software. This simplicity was selected over a potentially more complex interface because for such a dedicated system, a complex interface would likely cause too much confusion over what the purpose of each control element is. The single button design approach pictured in Figure 3-5 has proven to be both effective and efficient.

Once this button is pressed, the program is allowed to continue along according to the algorithms discussed earlier. The interface detects that the user has pressed the button and makes an appropriate system call to the communication program in order to acquire the image. Figure 3-6 shows a screenshot of what the interface looks like after the "Acquire Image" button is pressed. Examining Figure 3-6, the two clear features present are a large image being displayed in the middle of the screen, and the "Acquire Image" button still being present. The purpose of displaying the image in the interface is that it allows for the user to view the image without requiring the need for an external, 3rd party graphics program to be used.

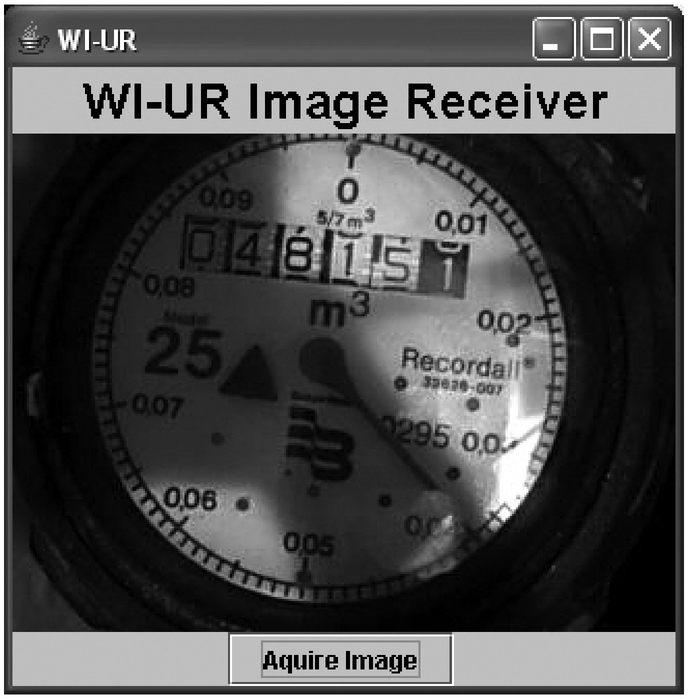


Figure 3-6. Wi-UR System GUI following an Image Request.

Displaying the image in the interface allows for quicker access to the picture, as well as easy verification of the current image to determine if it is of suitable quality for the application. This second feature of Figure 3-6 is how the "Acquire Image" button is still present. This allows for a request of additional images to be taken after the initial one is received. An example of a useful utilization of this feature is to reduce noise in images acquired from a noisy environment. This is accomplished by allowing for multiple samples to be captured for use in Image Averaging [5], a form of noise reduction.

The interface is constructed from the supplied graphical user interface design framework in Java. This framework is part of the Swing toolkit of the java extensions. Programming an interface in Swing is much simpler than coding an equally complex interface in a language such as C++. This allowed for more time to be spent working on the design, rather than working on the aesthetic side of the project. Using Swing relies on the theory of frameworks. A framework is one school of design for using pre-existing code to assemble new designs. The alternative is the idea of libraries, collections of functions that perform tasks that could be useful in the new concept. The major difference between libraries and frameworks is that in a framework, the developer must only implement the functionality that differs from the base functionality. With a library, there is no concept of a base functionality and everything must be implemented by a developer, even if they do not intend to use it. The differences are especially noticeable when initialization must be done for the design to function properly. A library will require the user to implement the entire initialization routine, while a framework will automatically initialize everything, and unless the user has some custom initialization, they would not have to contribute at all to the process.

The interface itself is composed of three components; a label, a button and an image frame. Out of these three components, the image frame is the most difficult to get functioning properly. The initial attempt at the image frame worked fairly well, however it was quickly discovered that the image would not update when multiple pictures were loaded into the user interface. The solution to this critical problem with the interface revolves around how Java handles reading in files. In order to save execution time, Java caches images that use the same file name when they are read multiple times by the same program during the same execution. This was solved by forcing Java to clear its cache whenever a file was to be read in. This is accomplished through using a supplied toolkit in Java to read the image data instead of using an *Image* object to do the job. The toolkit is used by the following line *Toolkit.getDefaultToolkit().createImage()* and can read in a file without using the pre-existing cached data from a previous read of the file.

Source code for the Graphical User Interface can be found in Appendix G.

4 – Stage 2, a Wireless Solution

The wireless solution was broken down into two components, the reader device and the camera device. These two devices required code to be developed on different platforms – Linux 2.4 for the camera and Windows XP for the reader. The requirement established for the camera device is that it must be able to start up a ZigBee network, accept connections to this network and then enable message transfers. The reader device needed to be able to scan for a ZigBee network, connect to it and then communicate across the network. The wireless solution is the final stage of the system and subsequently, after its completion, the project should be deployable. Despite this, the project is still developed to be a prototype and further development would be required to make it completely deployable. This is stage is an extension on Stage 1, as it takes the design completed in Stage 1 and fills in the remaining piece, a wireless link. The primary objective of the system, having wireless access to a meter at a minimal cost of power and money, was accomplished by completing this stage.

The camera system was programmed in C for use with the Linux GCC compiler, and the code for the reader device was written in C++. C++ was chosen for the reader because it reuses code that was supplied with the development kit. The project reused this code as it demonstrates the capabilities of the ZigBee devices, as well as reducing the amount of development and testing that is required. The sample application used, RFMessenger, is built with a GUI for Windows that was used in conjunction with the GUI from Stage 1.

4.1 – ZigBee USB Dongle

These dongles are the sole focus of Stage 2, and the last step of the final design. The sample applications provided Integration provided valuable insight on how the ZigBee devices operated. This furthered the goal of the project of automating the entire image capture process.

4.1.1 – Hardware

The ZigBee hardware used is Integration's ZigBee USB Dongle, IA OEM-DAUB1 2400. The technical specifications are listed below in Table 4-1.

Table 4-1. Technical Specifications of the ZigBee USB Dongle

Technical Specifications	
Property	**Value**
USB Compliant	1.1 Interface only
Range	30m (typical)
Transmission Active	108 mA
Receiver Sensitivity	-87 dBm
Low Power Modes	Supported
Windows Plug & Play	Supported
FFD Capabilities	Supported

These devices were found to meet the requirements outlined in Table 2-2 and Table 2-4. These ZigBee devices are not truly Plug and play compatible; they still require custom installed drivers to function. However, this should not be a problem since both devices will need to be configured before deployment.

The functionality of the ZigBee USB dongles was tested with the sample applications supplied by the development kit. The useful Windows test applications for the devices include:

- A simple chat program, "RFMessenger" and "RFMessengerConsole".
- A beacon generator, "BeaconGenerator".
- A network scanning program, "ActiveScan".
- A wireless protocol analyzer.

In Linux, the project was based off a two supplied programs. One, "gtkmacgui", which issued simple MAC commands, and another program, "snooper", that listened for incoming data transmissions. These programs, together with their documentation, helped distinguish the differences between startup, connection and transfer sequences, as well as how to implement them.

4.1.2 – ZigBee

ZigBee networks are low power, cost effective wireless transmission networks. They are used mainly for applications involving sensing and monitoring where a wired interface would be costly, bulky or simply impractical. ZigBee networks can be broken down into three different network categories; star, tree and mesh. In this project, the focus is on the tree topology. For future work, however, the network will need to be expanded to a mesh topology to incorporate the large number of devices. This will be discussed later in Section 6. In a ZigBee network, each node is given a unique address. Depending on the type of network, they can be either long or short address. Long addresses are 64 bits while short addresses are 16 bits long [6].

ZigBee networks can be additionally broken down into another category, beaconing and non-beaconing networks. A beacon is a packet that is transmitted periodically by the network coordinator to synchronize the other nodes. Beaconing networks can provide a superframe during each beacon where other nodes are guaranteed some specific bandwidth and low latency. Beacons are broadcasted every 15.38ms up to 252s depending on the beacon order [6], and contain information such as the network name and information required for synchronization, including the addresses of the coordinator and of other nodes. Each ZigBee network has a beacon order and superframe order. Increasing the beacon order will increase the time between subsequent beacons, and the superframe order has to be equal to or less than the beacon order as the superframe must be able to fully transmit within one beacon length. These Personal Area Network (PAN) variables are either transmitted in the beacon packet, or in the case of a non-beaconed network, in the response to a beacon request. Beacon enabled networks are more costly in terms of power consumption, and as a result this project focused on non-beaconing networks that do not require a superframe. In the current system, a node can find active networks simply by performing an active scan. An active scan sends out a beacon request frame which is then captured by passive network coordinators and replied to with a single beacon frame.

4.1.3 – RFMessenger

RFMessenger is a sample application supplied by Integration and includes a limited use license. This license allows for reuse and modification of the given code as long as this permission is acknowledged in the documentation and derivative source code. RFMessenger is a basic chat program running on Windows XP that allows for text messages to be sent from one

ZigBee device to another. This program was easily modified to fit the needs of the project as well as helping to determine the real-time creation and discovery of the ZigBee network, along with how connections and transmissions are performed. The GUI for this program had to be able to interface with the GUI that was created in Stage 1. RFMessenger was modified in such a way that it was able to display the name of the network to which it is connected. The Java GUI is able to call the modified RFMessenger program so that the user can then select which camera is needed. This will further developed in Section 4.2.2.

4.2 – Camera Device

The camera device code was developed for the Linux platform using C. Although since the ZigBee USB dongles are not truly Plug and Play compatible, device drivers needed to be installed. These drivers allowed for the ZigBee devices to act as temporary hard drives, allowing for the ability to open, read and write to the devices. The ZigBee devices also came with two sample Linux applications; the major point of interest was the library shared by the both of them, “mac.h.” This library condenses all the MAC functions into one neat package and greatly simplifies writing code. The library provided callback function interfaces for commands that needed to be customized for proper operation of the camera device. The list of functions that were used in the camera device code is listed in Appendix H, Table H-2.

Development led to the noticing of a small, almost unseen error in the “mac.c” file (the implementation file of “mac.h”). The bug was that the parameter representing the beacon payload length was not being properly set. This led to memory issues while trying to assign a name to the network. Since the network name was being improperly assigned, the beacon payload became invalid and any device trying to access the network would not be able to pass the garbled beacon frame past the physical layer. The ideal solution would be to first check if the payload length attribute was being set inside the function that saves all the MAC variables. Then, if the MAC attribute which was being set was indeed the payload length, then the associated parameter would be assigned to the MAC attribute variable. However an error within either the firmware of the dongle or the management of the code resulted in the ideal solution unusable. The result is that the payload length had to be hard-coded within the “mac.c” file. While not necessarily a major set back, it does limit some of the extensibility; the project cannot be fully customizable without directly modifying the library functions.

4.2.1 – Camera Algorithm

The algorithm used to start a ZigBee network is used only in the initialization of the camera device. The ZigBee network initialization routine is described by the flowchart in Figure 4-1. The only setup that is required for the camera device is upon deployment of the system. The user will need to place the camera above the meter of the client and then manually adjust the focus of the camera. Then they will then be able to execute the camera software. The program starts by parsing any command line parameters; these are not required however as the camera will still be able operate without them. The program will then set the PAN variables found in Table 4-2. The program then waits for incoming connections and then chooses the correct subroutine to execute based on the packet identifiers.

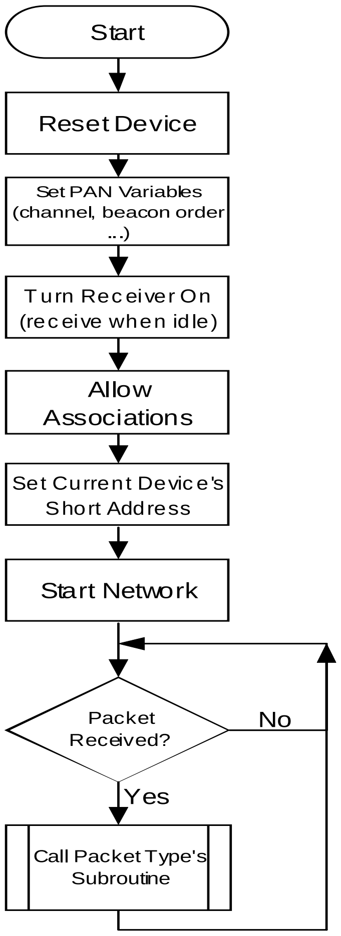

Start
Reset Device
Set PAN Variables (channel, beacon order ...)
Turn Receiver On (receive when idle)
Allow Associations
Set Current Device's Short Address
Start Network
Packet Received?
No
Yes
Call Packet Type's Subroutine

Figure 4-1. Initialization Routine

Upon initialization, the camera device issues a number of MAC variables to the ZigBee device. The commands are listed below in Table 4-2.

Table 4-2. Commands Executed for Camera Device Initialization

Command	Variable Set
Reset	n/a
Set Channel	11 (Default)
Receive on Idle	True
Set Beacon Payload Length	48 Bytes
Set Beacon Payload	“Customer #”
Permit Association	True
Set Short Address	0000
Beacon Order	15
Start Network	n/a

These variables reflect the basic properties of the network. The beacon payload can be set as a unique identifier to identify which camera you are connecting to, and picture comes from which camera. This will be necessary in future revisions of the project but it is ignored for now. A beacon order of 15 does not mean that this is a beaconing network – it is used to represent a beacon disabled network.

On a device association request, the camera will assign an address to the incoming device. It then initiates the image capture process described in Section 3.1.2.3 while it waits for the reader device to connect. After connection the reader device sends a data message which commences the file transfer. Once the file transfer is complete, the camera sends an END command. This is shown below in Figure 4-2.

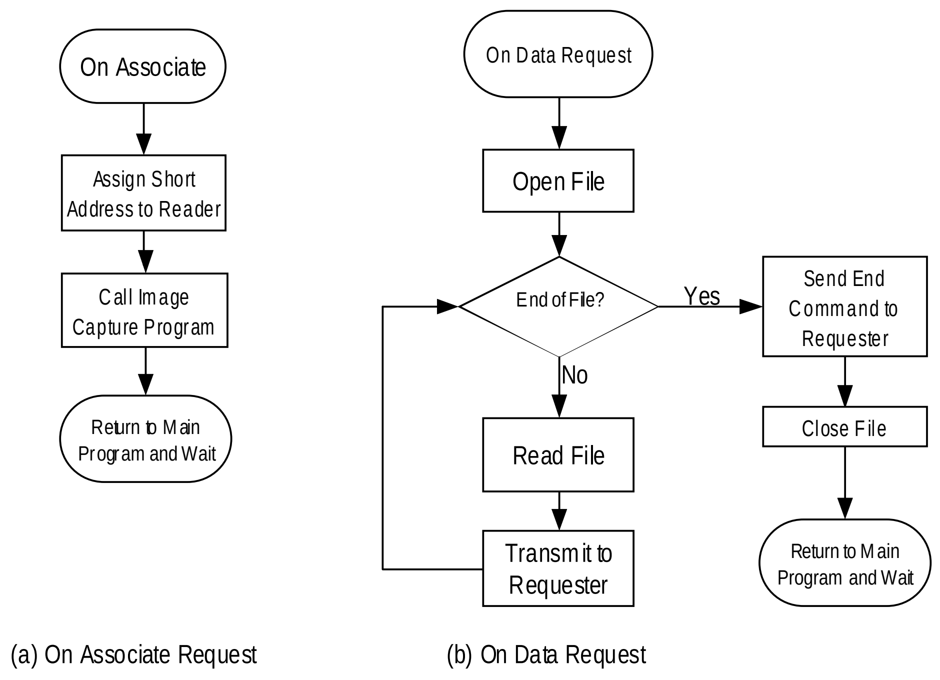

On Associate
Assign Short Address to Reader
Call Image Capture Program
Return to Main Program and Wait
On Data Request
Open File
End of File?
Yes
No
Send End Command to Requester
Close File
Read File
Transmit to Requester
Return to Main Program and Wait
(a) On Associate Request
(b) On Data Request

Figure 4-2. ZigBee File Transfer, Camera Side

Since the ZigBee protocol guarantees packet transfer via retransmission [6], there is no need for packet tracking and error detection. This implies that data sent from a transmitter will successfully reach the receiver unless an error is thrown by the ZigBee dongle. This feature makes the image transmission significantly easier than Stage 1.

4.2.2 – Java to C++

In order to interface the Java GUI developed in Stage 1 to the ZigBee program, coded in C++, an interface between the two languages had to be created. This is done by using the built in support of the Java *Runtime* and *Process* objects. These are both found in the default java.lang library, and using a simple sequence of library functions can be made to run an external program. The algorithm for accomplishing this is as follows:

1. Get the current system runtime environment through the *Runtime* object.
2. Call the *exec* function on the runtime environment, passing the desired executable program as a parameter.
3. Store the result of the exec function into a *Process* object.
4. Use the *waitfor* function of the *Process* object to wait for the executable opened in step 2 to terminate.

This simple set of library calls allows for the interface of any external software with a Java program. This works independent of the system the code is being run on.

4.3 – Reader Device

The reader device was developed by using a modified version of the RFMessenger program in C++. The use of RFMessenger allows the reader to actively scan for, connect to and join existing networks. Since the project assumes only one camera and one reader at time, it follows that there is only one network active at a time. This will be discussed in further detail in Section 4.4. Most of the options available in the RFMessenger GUI have been able to be automated in order to minimize the actions that are required by the user. The Java GUI created in Stage 1 calls the modified RFMessenger program, selects the appropriate network and associates the reader with it. RFMessenger then receives the image from the ZigBee link and writes it to a file where the Java program can display it.

Since the project is reusing the RFMessenger source code, the functions used to connect to the physical hardware were built in. The various functions needed to detect, connect and transmit across the network were also included in the software. This reduced the amount of original code needed for the project and shifted the focus from writing code to access the ZigBee network, to accessibility code which was needed to automate the entire image capture project.

4.3.1 – Reader Algorithm

Upon receiving the initialization signal from the Java GUI, the reader program starts up and scans for networks using an active scan. The algorithm will choose a network based on the lowest channel it finds. If the reader detects several networks on the same channel it chooses the network with the best signal quality by comparing the energy levels detected. If no response is detected by the scan, the reader will then start its own ZigBee network; otherwise it will associate itself to the previously selected ZigBee network. This detection and connection algorithm is depicted below in Figure 4-3.

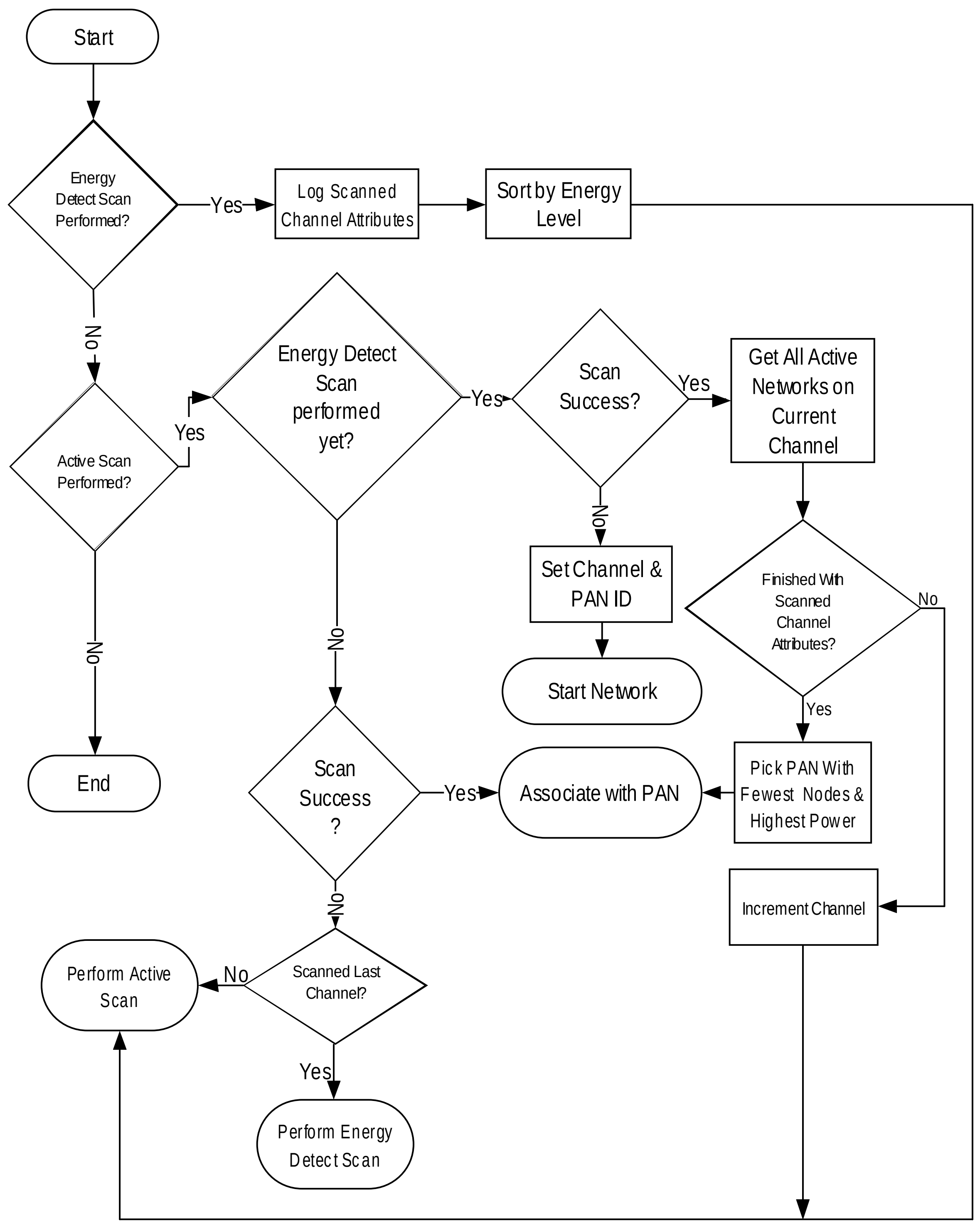

Start
Energy Detect Scan Performed?
Yes
Log Scanned Channel Attributes
Sort by Energy Level
No
Active Scan Performed?
Yes
Energy Detect Scan performed yet?
Yes
Scan Success?
Yes
Get All Active Networks on Current Channel
No
Set Channel & PAN ID
Start Network
Finished With Scanned Channel Attributes?
No
Yes
No
No
End
Scan Success ?
Yes
Associate with PAN
Pick PAN With Fewest Nodes & Highest Power
Increment Channel
No
Perform Active Scan
No
Scanned Last Channel?
Yes
Perform Energy Detect Scan

Figure 4-3. PAN Selection

After the reader device is done associating to a ZigBee network, it sends a message in the format of a data request packet. The reader will then wait for the camera to reply. It is possible to add in a timeout so the reader does not wait infinitely long for a reply. However, it is assumed that the ZigBee network has very little interference and that the network will not be lost indefinitely. Once the reader receives the END command, it disconnects from the network and writes the image to file. It then informs the Java GUI to display the received picture. This algorithm is displayed below in Figure 4-4.

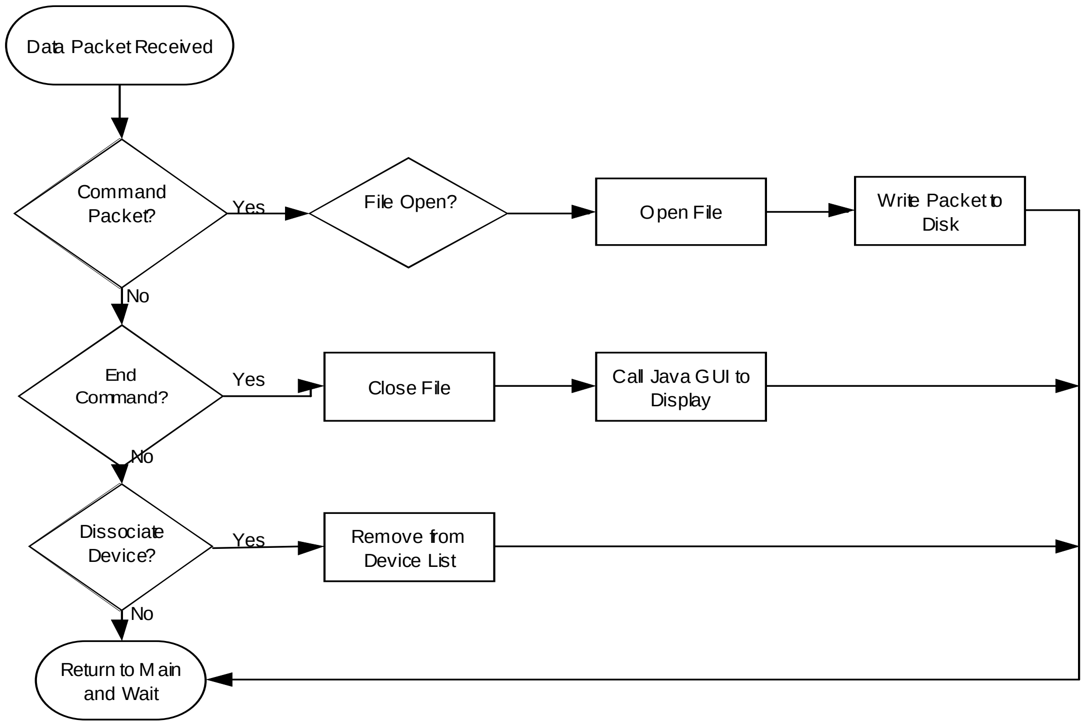


Figure 4-4. ZigBee File Transfer, Reader Side

4.3.2 – C++ to Java

An interface for running a Java program from inside a C++ program is required on the camera's system. Once again, the ZigBee software is written in C++ while the camera capture is written in Java. Executing a Java program from inside a C++ program is also straight forward; however it involves more system specific implementation issues than what was necessary to run another program from Java. The interface program was written for a UNIX based operating system, such as Linux, and will only work on that. It uses two commands from the library *unistd.h*, the *fork* and *execvp* functions.

The *fork* command splits the current execution thread into two separate processes, and *execvp* will execute the supplied command and arguments. Once a call to *fork* is made, the forked process will be a copy of the original process, only with a different process identifier. Both copies of the program will commence execution immediately following the *fork* command.

The procedure for running a Java program from C++ is similar, but quicker than doing the inverse in Java. It is simply:

1. Call *fork* and save the Process ID (PID) returned by it.
2. Check the PID of the currently execution process
    - If it is 0, then this process is the child process
    - If it is not 0, then this process is the parent process and needs to wait for the child to finish by using the wait(0) function, which waits for process 0 to finish.
3. If the PID is for the child, then *execvp* is called with the supplied system call being "Java" and the parameters being the class name, in this case "Camera".

This algorithm, when combined with the algorithm discussed in Section 4.2.2

4.4 – Tweaking & Limitations

There was one major problem implementing the transmission program. In the original implementation, whenever the camera device received a request for data it would immediately burst all the data it had. This caused the transmitter program to crash and the transmitter device to timeout and malfunction. The only solution to this critical malfunction was to restart the entire camera system; simply restarting the program would just detect an unresponsive ZigBee device. The most likely suspected error is a buffer overflow error in the ZigBee transmitter, probably due to the large amount of data being flushed from the write buffer in the camera software. This means that after transmitting a packet there must be either a command to tell the program to wait while the buffer finishes writing data, or a command to flush the buffer. The former is used within this project. A side effect of this is that it slows the transmission down considerably. Therefore, instead of waiting for the buffer to clear after every packet transmission, the program waits 1 second for every 15 packets transmitted. This is likely not the most optimal solution, as the method of determining the optimal packet transmission and wait periods were determined via trial and error. The revised algorithm is shown below in Figure 4-5.

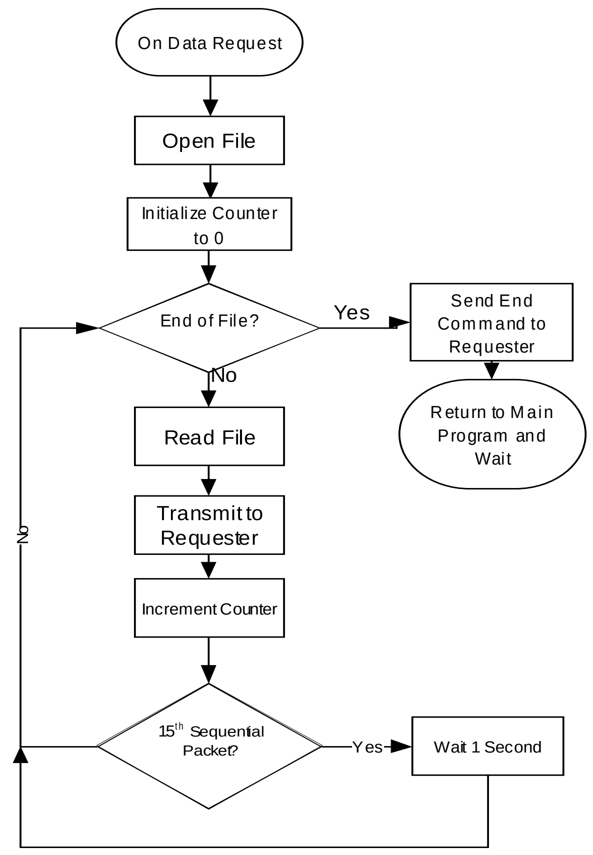


Figure 4-5. Revised Transmission Algorithm

There is a limitation regarding the number of reader devices that are allowed on the network. Since the design assumption is that only one reader device will be able to connect to the network at any given time, the camera software does not assign a unique PAN address nor does it take into account any disassociate events generated by the reader. The solution to this problem, while simple, is not implemented in the project as it is not necessary for successful completion of the design solution. The solution for allowing multiple readers per camera device would be to generate and maintain a list of all short addresses given out and then generate new ones whenever a device requests to be associated to the network. When a reader generates a disassociate event, the camera would then remove the address from its list.

Another current limitation is that the reader cannot choose which network to join. It will automatically connect to the network with the highest transmission power on the lowest channel. At the moment, this is not a problem as the assumption is that there is only one of each device. The problem can be solved by generating a list of available networks whenever a scan is performed, and then displaying it for the user to select from. Again, this was not implemented as it exceeds the requirements outlined in the project goals.

5 – Performance Analysis

5.1 – Camera

The performance of the camera was measured in a controlled environment, and the results are summarized in Table 5-1. The camera was run with a simple driver program that simply started the camera, took a picture, then closed the camera; this was repeated 100 times and the different data was recorded.

The parameters that the procedure tests, and that the data shows are summarized as follows:

- Location time for the Camera.
- Startup Time for the Camera.
- Initialization Time for the Camera.
- Time to initialize a Video Stream.
- Time to get a frame from the Video Stream.
- Time to write the frame to the Video Stream.
- Time required closing the Camera.

The times given in Table 5-1 reflect the time required to complete all of these tasks.
The parameters that are not tested by this procedure include anything related to the communication between the devices. This is run separately from the camera software and tests the effectiveness of the ZigBee connection, the range and the transfer bandwidth for different file sizes. The performance of the ZigBee system will be explored in subsection 5.2.

Table 5-1. Performance of the Camera System

Parameter	Windows	Linux
Number of Executions	100	100
Total Time	358.828 seconds	367.383 seconds
Average Time	3.588 seconds	3.674 seconds
Maximum Time	9.141 seconds	6.029 seconds
Minimum Time	3.078 seconds	3.401 seconds

The test was run on both a Windows machine and a Linux machine in order to compare the efficiency of the vendor supplied Windows drivers compared to user created Linux drivers that were used in the project. As it can be shown, the Linux drivers are slightly slower on average than the Windows ones; however this difference is only 2% and thus is not a very significant when looking at the big picture.

Further examination of Table 5-1 shows several factors. There is guaranteed to be a large time delay between when an image is requested and when the image arrives at the user's machine due to the delay in the camera. This delay is independent of any communication delays and thus must be looked at separately. As well, the data for the Linux driver suggests that it is more predictable; there is only a 2.628 second difference between the best and worse times, compared to a 6.063 second difference for the Windows drivers.

Despite the time delay being large (3.674 seconds), the specifications given for the system require a time delay of less than 30 seconds; this leaves approximately 26 seconds for the transfer progress, which should be more than enough excepting extreme interference with the wireless signal.

5.2 – ZigBee Communication

The performance analysis was executed in Linux and the results are shown in Table 5-2. The performance analysis includes only the speed of transmission and does not measure reception times. The analysis was executed in this fashion because the bottleneck of the ZigBee communication link is due to the limited amount of bandwidth available to the transmitter; the only limitation on the receiver is the speed at which it can empty its buffer and write to the disk.

The test parameters did not include the camera execution time, which is summarized above in Section 5.1. In addition, the test parameters did not include the start up or shut down time for the entire camera system, as it is an "always-on" type of device. For the test, the transmitter and the receiver were place approximately 60 cm apart. This was done reduce the effect of noise and range on the ZigBee link. Therefore, the test results in Table 5-2 reflect only the speed of the transmitter.

Table 5-2. Performance of the ZigBee Communication Link

Parameter	10KB	100KB	1MB
Number of Executions	25	25	5
Total Time	182 seconds	1,879 seconds	3,715 seconds
Average Time	7.28 seconds	75.16 seconds	743 seconds
Maximum Time	8 seconds	76 seconds	743 seconds
Minimum Time	6 seconds	74 seconds	743 seconds
Average Throughput	1.37 KB/s	1.33 KB/s	1.35 KB/s

The test results in Table 5.2 show that the transmission time is a linear function of the size of the file. That is, as the file size increases the transmission time will increase accordingly. Extrapolating the results to achieve a transmission time of 30 seconds, the maximum allowable file size was found to be 41.1KB. This does not include the camera execution times outlined in subsection 5.1, and as a result, the true allowable maximum file size is somewhat smaller. An implication of this is that files significantly larger than 10KB will not meet the transmission specifications outlined earlier in Section 2.1.2. However, for files between 10KB and 30KB, the transmission specifications can easily be met given that there is minor interference due to attenuation, noise and other ZigBee systems attempting to access the network.

The results of the average throughput are far below the possible output of ZigBee, which is approximately 30KB/s (250 kbps) [6]. This seems to be due to limitations imposed by the written software and the ZigBee devices used. As outlined earlier in Section 4, the transmission is done by first transmitting 15 packets sequentially, and then waiting 3 seconds for the buffer to clear. To improve the speed of the transmitter, the buffers in the code as well as the buffers on the ZigBee transmitter must be either extended or flushed quicker.

Since the system can be modeled as a linear system, the effective time is the time required to transfer a file plus the time required to take a picture of the meter. This puts the average time at (15 + 3.5 =) 18.5 seconds for a file around 20KB.

6 – Extensions

During design discussions, it was discovered that a large amount of extensions could be applied to the Wi-UR system. These extensions could also be applied as a design project for a group in following years. This section will be dedicated to elaborating on the possible extensions to this project that could be made for the future.

The main extra feature that was discussed in initial design meetings is that the whole process could be entirely automated. This would require the use of several new systems; a web server and an Optical Character Recognition (OCR) technique. Optical Character Recognition would enable the system to take an image of meter and automatically determine what the value is. In order for this system to be deployed in any sort of large scale operation, the security of the entire system must be enhanced to protect the privacy of the user and guarantee the accuracy of the measurements. In order to implement these changes however, some basic changes must be made to the entire system.

6.1 – Communication Link

The current communication link employed in the Wi-UR system is implemented using ZigBee. However, for the system to be fully automated and distributed it is an impracticality to use ZigBee devices as the communication sources of choice. The reason for this is that although ZigBee is low power with a decent range, there is no infrastructure support for a mass distributed ZigBee network. In the future, if these ZigBee systems are to be distributed amongst many households, it would be possible to create a mesh network. By using a mesh network, a fully automated system could be created that would forward the information captured from each individual camera device and forward it to a common destination. This would lead to less development time since the only source that would need to be written would be the routing algorithms which can be adopted from current Ethernet routing algorithms. This would also solve the high power cost at the expense of building a ZigBee infrastructure. However, while a mesh network could be set up for the purposes of this device, it would still be economically prohibitive to set up such a network. This is due to the limited range on the transmitters requiring an extreme number of nodes to place a city on a mesh network.

The system could be better implemented using wireless Ethernet, since it can piggy-back off pre-existing Ethernet networks in households and neighbourhoods. While this switch would make the communication routines more standard, wireless Ethernet requires more power than a ZigBee device. This in turn would require additional development into ways to put the device to sleep for long periods of time, to wake up only when it needs to transmit its data back to the centralized database.

6.2 – Database

To have a completely automated process, a database must be implemented that is capable of accepting requests and storing all the incoming data. For the database, the ideal system of implementation would be a centralized web server. This server would have its own IP Address, allowing for easy communication between it and the Wi-UR devices, which would be running wireless Ethernet. Since both devices would be connected directly to the Ethernet grid currently in existence, and both devices would have their own IP addresses, a transmission protocol such as TCP could be employed to guarantee reliability of transmissions in this setup.

By having an automated database, you eliminate the need to send a worker out to the house to gather the data. The handheld reader becomes obsolete and the parts of the software for it become incorporated into the server's support software.

6.3 – Optical Character Recognition

Possibly the most demanding task out of the extensions would be adding in Optical Character Recognition, or OCR, to the database system. Using OCR would allow for everything to be fully automated, since a human would not be necessary to interpret the image. However, in order to perform OCR on this image would require a complex segmentation algorithm as well as determining which numbers are the ones that should be kept and which should be discarded. Numbers would have to be discarded because of model numbers and other such extraneous information that is not required for the solution to the problem. Due to the non-uniformity of legacy meters installed in houses, this would be an extremely difficult task to perform.

6.4 – Security

To reliably trust the measurements obtained and guarantee the privacy of the clients, both the device positioning and the data transmission must be secure. The device security could easily be supplied simply by a container with alarms that would be tripped if the container was tampered with. The reader would then have to be able to differentiate between altered and unaltered devices. For transmission security, ZigBee devices have a built-in security option to encrypt data symmetrically using the Advanced Encryption Standard (AES) [6]. In addition to using the built-in security options, the reader and transmitter applications should employ their own encryption schemes to verify the identities of each device. This would disallow rogue devices from impersonating the reader or camera devices, preventing the collection of private data or transmission of false measurements.

A project involving all four of these extensions could be capable of being used in later years as an additional design project for an undergraduate thesis.

7 – Conclusions

To solve the problem of remote meter reading, knowledge of external device interfacing, as well as the ZigBee Protocol was required. The actual implementation of the system was done using a combination of two different programming languages, Java and C/C++. The ZigBee communication link was coded in C and C++, while the camera and the interface were coded in Java. The camera was interfaced to the host machine through an additional Java package known as the Java Media Framework. To get the project working, drivers had to be found and used for the design. These drivers were not simply the vendor supplied drivers for the devices, and some had to be searched for and found online. The usage of the drivers also had to be learnt and understood for the developed software to be able to correctly interface and control them.

The project consisted solely of a proof of concept design; as such most of the hardware was emulated on other systems. The system for the camera is a simulated Linux 2.4 kernel run on a personal computer, and a laptop is being used as the handheld reader. It was determined that while emulating these components did not detract from the difficulty of the design, they made the project more cost-effective. If the design is to be taken out of a prototype stage and into development for implementation in the environment, a transition from the emulated systems to their physical representations should be straight forward and easily accomplished.

Transmission using ZigBee was discovered to be slower than a comparable transmission over different wireless standards, such as Bluetooth or Ethernet, rating at an average of 1.35KB/s up to a maximum of roughly 31KB/s. This is due to the lower transmitter power utilized with a ZigBee communication. There is no free lunch, and something must be sacrificed to have a low power, wireless transmission device. However, despite this limitation the communication speed on small image files is not prohibitively long and can still be used with adequate results in this design solution.

The project is also capable of acting as a starting point for future projects, should students want to pursue a similar subject. The project can be extended into a fully automated system with emphasis on automatic data collection from a centralized location, rather than individuals going around collecting the required data.

In conclusion, it was determined that the project is feasible for implementation in the world provided that the systems are ported to their physical components, and that the software is idealized for the type of implementation that is required.

References

[1] Integration Associates, Inc. *Integration IEEE 802.15.4/ZigBee^{TM} USB Dongle IA OEM-DAUB12400 Datasheet*. Internet, http://www.integration.com/docs/IA_OEM-DSUB1_2400.pdf. 2006.

[2] *Universal Serial Bus Revision 2.0 specification.* Internet, http://www.usb.org/developers/docs/usb_20_05122006.zip. 2006.

[3] IEEE. *IEEE 802.11g-2003. Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications—Amendment 4: Further Higher-Speed Physical Layer Extension in the 2.4 GHz Band*. Internet, http://standards.ieee.org/getieee802/download/802.11g-2003.pdf. 2003.

[4] Sun Microsystems. *JMF 2.0 API (03/10/01)*. Internet, http://java.sun.com/products/java-media/jmf/2.1.1/apidocs/index.html. 2001.

[5] R.C. Gonzalez and R. E. Woods, *Digital Image Processing, Second Edition*. New Jersey: Prentice Hall, 2002.

[6] ZigBee Alliance. *ZigBee Specification. v1.0*. Internet, http://www.zigbee.org. 2004.

Appendix A – Installer Guide for JMF on Linux

Download the Linux Platform JMF .bin file, **`jmf-2_1_1e-linux-i586.bin`** from the Sun Microsystems webpage

To install, enter the following commands while logged in as root:

1. `apt-get update`
2. `apt-get install fakeroot java-package`
3. `fakeroot make-jpkg jre-1_5_0_06-linux-i586.bin`
4. `ls *deb`
   - `Let the filename given by this command be` **`[javafile].`**
5. `dpkg -i` **`[javafile]`**`.deb`

This will install a java runtime environment on your machine.

To set the Java paths, do the following:

- `vi /etc/profile`
- `Press 'a' to edit.`
- `Add the following if it is not present:`
  - `JAVA_HOME="usr/lib/java"`
  - `export JAVA_HOME`
- `Press escape and type ':wq' to save and exit.`

To verify that java was installed properly, type

- `java -version`

Once the Java Runtime Environment is installed, place **`jmf-2_1_1e-linux-i586.bin`** in the directly /usr/lib

Type

- `sh ./jmf-2_1_1e-linux-i586.bin`
  - `You will be asked a few questions, give the suggested answers.`

It is possible you will have to copy some files into your java folders. Find the folder that your java runtime environment was installed into, let this folder be **[java]** (it will be something like /user/lib/jre…)

From the JMF directory, type

- `cp *.so `**`[java]`**`/lib/ext`
- `cp *.jar `**`[java]`**`/lib/i386`

To set JMF paths type:

- `vi /etc/profile`
- `Press ‘a’ to edit.`
- `Add the following if it is not present:`
    - `JMFHOME="user/lib/JMF-2.1.1e"`
    - `export JMFHOME`
    - `LD_LIBRARY_PATH="$JMFHOME/lib:$JAVA_HOME/lib/i368:$JAVA_HOME/lib/client"`
    - `export LD_LIBRARY_PATH`
- `Press escape and type ':wq' to save and exit.`

After this is done, JMF should be useable on your Linux Machine.

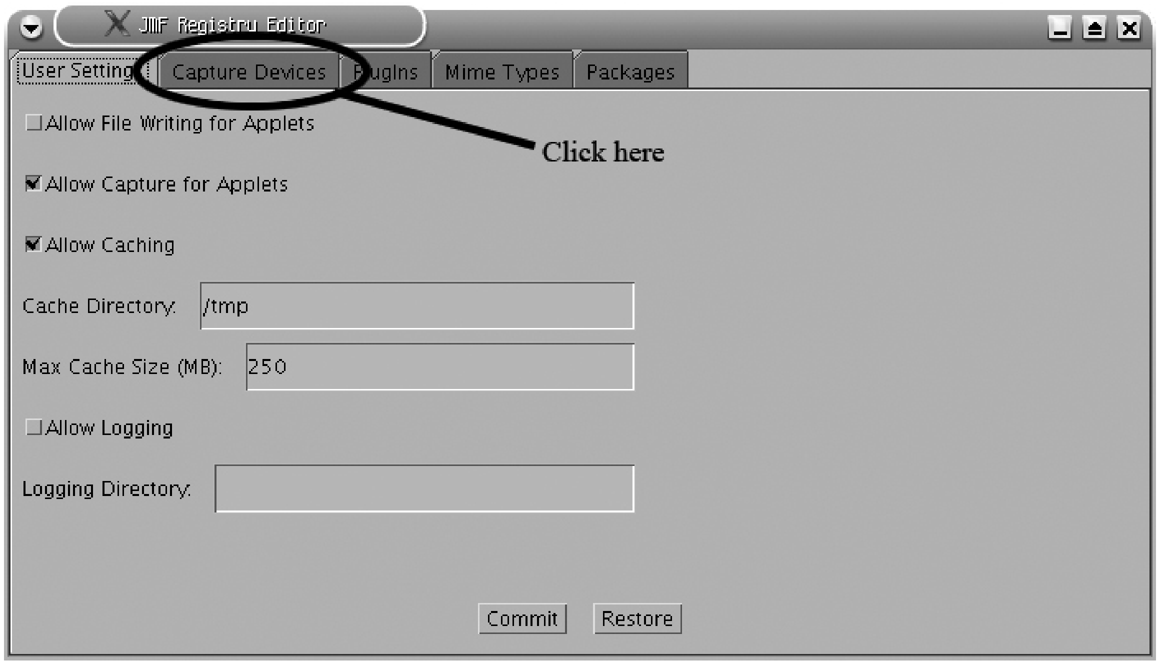
Appendix B – Using JMFRegistry on Linux

You will need to use the command line and be logged in as the root. This can be done by typing "su root" and then entering your password.

Next, find the directory that JMFRegistry.class is located in. Get to this directory using the "`cd`" command.

Once at this directory, type "`java JMFRegistry`"

Once inside the Registry, click on the "Capture Devices" tab as shown in Figure B-1.



☐ Allow File Writing for Applets

☑ Allow Capture for Applets

☑ Allow Caching

Cache Directory: /tmp

Max Cache Size (MB): 250

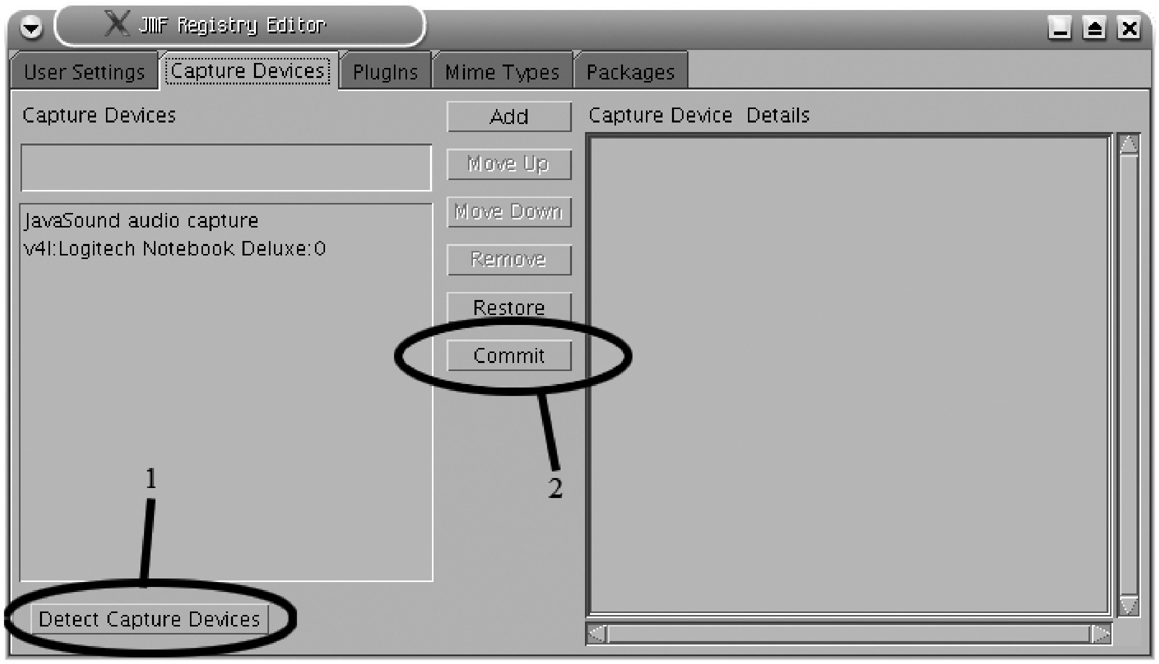
☐ Allow Logging

Logging Directory:

Figure B-1. JMFRegistry Initial GUI

Next, click on "Detect Capture Devices", wait for it to finish, and then click on "Commit" to save changes. This is shown in Figure B-2 as 1 and 2, respectively.

After doing these 3 steps, close the Registry and your devices should be useable with the Java Media Framework.



JavaSound audio capture
v4l:Logitech Notebook Deluxe:0

Figure B-2. JMFRegistry

Appendix C – Emulation Guide

The original emulation program used was Microsoft's Virtual PC. This was used because the software license was free. However the one major problem encountered was that it does not support USB devices. Thus, Virtual PC was dropped in favour of VMware.

VMware does require that its user pay a licensing fee for their Workstation program, however there is a trial version for it. In addition VMware has another program called VMware Player which can use virtual machines created in VMware Workstation. The bonus to this is that VMware Player is free to use.

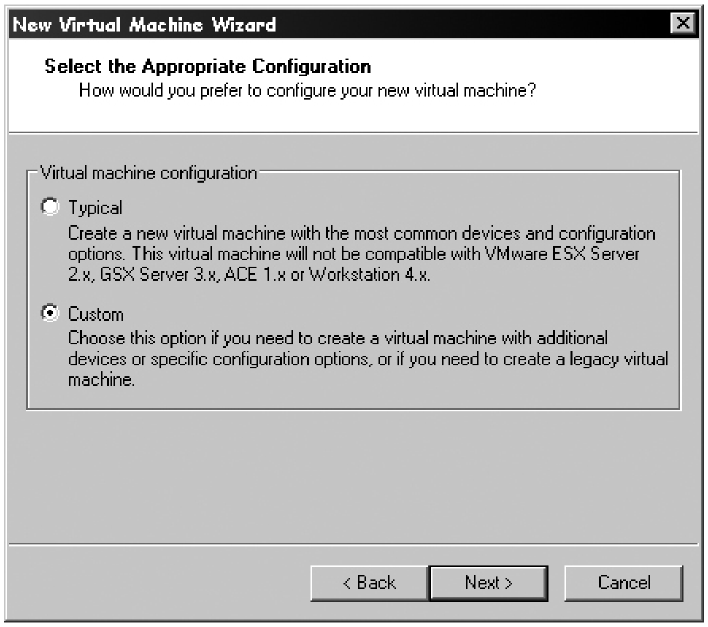
The program installed was VMware Workstation v5.5.3.34685 which can download via the VMware webpage (www.vmware.com).

- Double click to install.
- Follow the on-screen instructions.

You have now installed VMware on your system.

To create a machine:

- Click File, New Virtual Machine
- Click the custom machine option as shown in Figure C-1



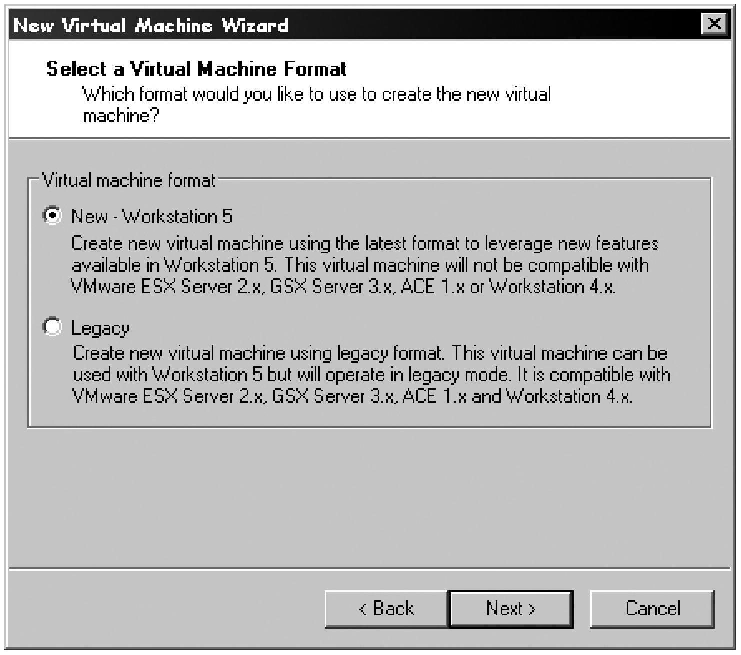
Select the Appropriate Configuration
How would you prefer to configure your new virtual machine?

Virtual machine configuration

- Typical
  Create a new virtual machine with the most common devices and configuration options. This virtual machine will not be compatible with VMware ESX Server 2.x, GSX Server 3.x, ACE 1.x or Workstation 4.x.
- [✓ Custom]
  Choose this option if you need to create a virtual machine with additional devices or specific configuration options, or if you need to create a legacy virtual machine.

Figure C-1. Virtual Machine Configuration

- Click on either option in the next window, depending on what you need as shown in Figure C-2.



Select a Virtual Machine Format
Which format would you like to use to create the new virtual machine?

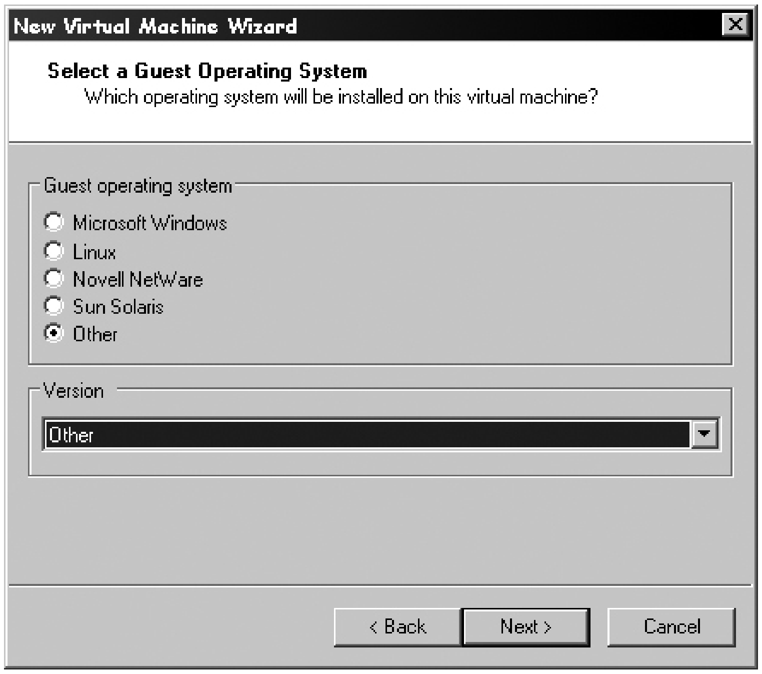
Virtual machine format

[✓ New - Workstation 5]
Create new virtual machine using the latest format to leverage new features available in Workstation 5. This virtual machine will not be compatible with VMware ESX Server 2.x, GSX Server 3.x, ACE 1.x or Workstation 4.x.

Legacy
Create new virtual machine using legacy format. This virtual machine can be used with Workstation 5 but will operate in legacy mode. It is compatible with VMware ESX Server 2.x, GSX Server 3.x, ACE 1.x and Workstation 4.x.

Figure C-2. Select Virtual Machine Format

- Select the “Other” option for both “Guest Operating System” and “Version” this is shown in Figure C-3.



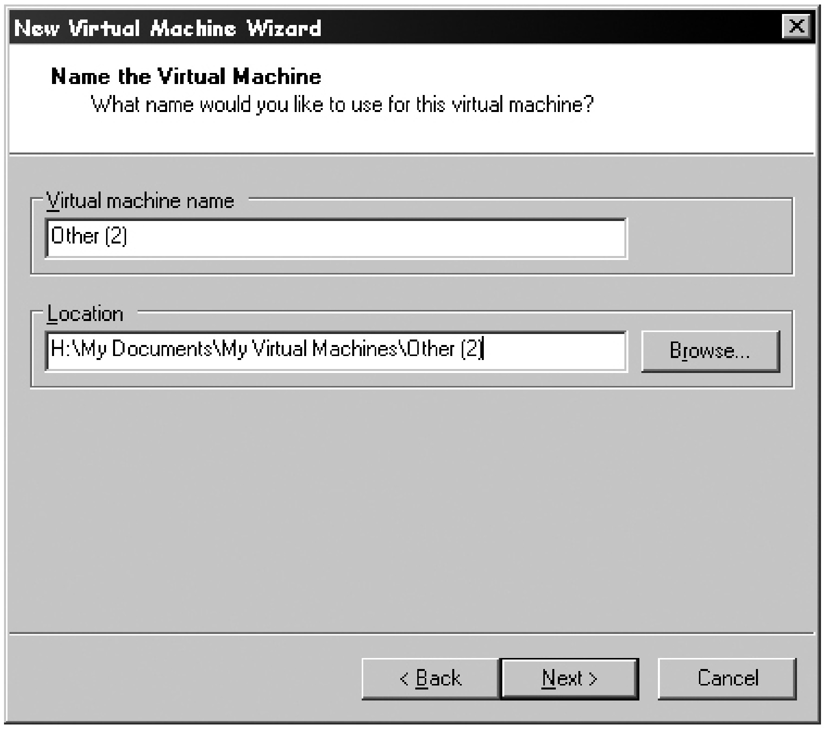
Select a Guest Operating System
Which operating system will be installed on this virtual machine?

Guest operating system
Microsoft Windows
Linux
Novell NetWare
Sun Solaris
[✓ Other]

Version
Other

Figure C-3. Select a Guest Operating System

- Choose a name for your virtual machine and the location to store it in. This is shown in Figure C-4.

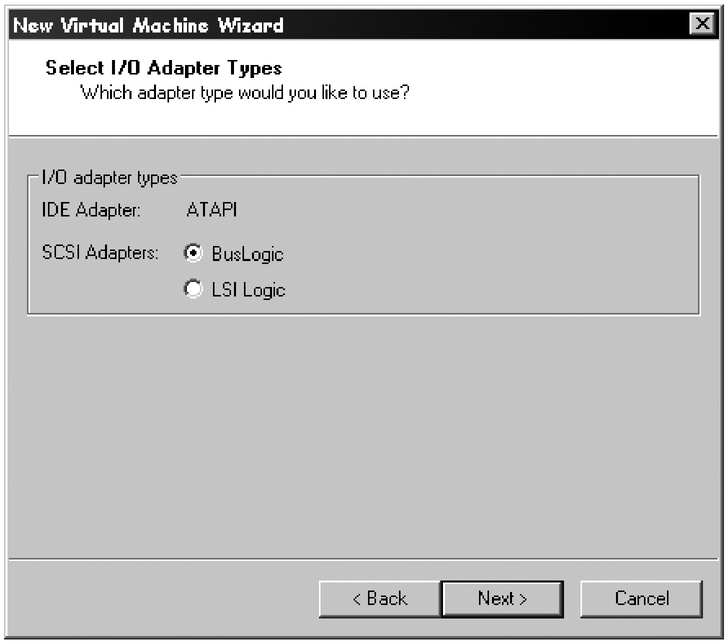

New Virtual Machine Wizard
Name the Virtual Machine
What name would you like to use for this virtual machine?
Virtual machine name
Location
Browse...
< Back
Next >
Cancel

Other (2)

H:\My Documents\My Virtual Machines\Other (2)

Figure C-4. Choose a Virtual Machine Name

- Choose the number of processors you want to emulate. This project uses one.
- Choose the amount of RAM to simulate. This project initially started at 256MB of RAM to use Knoppix's GUI.
- Choose "Bridged Networking" in the Network Connection options. It doesn't really matter for this project the only communication link needed is ZigBee.
- Choose the "BusLogic" option for SCSI Adapters. This is demonstrated in figure C-5.


New Virtual Machine Wizard
Select I/O Adapter Types
Which adapter type would you like to use?
I/O adapter types
< Back
Next >
Cancel

IDE Adapter: ATAPI

SCSI Adapters: [✓ BusLogic] LSI Logic

Figure C-5. Select I/O Adapter Type

- Select “Create a New Disk.”
- Select IDE for Virtual Disk Type.
- Set the Disk Capacity to 4GB to install Knoppix to the hard drive. Otherwise set it to any value you wish.
- Call the disk anything you want. This file will store whatever is installed onto the machine.
- After creating the virtual machine, click on “Edit virtual machine settings”.
- Click the “Add…” under the Hardware tab.
- Hit “Next” at the “Add Hardware Wizard”.
- Click “USB Controller” under “Hardware Types”
- Then click “Finish”

To detect a USB device to the virtual machine:

- Attach it to the computer
- Select VM, Removable Devices and select the device from the list

Appendix D – Knoppix Installation Guide

To install Knoppix to the hard drive:

Download Knoppix image from a website. The project used KNOPPIX_V3.7-2004-12-08-EN.iso

- Boot the machine from the CD-ROM. This will start Knoppix.
- At the prompt, type in “knoppix” without the quotation marks. This will start Linux’s 2.4 kernel.
- Wait while it configures itself.
- Hit CTRL-ALT-F1 to bring up the console
- Type “sudo knoppix-installer” without the quotation marks.
- Use the built-in software to format and the drive. Make sure to create both a root and swap partition.
    - If the software is not creating the partitions, you will have to manually set the partitions.
    - Type in “mkswap /dev/hdb5” to make this a swap partition. Again, do not include quotation marks
    - Type in “edit /etc/fstab”, no quotation marks.
    - Replace the entry for “/dev/hdb5” with the following line “/dev/hdb5 none swap defaults 0 0” without quotation marks
    - Finally, type “swapon –a” to enable the swap partition
- This project used the default options for installation.

Linux is now installed on to the machine.

Appendix E – Camera Source Code

Camera.java

```
import java.awt.*;
import java.awt.image.*;
import java.io.*;
import java.util.*;
import java.awt.event.*;
import java.io.*;
import javax.imageio.*;

import javax.media.*;
import javax.media.control.*;
import javax.media.format.*;
import javax.media.util.*;

class NoCameraException extends Exception
{
        String message;

        public NoCameraException()
        {
                message = null;
        }

        public NoCameraException(String message)
        {
                this.message = message;
        }

        public String getMessage()
        {
                return message;
        }
}

public class Camera
{
        public static void main(String[] args)
        {
                Camera c = null;

                try
                {
                        c = new Camera(320,240);
                }
                catch (NoCameraException nce)
                {
                        System.out.println(nce.getMessage());
                        System.exit(0);
                }

                c.startCamera();

                while (!c.takePicture("wiur.jpg"));

                c.stopCamera();

                System.exit(0);

        }

        //Dimensions
        private int xSize;
        private int ySize;
        //Scaling Factors
        private double xScaleFactor;
        private double yScaleFactor;
        private static final String CAMERA_NAME = "v4l:Logitech Notebook Deluxe:0";

        private String outputFileName = null;

        private Player p;
        private FrameGrabbingControl pictureTaker;
        private BufferToImage bufferToImage = null;
        private boolean closedDevice;

        private static final int MAX_TRIES = 300; //try 300 times
        private static final int TRY_PERIOD = 100; //milliseconds

        //Make a new camera
        public Camera(int xSize, int ySize) throws NoCameraException
        {
                this.xSize = xSize;
                this.ySize = ySize;
                closedDevice = true;

                try
                {
                        MediaLocator ml = findMedia(CAMERA_NAME);

                        p = Manager.createRealizedPlayer(ml);
                        System.out.println("Found Camera");
                }
                catch (Exception e)
                {
                        throw new NoCameraException("Error initializing Camera");
                }

                // create the frame grabber
                pictureTaker =  (FrameGrabbingControl)
p.getControl("javax.media.control.FrameGrabbingControl");
                if (pictureTaker == null)
                {
                        System.out.println("Frame grabber could not be created");
                        System.exit(0);
                }
    }

        //Start and initialize the camera
    public void startCamera()
    {
                // wait until the player has started
                System.out.println("Starting the player...");
                startPlayer(p);

                System.out.println("Player started");
                initCamera();
        }

        //Stop the camera by stopping the player
        public void stopCamera()
        {
                p.close();
                closedDevice = true;
        }

        //Check if the camera is currently closed.
        public boolean isClosed()
        {
                return closedDevice;
        }

        //Start the Camera, which is represented as a Player
    private void startPlayer(Player p)
    {
                double time;
```

```
               p.start();

               //Wait for the player to start, 5 second "timeout"
               time = System.currentTimeMillis();
               while (p.getState() != Controller.Started && System.currentTimeMillis() -
time < 5000);

               if (p.getState() != Controller.Started)
                       startPlayer(p);
       }

       //Sets the output resolution of the camera
    public String setResolution(int xSize, int ySize)
    {
               //Only set the resolution if the device is closed (off)
               if (closedDevice)
               {
                       this.xSize = xSize;
                       this.ySize = ySize;
               }
               else
                       return "Device not closed";

               return null;
       }

       //Get a media locator for the needed device
       private MediaLocator findMedia(String requireDeviceName)
       {
               //Get the device list
               Vector devices = CaptureDeviceManager.getDeviceList(null);

               if (devices == null || devices.size() == 0)
               {
                       return null;
               }

               //Search the device list for the one we want.
               for (int i = 0; i < devices.size(); i++)
               {
                       CaptureDeviceInfo devInfo = (CaptureDeviceInfo)
devices.elementAt(i);
                       String devName = devInfo.getName();
                       // found device
                       if (devName.equals(requireDeviceName))
                       {
                               System.out.println("Found device: " + requireDeviceName);
                               return devInfo.getLocator();
                       }
               }

               return null;
       }

       //Initialize the camera
       private void initCamera()
       {
               int tries = 0;
               System.out.println("Initializing Camera...");

               //Keep trying to initialize it until we give up (after 300 tries)
               while (tries < MAX_TRIES)
               {
                       if (cameraInitialized())   // initialization succeeded
                       {
                               closedDevice = false;
                               return;
                       }
                       try
                       {   // initialization failed so wait a while and try again
                               System.out.println("Waiting...");
				Thread.sleep(TRY_PERIOD);
			}
			catch (InterruptedException e)
			{
				System.out.println(e);
			}
			tries++;
		}

		if (tries == MAX_TRIES)
		{
			System.out.println("Giving Up");
			closedDevice = true;
			return;
		}

		closedDevice = false;   // device now available
	}

	//Checks if the camera is initialized by trying to take a picture
	private boolean cameraInitialized()
	{
		//Try to take a picture
		Buffer buf = pictureTaker.grabFrame(); //get a frame

		//No picture taken; camera is not initialized
		if (buf == null)
		{
			System.out.println("No grabbed frame");
			return false;
		}

		//No data received; camera is not initialized
		if (buf.getLength() == 0)
		{
			return false;
		}

		// there is a buffer, but check if it's empty or not
		VideoFormat vf = (VideoFormat) buf.getFormat();

		if (vf != null) //We have data and the camera is initialized
		{
			// the image's dimensions
			int width = vf.getSize().width;
			int height = vf.getSize().height;

			//scale the image
			xScaleFactor = ((double) xSize) / width;
			yScaleFactor = ((double) ySize) / height;

			// initialize bufferToImage with the video format info.
			bufferToImage = new BufferToImage(vf);
			return true;
		}

		return false;
	}

	//Takes a picture using the camera
	public boolean takePicture(String outputFile)
	{
		if (closedDevice)
			return false;

		outputFileName = outputFile;

		// grab the current frame as a buffer object
		Buffer buf = pictureTaker.grabFrame();
		if (buf == null)
		{
```

```
                        return false;
                }

                // convert buffer to image
                Image im = bufferToImage.createImage(buf);
                if (im == null)
                {
                        return false;
                }

                writeToFile(im);

                return true;
        }

        //Writes the Image im to a file
        private void writeToFile(Image im)
        {
                BufferedImage pic = new BufferedImage(xSize, ySize,
BufferedImage.TYPE_INT_RGB);

                //Format the buffered image for output
                Graphics2D g2d = pic.createGraphics();

                g2d.scale(xScaleFactor, yScaleFactor);
                g2d.drawImage(im,0,0,null);
                g2d.dispose();

                try
                {
                        ImageIO.write(pic, "jpg", new File(outputFileName));
                }
                catch (IOException ioe)
                {
                }
        }
}
```

Appendix F – Communication System Source Code

F–1. Client

ImageClient.java

```
/*********************************************************************
*
*   ImageClient.java
*
*   This class implements the client half of a file transfer protocol
*
*********************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class ImageClient
{
	/* Data */

	// Private Variables
	private static BufferedReader inFromUser   = new BufferedReader(new
InputStreamReader(System.in));
	private static DatagramSocket clientSocket;
	private static InetAddress    ipAddress;
	private static FileOutputStream writer;
	private static String filename;
	private static String message;
	private static int length;
	private static int seqNum = 0;
	private static DatagramPacket sendPacket;
	private static DatagramPacket receivePacket;

	private static byte[] sendData;
	private static byte[] receiveData = new byte[1024];
	private static byte[] dataBuffer;

	private static long timestamp;
	private static double percent;
	private static double oldpercent = 100;

	private static int state = States.SEND_RFF;

	private static Vector missing;

	/* Public Methods */

	private static void init()
	{
		inFromUser = new BufferedReader(new InputStreamReader(System.in));
		clientSocket = null;
		ipAddress = null;
		writer = null;
		filename = "";
		message = "";
		length = 0;
		seqNum = 0;
		sendPacket = null;
		receivePacket = null;

		sendData = null;
		receiveData = new byte[1024];
		dataBuffer = null;

		timestamp = 0;
		percent = 0;
		oldpercent = 100;

		state = States.SEND_RFF;

		missing = null;

	}

	public static byte[] getImage(String file) throws Exception
	//public static void main(String args[]) throws Exception
	{
		//byte[] ip = {24,78,119,23};
		//byte[] ip = {24,76,122,104};
		byte[] ip = {127,0,0,1};

		init();

		clientSocket = new DatagramSocket();
		ipAddress =
InetAddress.getByAddress(ip);//InetAddress.getByName("localhost");
		//clientSocket.setSoTimeout(1000);

		filename = file;
		//getFilename();

		System.out.println(filename);

		while (!filename.equals("quit"))
		{
			switch (state)
			{
				case States.SEND_RFF:
					seqNum = -1;
					sendRFF();
					state = States.NEED_LENGTH;
					break;
				case States.NEED_LENGTH:
					waitLength();
					if (state != States.GET_INPUT)
						state = States.SEND_CTS;
					else
						sendNAK();
					break;
				case States.SEND_CTS:
					sendCTS();
					state = States.NEED_START;
					break;
				case States.NEED_START:
					waitStart();
					state = States.RECEIVE_FILE;
					break;
				case States.RECEIVE_FILE:
					receiveFile();
					if (missing.size() >0)
					{
						state = States.TRANSMIT_MISSING;
					}
					else
						state = States.SEND_GOOD;
					break;
				case States.GET_INPUT:
					/*
					if (writer != null)
						writer.close();
					getFilename();
					state = States.SEND_RFF;
```

```
                                */
                                filename = "quit";
                                break;
                        case States.TRANSMIT_MISSING:
                                transmitMissing();
                                if (missing.size() <=0)
                                        state = States.SEND_GOOD;
                                else
                                        state = States.NEED_START;
                                break;
                        case States.SEND_GOOD:
                                sendGood();
                                state = States.GET_INPUT;
                                break;
                }
        }
        clientSocket.close();
        if (writer != null)
                writer.close();

        return dataBuffer;
    }

    /* Private Methods */
    /**
     * Sends the "CTS" message to the Server.
     *
     * @return      void
     */
    private static void transmitMissing() throws Exception
    {
        System.out.println("Sending Missing");
        message = Message.MISSING;
        message = message + missing.toString();
        sendData = message.getBytes();
        sendPacket = new DatagramPacket(sendData, sendData.length, ipAddress,
9876);
        clientSocket.send(sendPacket);
        writer = new FileOutputStream("new"+filename);
    }

    /**
     * Gets a file name from the user.
     *
     * @return      void
     */
    private static void getFilename() throws Exception
    {
        System.out.print("Enter a file to read: ");
        filename = inFromUser.readLine();
    }

    /**
     * Performs a blocking wait for the Client until it receives the
     * "START" message from the Server.
     *
     * @return      void
     */
    private static void waitStart() throws Exception
    {
        System.out.println("Waiting for start");
        while (true)
        {
                receiveData = new byte[Message.size];
                receivePacket = new DatagramPacket(receiveData,
receiveData.length);
                clientSocket.receive(receivePacket);
                message = new String(receiveData);

                System.out.println(message.trim());
                if(message.startsWith(Message.START))
				{
					return;
				}
			}
		}

		/**
		 * Sends the "CTS" message to the Server.
		 *
		 * @return	void
		 */
		private static void sendCTS() throws Exception
		{
			System.out.println("Sending CTS");
			message = Message.CTS;
			sendData = message.getBytes();
			sendPacket = new DatagramPacket(sendData, sendData.length, ipAddress,
9876);
			clientSocket.send(sendPacket);
		}

		/**
		 * Sends the "GOOD" message to the Server.
		 *
		 * @return	void
		 */
		private static void sendGood() throws Exception
		{
			System.out.println("Sending GOOD");
			message = Message.GOOD;
			sendData = message.getBytes();
			sendPacket = new DatagramPacket(sendData, sendData.length, ipAddress,
9876);
			clientSocket.send(sendPacket);
		}


		/**
		 * Sends the "NAK" message to the Server.
		 *
		 * @return	void
		 */
		private static void sendNAK() throws Exception
		{
			System.out.println("Sending NAK");
			message = Message.NAK;
			sendData = message.getBytes();
			sendPacket = new DatagramPacket(sendData, sendData.length, ipAddress,
9876);
			clientSocket.send(sendPacket);
		}

		/**
		 * Sends the "RFF" message to the Server.
		 *
		 * @return	void
		 */
		private static void sendRFF() throws Exception
		{
			System.out.println("Sending RFF");
			message = Message.RFF + filename;
			sendData = message.getBytes();
			sendPacket = new DatagramPacket(sendData, sendData.length, ipAddress,
9876);
			clientSocket.send(sendPacket);
		}

		/**
		 * Performs a blocking wait for the Client until it receives the
		 * "LENGTH" message from the Server.
		 *
	 * @return	void
	 */
	private static void waitLength() throws Exception
	{
		System.out.println("Waiting for Length");
		while (true)
		{
			receiveData = new byte[Message.size];
			receivePacket = new DatagramPacket(receiveData, 
receiveData.length);
			clientSocket.receive(receivePacket);

			message = new String(receivePacket.getData());
			if(message.startsWith(Message.LENGTH))
			{
				missing = new Vector();
				length = 
Integer.parseInt(message.trim().substring(Message.LENGTH.length()));
				for (int i=0; 
i<Math.ceil((double)length/(double)Message.size); i++)
				{
					missing.add(new Integer(i));
				}
				dataBuffer = new byte[length];
				writer = new FileOutputStream("new"+filename);
				return;
			}
			else if (message.startsWith(Message.NAK))
			{
				state = States.GET_INPUT;
				return;
			}
		}
	}

	/**
	 * Accepts packets with sequence numbers and stores them in a byte array.
	 *
	 * @return	void
	 */
	private static void receiveFile() throws Exception
	{
		//missing = new Vector();
		int oldSeq = -1;
		seqNum = -1;

		timestamp = (new Date()).getTime();

		while ((seqNum+1) * Message.size < length && missing.size() >0)
		{
			receiveData = new byte[Message.size + 2];
			receivePacket = new DatagramPacket(receiveData, 
receiveData.length);
			try
			{
				clientSocket.receive(receivePacket);
	//				System.out.println(dataBuffer.length);
				seqNum = Message.extractSeqNum(receiveData);
//				for (int i=oldSeq+1; i<seqNum; i++)
//					missing.add(new Integer(i));
				missing.remove(new Integer(seqNum));
				oldSeq = seqNum;
//				System.out.println(seqNum);
				System.arraycopy(receiveData, 0, dataBuffer, 
seqNum*Message.size, Math.min(Message.size, dataBuffer.length-(seqNum*Message.size)));
				percent = (int) (100.0 * 
(double)Math.min(dataBuffer.length, ((seqNum+1)*Message.size))/length);
				if (percent != oldpercent)
				{
					System.out.println("Received " + percent + "% of " + 
filename);
```

```
                                        oldpercent = percent;
                                }
                        }
                        catch (SocketTimeoutException ste)
                        {
                                System.out.println("Timeout");
                                break;
                        }
                }
                writer.write(dataBuffer);
                writer.close();

                System.out.println("Missing seqNum: " + missing);
                timestamp = (new Date()).getTime() - timestamp;
                System.out.println("Transfer took " + timestamp + " ms");
                System.out.println("Done receiving " + filename);
        }
}
```

F–2. Server

UDPServer.java

```
/******************************************************************************
*
*   UDPServer.java
*
*   This class implements the server half of a file transfer protocol
*
******************************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;

public class UDPServer
{
       /* Data */

       // Private Variables
       private static DatagramSocket serverSocket;
       private static FileInputStream reader;
       private static byte[] receiveData = new byte[Message.size];
       private static byte[] sendData;// = new byte[1024];
       private static String sentence;
       private static DatagramPacket receivePacket;
       private static DatagramPacket sendPacket;
       private static String filename;
       private static int seqNum = 0;
       private static long latency;
       private static String[] missingNumbers;

       private static int state = States.NEED_RFF;

       /* Public Methods */
       public static void main(String args[]) throws Exception
       {
              serverSocket = new DatagramSocket(9876);
              receivePacket = new DatagramPacket(receiveData, receiveData.length);

              while(true)
              {
                     if (state != States.SENDING_FILE)
                     {
                            receiveData = new byte[Message.size];
                            receivePacket = new DatagramPacket(receiveData,
receiveData.length);
                     }

                     switch (state)
                     {
                            case States.NEED_RFF:
                                   if (reader != null)
                                          reader.close();
                                   waitRFF();
                                   seqNum = 0;
                                   break;
                            case States.NEED_CTS:
                                   latency = (new Date()).getTime();
                                   waitCTS();
                                   latency = (new Date()).getTime() - latency;
                                   System.out.println("Latency = " + latency);
                                   state = States.SENDING_FILE;
                                   break;
                            case States.SENDING_FILE:
                                   transmitFile();
                                   state = States.WAIT_MISSING;
                                   break;
```

```
                        case States.WAIT_MISSING:
                            waitMissing();
                            break;
                    }
                }
        }

        private static void takePicture() throws Exception
        {
                Camera c = null;

                try
                {
                        c = new Camera(320,240);//Camera(320,240);
                }
                catch (NoCameraException nce)
                {
                        System.out.println(nce.getMessage());
                        System.exit(0);
                }

                c.startCamera();

                while (!c.takePicture(filename));

                c.stopCamera();
        }

        /* Private Methods */
        /**
         * Performs a blocking wait for the Server until it receives the
         * "MISSING" message from the Client.
         *
         * @return      void
         */
        private static void waitMissing() throws Exception
        {
                System.out.println("Waiting for MISSING");
                while (true)
                {
                        receiveData = new byte[Message.size];
                        receivePacket = new DatagramPacket(receiveData,
receiveData.length);
                        serverSocket.receive(receivePacket);
                        sentence = new String(receiveData);

                        if(sentence.startsWith(Message.MISSING))
                        {
                                sentence =
sentence.trim().substring(Message.MISSING.length());
                                sentence = sentence.substring(1);
                                sentence = sentence.substring(0,sentence.length()-1);
                                System.out.println(sentence);
                                missingNumbers = sentence.split(", ");


                                System.out.println("Sending start");

                                sentence = Message.START;
                                InetAddress ipAddress = receivePacket.getAddress();
                                int port = receivePacket.getPort();

                                sendData = null;
                                sendData = sentence.getBytes();

                                sendPacket = new DatagramPacket(sendData, sendData.length,
ipAddress, port);
                                serverSocket.send(sendPacket);


                                System.out.println("Just sent " + new String(sendData));
```

```
				Thread.sleep(5+latency);
				transmitMissing();
				return;
			}
			else if (sentence.startsWith(Message.GOOD))
			{
				state = States.NEED_RFF;
				return;
			}
		}
	}

	/**
	 * Breaks down a file into reasonable sized packets (1022 bytes), and then
	 * sends these to the client.
	 *
	 * @return      void
	 */
	private static void transmitFile() throws Exception
	{
		int test;
		File temp = new File(filename);
		sendData = new byte[Message.size];
		test = reader.read(sendData);
		while(test != -1)
		{
			InetAddress ipAddress = receivePacket.getAddress();
			int port = receivePacket.getPort();

			//if (seqNum != 6 && seqNum != 7)
			//{
				sendData = Message.attachSeqNum(seqNum, sendData);
				Thread.sleep(5 + latency);
				sendPacket = new DatagramPacket(sendData, sendData.length,
ipAddress, port);
				serverSocket.send(sendPacket);
			//}

			sendData = new byte[Message.size];
			test = reader.read(sendData);
			seqNum++;
		}
	}

	/**
	 * Retransmits missing packets to the client.
	 *
	 * @return      void
	 */
	private static void transmitMissing() throws Exception
	{
		int test;
		File temp = new File(filename);
		reader = new FileInputStream(filename);
		sendData = new byte[Message.size];
		test = reader.read(sendData);
		int i=0;
		int needed;

		seqNum = 0;

		while(test != -1 && i < missingNumbers.length)
		{
			needed = Integer.parseInt(missingNumbers[i]);
			InetAddress ipAddress = receivePacket.getAddress();
			int port = receivePacket.getPort();
			if (seqNum == needed)
			{
				sendData = Message.attachSeqNum(seqNum, sendData);
				Thread.sleep(5 + latency);
```

```
                              sendPacket = new DatagramPacket(sendData, sendData.length, 
ipAddress, port);
                              serverSocket.send(sendPacket);
                              sendData = new byte[Message.size];
                              i++;
                      }
                      test = reader.read(sendData);
                      seqNum++;
              }
       }

       /**
        * Performs a blocking wait for the Server until it receives the
        * "CTS" message from the Client.
        * After receiving the "CTS" message, this method sends out the
        * "START" message to continue the protocol.
        * 
        * @return      void
        */
       private static void waitCTS() throws Exception
       {
              System.out.println("Waiting for CTS");
              while (true)
              {
                      receiveData = new byte[Message.size];
                      receivePacket = new DatagramPacket(receiveData, 
receiveData.length);
                      serverSocket.receive(receivePacket);
                      sentence = new String(receiveData);

                      if(sentence.startsWith(Message.CTS))
                      {
                              sentence = Message.START;
                      }
                      else if (sentence.startsWith(Message.NAK))
                      {
                              state = States.NEED_RFF;
                              return;
                      }
                      else
                      {
                              sentence = Message.NAK;
                      }

                      InetAddress ipAddress = receivePacket.getAddress();
                      int port = receivePacket.getPort();

                      sendData = null;
                      sendData = sentence.getBytes();

                      sendPacket = new DatagramPacket(sendData, sendData.length, 
ipAddress, port);
                      serverSocket.send(sendPacket);

                      return;
              }
       }

       /**
        * Performs a blocking wait for the Server until it receives the
        * "RFF" message from the Client.
        * After receiving the "RFF" message, this method sends out the
        * "LENGTH" message to continue the protocol.
        * 
        * @return      void
        */
       private static void waitRFF() throws Exception
       {
              System.out.println("Waiting for RFF");
              while (true)
              {
```

```
                        receiveData = new byte[Message.size];
                        receivePacket = new DatagramPacket(receiveData,
receiveData.length);
                        serverSocket.receive(receivePacket);
                        sentence = new String(receiveData);

                        if(sentence.startsWith(Message.RFF))
                        {
                                filename = sentence.trim().substring(Message.RFF.length());

                                try
                                {
                                        takePicture();
                                        reader = new FileInputStream(filename);
                                        File temp = new File(filename);
                                        sentence = Message.LENGTH + temp.length();
                                        state = States.NEED_CTS;
                                }
                                catch (IOException ioe)
                                {
                                        System.out.println(ioe.getMessage());
                                        state = States.NEED_RFF;
                                        sentence = Message.NAK;
                                }

                                InetAddress ipAddress = receivePacket.getAddress();
                                int port = receivePacket.getPort();

                                sendData = null;
                                sendData = sentence.getBytes();

                                sendPacket = new DatagramPacket(sendData, sendData.length,
ipAddress, port);
                                serverSocket.send(sendPacket);

                                return;
                        }
                }
        }
}
```

F–3. States

States.java

```
/**********************************************************************
*
*   States.java
*
*   This class contains constants for different states of the system.
*
**********************************************************************/

public class States
{
       /* Data */

       // Public Constants

       public final static int NEED_RFF = 0;
       public final static int NEED_CTS = 1;
       public final static int TRANSMIT_MISSING = 2;
       public final static int SENDING_FILE = 3;
       public final static int NEED_LENGTH = 4;
       public final static int RECEIVE_FILE = 5;
       public final static int NEED_START = 6;
       public final static int SEND_RFF = 7;
       public final static int SEND_CTS = 8;
       public final static int SEND_LENGTH = 9;
       public final static int GET_INPUT = 10;
       public final static int SEND_GOOD = 11;
       public final static int NEED_GOOD = 12;
       public final static int WAIT_MISSING = 13;
}
```

F–4. Message Routines

Message.java

```
/*********************************************************************
*
*   Message.java
*
*   This class contains constants for different messages used in the system.
*
*********************************************************************/

public class Message
{
	/* Data */

	// Public Constants
	public final static String DONE = "DONE";
	public final static String RFF = "RFF";
	public final static String CTS = "CTS";
	public final static String MISSING = "MISSING";
	public final static String GOOD = "GOOD";
	public final static String FINISHED = "FINISHED";
	public final static String LENGTH = "LENGTH";
	public final static String START = "START";
	public final static String ACK = "ACK";
	public final static String NAK = "NAK";

	public final static int size = 1022;
	//0 to 65535 are valid sequence numbers

	/* Public Methods */
	/**
	 * Removes a sequence number from the start of a byte array.
	 *
	 * @param      data   The byte array of the data that contains a sequence number
	 * @return            The extracted sequence number
	 */
	public static int extractSeqNum(byte[] data)
	{
		int result = 0;
		int num;
		byte temp;
		byte[] tempData = new byte[data.length-2];

		temp = data[0];
		num = (int)temp;
		num = num << 8;
		num = num & 0xFF00;
		result = result + num;

		temp = data[1];
		num = (int)temp;
		num = num & 0xFF;
		result = result + num;


		System.arraycopy(data, 2, data, 0, data.length-2);
		return result;
	}

	/**
	 * Attaches a sequence number at the start of a byte array.
	 *
	 * @param      seqNum The sequence number to attach
	 * @param      data   The byte array of the data that needs a sequence number
	 * @return            The new byte array with the sequence number at the start
	 */
	public static byte[] attachSeqNum(int seqNum, byte[] data)
	{
		int num;
		byte temp;
		byte[] sendData = new byte[data.length + 2];

		System.arraycopy(data, 0, sendData, 2, data.length);

		num = seqNum >> 8;
		num = num & 0xFF;
		temp = (byte)num;
		sendData[0] = temp;
		num = seqNum;
		num = num & 0xFF;
		temp = (byte)num;
		sendData[1] = temp;

		return sendData;
	}

}
```

Appendix G – Graphical User Interface Source Code

GUI.java

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class GUI
{
	public static void main(String[] args)
	{
		MainFrame window = new MainFrame();

		window.show();
		window.pack();
	}
}

class MainFrame extends JFrame
{
	ImageIcon testPic;// = new ImageIcon(null);
	JLabel image;

	public MainFrame()
	{
		Container content = getContentPane();

		JLabel text = new JLabel("Wi-UR Image Receiver");
		JPanel subLayout = new JPanel();
		JButton button = new JButton("Aquire Image");
		Font textFont = text.getFont();
		textFont = textFont.deriveFont(24.0f);
		text.setFont(textFont);

		subLayout.setLayout(new BoxLayout(subLayout, BoxLayout.Y_AXIS));

		setTitle("Wi-UR");
		content.setLayout(new BorderLayout());

		image = new JLabel(testPic);
		content.add(image, BorderLayout.CENTER);

        button.setAlignmentX(Component.CENTER_ALIGNMENT);
        button.addActionListener(new AquireListener(this));
		subLayout.add(button);

		content.add(subLayout, BorderLayout.SOUTH);


		text.setHorizontalAlignment(SwingConstants.CENTER);
		content.add(text, BorderLayout.NORTH);
	}

	public void setImage(String filename)
	{
		Container content = getContentPane();
		Image img;

		content.remove(image);

		img = Toolkit.getDefaultToolkit().createImage(filename); //prevents
caching of the image
		testPic = new ImageIcon(img);//new ImageIcon(filename);
		image = new JLabel(testPic);

		content.add(image, BorderLayout.CENTER);
		pack();
	}
}

class AquireListener implements ActionListener
{
	MainFrame parent;

	String[] command;
	String PROGRAM_NAME = "RFMessenger.exe";


	public AquireListener(MainFrame parent)
	{
		this.parent = parent;
		command = new String[2];
		command[0] = PROGRAM_NAME;
	}

	public void actionPerformed(ActionEvent e)
	{
		String name;

		try
		{
			name = JOptionPane.showInputDialog(null,  "Enter Filename");

			command[1] = name;

			Process proc = Runtime.getRuntime().exec(command);
			proc.waitFor();

			//ImageClient.getImage(name);
			parent.setImage(name);
		}
		catch (Exception ex)
		{
			ex.printStackTrace();
			System.exit(0);
		}
	}
}
```

Appendix H – Camera Device ZigBee Link Source Code

camera.c

```
/*
 * 802.15.4 snooper.
 *
 * Written by Jon Beniston <jbeniston@integration.com>
 *
 * Simple snooper application that puts the Steeple 3 MAC into
 * promicuous mode and outputs all the MCPS-DATA.indications
 * that are received.
 *
 * Copyright 2005 Integration Associates Inc.  All rights reserved.
 *
 * LIMITED USE LICENSE.  By using this software, the user agrees to the terms
 * of the following license.  If the user does not agree to these terms, then
 * this software should be returned within 30 days and a full refund of the
 * purchase price or license fee will provided.  Integration Associates
 * hereby grants a license to the user on the following terms and conditions:
 * The user may use, copy, modify, revise, translate, abridge, condense, expand,
 * collect, compile, link, recast, distribute, transform or adapt this software
 * solely in connection with the development of products incorporating
 * integrated circuits sold by Integration Associates.  Any other use for any
 * other purpose is expressly prohibited with the prior written consent of
 * Integration Associates.
 *
 * Any copy or modification made must satisfy the following conditions:
 *
 * 1. Both the copyright notice and this permission notice appear in all copies
 * of the software, derivative works or modified versions, and any portions
 * thereof, and that both notices appear in supporting documentation.
 *
 * 2. All copies of the software shall contain the following acknowledgement:
 * “Portions of this software are used under license from Integration Associates
 * Inc. and are copyrighted.”
 *
 * 3  Neither the name of Integration Associates Inc. nor any of its
 * subsidiaries may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY “AS IS” AND ALL WARRANTIES OF ANY KIND,
 * INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR USE,
 * ARE EXPRESSLY DISCLAIMED.  THE DEVELOPER SHALL NOT BE LIABLE FOR ANY DAMAGES
 * WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.  THIS SOFTWARE MAY NOT
 * BE USED IN PRODUCTS INTENDED FOR USE IN IMPLANTATION OR OTHER DIRECT LIFE
 * SUPPORT APPLICATIONS WHERE MALFUNCTION MAY RESULT IN THE DIRECT PHYSICAL
 * HARM OR INJURY TO PERSONS.  ALL SUCH IS USE IS EXPRESSLY PROHIBITED.
 *
 */

#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <assert.h>
#include <unistd.h>
#include <sys/select.h>
#include <sys/time.h>
#include "mac.h"
#include <time.h>
#include <float.h>
/*Define Camera variables*/
#define SIZE 102
#define BEACON_PAYLOAD_LENGTH 48
int pid;
```

```
char* cmd[2]={"java","Camera"};
typedef struct {
        mac_address_mode_t addrMode;
               uint8_t shortAddress;
               uint8_t beaconOrder;
               uint8_t superframeOrder;
               mac_pan_id_t panid;
               char network_name[BEACON_PAYLOAD_LENGTH];
               int channel;
} pan_descriptor_t;
pan_descriptor_t panInfo;
const char END[4]="END\0";

static int display_unknown_primitive (int fd, uint8_t primitive, uint8_t *data, uint8_t
length);

static int display_unknown_primitive (int fd, uint8_t primitive, uint8_t *data, uint8_t
length)
{
        char buffer[1024];
        char *b;
        int i;
       /* Display textual version of primitive */
        b = buffer;
       b += sprintf (b, "Unknown primitive %x: ", primitive);
        for (i = 0; i < length; i++) {
                b += sprintf (b, "%02x", data[i]);
        }
        sprintf (b, "\n");
        printf(buffer);
       return 1;
}

/* Display an error message and exit. */
static void error (int fd, const char *message)
{
        fprintf (stderr, "%s\n", message);
       if (errno != 0) {
               fprintf (stderr, "%s.\n", strerror (errno));
       }
        close (fd);
        exit (EXIT_FAILURE);
}

/*Confirm that network has started*/
static int display_MLME_START_confirm(
       int fd,
       mac_status_t status
       )
{
       if(status==mac_success)
               printf("Network starting.....\n");
       else
               printf("Could not start network!\n");

       return 1;
}

/*Association requested*/
static int ASSOCIATE_indication(
           int fd,
           uint8_t *DeviceAddress,
           uint8_t CapabilityInformation,
           _Bool SecurityUse,
           mac_acl_entry_t ACLEntry
          )
{
       /*Call the Java Camera program to take a picture*/
       pid=fork();

       if(pid!=0)
```

```
                wait(0);
        else
                execvp(cmd[0],cmd);

        /*Not done here, but check if device is already in network*/
        /*Set the short address of the device added*/
        uint16_t AssocShortAddress=0x0101;
        printf("Device requested to join network\n");
        MLME_ASSOCIATE_response(fd, DeviceAddress, AssocShortAddress, mac_success, 0);

        return 1;
}

/*When Coordinator is searching for device, associate self with the network*/
/*Probably not needed*/
static int display_MLME_BEACON_NOTIFY_indication(
        int fd,
        uint8_t BSN,
        mac_pan_descriptor_t *PANDescriptor,
        uint8_t PendAddrSpec,
        uint8_t *AddrList,
        uint8_t sduLength,
        uint8_t *sdu
        )
{
        assert (fd > 0);
         assert (PANDescriptor != NULL);

        printf("BSN:  %x\n", BSN);

        MLME_ASSOCIATE_request(fd, PANDescriptor->LogicalChannel , PANDescriptor-
>CoordAddrMode, PANDescriptor->CoordPANId, &PANDescriptor->CoordAddress,
MAC_CAPABILITY_DEVICE_TYPE_FFD, false);

        return 1;
}

/* Callback function to display a MCPS-DATA.indication */
static int display_MCPS_DATA_indication (
         int fd,
         mac_address_mode_t SrcAddrMode,
         mac_pan_id_t SrcPANId,
         mac_address_t *SrcAddr,
         mac_address_mode_t DstAddrMode,
         mac_pan_id_t DstPANId,
         mac_address_t *DstAddr,
         uint8_t msduLength,
         uint8_t *msdu,
         uint8_t mpduLinkQuality,
         _Bool SecurityUse,
         mac_acl_entry_t ACLEntry
         )
{

         assert (fd > 0);
         assert ((SrcAddrMode == mac_no_address) || (SrcAddrMode == mac_short_address) ||
(SrcAddrMode == mac_extended_address));
         assert ((SrcAddrMode == mac_no_address) || (SrcAddr != NULL));
         assert ((DstAddrMode == mac_no_address) || (DstAddrMode == mac_short_address) ||
(DstAddrMode == mac_extended_address));
         assert ((DstAddrMode == mac_no_address) || (DstAddr != NULL));
         assert ((msduLength == 0) || (msdu != NULL));


        /*If data indication from any source, send them a picture*/
        /*To be safe, add some checks but not done here*/
        FILE* inFile;
        int numRead;
        uint8_t buffer[SIZE];
        int i=0;
        time_t start,finish;
        /*Test information commented out*/
        /*int k;
        double minTime=DBL_MAX;
        double maxTime=0;
        double avgTime=0;
        double currTime=0;
        int numTrials=2;

        for(k=0;k<numTrials;k++)
        {
                time(&start);

                i=0;*/
                inFile = fopen("test.jpg", "rb");
                printf("\nFile Requested\n");
                while (!feof(inFile))
                {
                        numRead = fread(&buffer[1], sizeof(uint8_t), sizeof(buffer)-1,
inFile);
                        buffer[0]=0x01; /*Tell the receiver that this is a data
transmission*/

                        if( MCPS_DATA_request (fd, DstAddrMode, DstPANId, DstAddr,
SrcAddrMode, SrcPANId, SrcAddr, numRead+1, &buffer[0],
0,0/*MAC_TX_OPTION_ACKNOWLEDGED*/)<=0)
                                error(fd, "Error sending");
                        i++;
                        if(i%15==0) /*wait for the buffer to finish sending*/
                        sleep(1.0);

                }

                /*Send END command*/
                buffer[0]=0x00;
                buffer[1]=0x03;
                if( MCPS_DATA_request (fd, DstAddrMode, DstPANId, DstAddr, SrcAddrMode,
SrcPANId, SrcAddr, 2, buffer, 0,0/*MAC_TX_OPTION_ACKNOWLEDGED*/)<=0)
                        error(fd, "Error sending");

                else
                        printf("!%s File Transfer Complete!\n", END);

                fclose(inFile);

/*              time(&finish);

                currTime=difftime(finish,start);
                avgTime+=currTime;

                if(currTime>maxTime)
                        maxTime=currTime;
                if(currTime<minTime)
                        minTime=currTime;

        }
        avgTime/=numTrials;
        printf("# of Trials: %i\tAverage Time: %f\tMax. Time: %f\tMin. Time:
%f\n",numTrials, avgTime,maxTime,minTime);
    */
    return 1;
}

/* Put MAC into promiscuous mode and display all MCPS-DATA.indications */
int main (int argc, char *argv[])
{
        int fd;
        char *device = "/dev/steeple30";
        uint8_t channel = 0xb;
        uint8_t true_value = true;
        mac_primitive_handler_t handler = {0};
        int beacon_payload_length=BEACON_PAYLOAD_LENGTH;

        strcpy(panInfo.network_name, "RFMessenger");

        panInfo.shortAddress=0x000;
        panInfo.beaconOrder=0x0f;
        panInfo.superframeOrder=0x0f;
        panInfo.panid=0x0000;

        static const uint8_t reset_confirm[] = {mac_mlme_reset_confirm, mac_success};
        static const uint8_t channel_set_confirm[] = {mac_mlme_set_confirm, mac_success,
phyCurrentChannel};
        /*static const uint8_t promiscuous_set_confirm[] = {mac_mlme_set_confirm,
mac_success, macPromiscuousMode};*/
        static const uint8_t rxonidle_set_confirm[] = {mac_mlme_set_confirm, mac_success,
macRxOnWhenIdle};
        static const uint8_t macPayloadLength_set_confirm[]={mac_mlme_set_confirm,
mac_success, macBeaconPayloadLength};
        static const uint8_t macPayload_set_confirm[]={mac_mlme_set_confirm, mac_success,
macBeaconPayload};
        static const uint8_t macAssociationPermit_set_confirm[]={mac_mlme_set_confirm,
mac_success, macAssociationPermit};
        static const uint8_t macShortAddress_set_confirm[]={mac_mlme_set_confirm,
mac_success, macShortAddress};
        /* Process command line arguments */
        if (argc >= 2) {
        /* First argument is device name */
        device = argv[1];
        }
        if (argc == 3) {
                /* Second argument is channel number */
                panInfo.channel = atoi (argv[2]);
                if (!((panInfo.channel >= 0xb) && (panInfo.channel <= 0x1a))) {
                        fprintf (stderr, "Channel number must be between 11 and 26.\n");
                        exit (EXIT_FAILURE);
                }
        }
        if (argc > 3) {
                fprintf (stderr, "Invalid command line arguments.\n");
                fprintf (stderr, "Usage: %s [device [channel]]\n", argv[0]);
                 fprintf (stderr, "Default options are %s %d\n", device, panInfo.channel);
                 exit (EXIT_FAILURE);
        }

        /* Open the Steeple 3 device */
         if ((fd = open (device, O_RDWR)) > 0) {
                /* Reset the MAC, and then enable the receiver in promiscous mode */
                if (MLME_RESET_request (fd, 1) <= 0) {
                                error (fd, "Failed to send MLME-RESET.request.");
                }
                if (mac_receive_primitive (fd, reset_confirm, sizeof (reset_confirm)) <=
0) {
                                error (fd, "Failed to receive MLME-RESET.confirm.");
                }
                if (MLME_SET_request (fd, phyCurrentChannel, &channel) <= 0) {
                                error (fd, "Failed to send MLME-SET.request
(phyCurrentChannel).");
                }
                if (mac_receive_primitive (fd, channel_set_confirm, sizeof
(channel_set_confirm)) <= 0) {
                                error (fd, "Failed to receive MLME-SET.confirm
(phyCurrentChannel).");
                }
                if (MLME_SET_request (fd, macRxOnWhenIdle, &true_value) <= 0) {
                                error (fd, "Failed to send MLME-SET.request
(macRxOnWhenIdle)");
                }
                if (mac_receive_primitive (fd, rxonidle_set_confirm, sizeof
(rxonidle_set_confirm)) <= 0) {
```

```
                              error (fd, "Failed to receive MLME-SET.confirm
(macRxOnWhenIdle).");
               }
               if(MLME_SET_request(fd,macBeaconPayloadLength,&beacon_payload_length) <=0)
               {
                      error (fd, "Failed to send MLME-
SET.request(macBeaconPayloadLength)");
               }

       if(mac_receive_primitive(fd,macPayloadLength_set_confirm,sizeof(macPayloadLength_s
et_confirm))<=0)
               {
                      error(fd, "Failed to receive MLME-
SET.confirm(macBeaconPayloadLength)");
               }
               if(MLME_SET_request(fd,macBeaconPayload, &panInfo.network_name)<=0)
               {
                      error(fd,"Failed to send MLME-SET.request(macBeaconPayload)");
               }
               if(mac_receive_primitive(fd, macPayload_set_confirm,sizeof(
macPayload_set_confirm))<=0)
               {
                      error(fd, "Failed to receive MLME-SET.confirm(macBeaconPayload)");
               }
               if(MLME_SET_request(fd,macAssociationPermit, &true_value)<=0)
               {
                      error(fd,"Failed to send MLME-SET.request(macAssociationPermit)");
               }

       if(mac_receive_primitive(fd,macAssociationPermit_set_confirm,sizeof(macAssociation
Permit_set_confirm))<=0)
               {
                      error(fd,"Failed to receive MLME-
SET.confirm(macAssociationPermit)");
               }
               if(MLME_SET_request(fd,macShortAddress, &panInfo.shortAddress)<=0)
               {
                      error(fd,"Failed to send MLME-SET.request(macShortAddress)");
               }

       if(mac_receive_primitive(fd,macShortAddress_set_confirm,sizeof(macShortAddress_set
_confirm))<=0)
               {
                      error(fd,"Failed to receive MLME-SET.confirm(macShortAddress)");
               }
               if(MLME_START_request(fd,0x0000,0xb,0x0f,0x0f, &true_value, 0,0,0)<=0)
/*Change the battery extension when done***/
               {
                      error(fd,"Failed to start the network");
               }


        handler.MCPS_DATA_indication = display_MCPS_DATA_indication;
               handler.MLME_BEACON_NOTIFY_indication =
display_MLME_BEACON_NOTIFY_indication;
        handler.MLME_START_confirm=display_MLME_START_confirm;
               handler.MLME_ASSOCIATE_indication=ASSOCIATE_indication;
                               handler.unknown_primitive = display_unknown_primitive;
               for (;;)
               {
                      mac_receive (&handler, fd);
               }
            close (fd);

        } else {
                fprintf (stderr, "Failed to open %s.\n%s\n", device, strerror (errno));
                exit (EXIT_FAILURE);
        }

        return 0;
}
```

Appendix I – Vitas

Name:	Marc Soiferman
Place of Birth:	Winnipeg, Manitoba, Canada
Year of Birth:	1985
Secondary Education:	Oak Park High School (1999–2003)
Honours & Awards:	University of Manitoba Queen Elizabeth II Entrance Scholarship (2003)
	Dean’s Honour List (2004, 2005, 2006)
	UMSU Scholarship (2005)
	UMSU Scholarship (2006)
	APEGM Scholarship (2006)

Name:	Andy Tang
Place of Birth:	Winnipeg, Manitoba, Canada
Year of Birth:	1985
Secondary Education:	Kildonan East Collegiate (1999-2003)
Honours & Awards:	U of M Entrance Scholarship (2003)