Controlling Electrical Appliances using Bluetooth & J2ME-enabled Mobile Phone

By

Redza Shafique Md. Ridzuan

Final Draft submitted in partial fulfillment of
requirement for the
Bachelor of Technology (Hons)
Business Information System

JANUARY 2007

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Controlling Electrical Appliances using Bluetooth & J2ME-enabled Mobile Phone

By

Redza Shafique Md. Ridzuan

A project dissertation submitted to the
Business Information System Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
Bachelor of Technology (Hons)
Business Information System

Approved by,

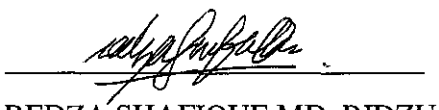
_____

(EN. MOHAMMAD NOOR IBRAHIM)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
January 2007

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



REDZA SHAFIQUE MD. RIDZUAN

ABSTRACT

This project focuses on the development of a universal remote control system that utilizes the use of Bluetooth and Java technology on mobile phones in controlling electrical appliances. The remote control does not confine to the physical barriers that are normally found at home like the typical IrDA remote controls. The system suggests the usage of Bluetooth technology in order to solve and expand the capability of the IrDA remote control technology that is still being widely used. This is also due to the fact that almost all electrical devices and appliances come with their own proprietary remote control. By having a universal remote control, it is possible to eliminate the need for such many remote controls at home. From the client application point of view, the user interface is constructed using Java 2 Mobile Edition as it is being supported by many mobile phones instead of other of different architectures. From the mobile phone, users are able to connect to the Bluetooth server that then interacts with the circuit where the electronic components reside. Immediate direct manipulation of the circuit is crucial as remote control system is one of the real time systems and it needs to be updated as soon as the state of appliances is modified. As a result, this project was developed using Bluez in Linux, for the server part; and Nokia 6230 Series 40 cell phone for the client application. In a nutshell, the end product gives flexibility to the users by promoting the usage of their mobile phones as a universal remote control for their electrical appliances.

ACKNOWLEDGEMENT

First and foremost, the author would like to express his gratitude to the almighty God for giving the strength to complete the Final Year Project based on his interest. He would like to take this opportunity to thank the people who are involved in this project directly and indirectly. Through out these past two semesters, a lot of people have been met for consultation in order to improve the understanding towards the project. Without the guidance and support from this special group of people, it is impossible for him to complete his final mission as a student of University Teknologi PETRONAS. It is a great pleasure for the author to take this opportunity to express his deepest gratitude to his supervisors, En. Mohammad Noor Ibrahim and En. Khairul Shafee Kalid for their continuous support, guidance and faith in him these two semesters. Next is to his Electrical and Electronic Engineering friends, Wan Muhammad Nasiruddin Wan Noordin, Anas Tajuddin and Syam Syairatul Husna Serilanang who have helped him to understand the electronic aspects of this project. Besides that, he would like to also thank Mohd. Izhar Firdaus Ismail for his guidance in using Linux as a way of life. Thank you to everyone who has helped through tough times, shared their thoughts and supported the writer from behind all the while. Last but not least, he would like to pay a special gratitude to his family members for their support and blessings. Thank you.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

This chapter contains brief information of the project which includes the background study of this project, problem statements that lead to the design of this project, its objectives as well as the project scope.

1.1 Background of Study

Remote controls have been around for quite sometime now. Through out the years, the usage of remote control is vastly expanded as well as the technology behind it. Now, almost all electrical appliances come with their own set of remote controls. Because of that, the consumers are bombarded with remote controls for their televisions, audio systems, air-conditioners, security systems and even toys. Therefore, instead of having many modular remote controls that only control one device, having a universal remote control would be a great help in reducing the amount of remote controls to be confused with.

This project is about turning a Bluetooth-enabled mobile phone into a universal remote control that able to control electrical appliances. One of the great advantages in Bluetooth lies in the huge support in all kind of devices. It is supported in USB dongles to ordinary PCs, PDAs like the iPAQ, cell phones and other embedded devices. This application requires the user to connect to a Bluetooth server using their mobile phone, and manipulate the switches logic from on state to off state, and vice-versa. At this moment, piconet topology is being used for the project. The server and client are assigned with a unique UUID that allows communication between them.

By having this system implemented in households, mobile phone owners are able to expand their phone capabilities by having to control as much devices and appliances as possible. This will then reduce the need of having one remote control for every electrical appliance available in the house.

1.2 Problem Statement

1.2.1 Problem Identification

1.2.1.1 IrDA devices are expected to be in the line-of-sight.

To date, there are still a huge number of remote controls that are still using infra-red (IrDA) technology. However, IrDA remote controls have their own share of limitations. As suggested by the characteristics of infra-red, IrDA devices are expected to be in the line-of-sight of those devices that are communicating with each other. This holds great disadvantages especially when there are obstacles in within the line of sight of the related devices.

In order to tackle that problem, Bluetooth has been the most ideal technology to replace IrDA as it makes use of radio frequency which is not limited by physical barriers. Devices will be able to communicate with each other even behind walls or other nonmetal objects besides having the ability to communicate wirelessly with up to eight devices in the range of 10 meters to 100 meters. The mobility features of Bluetooth give users the real-time access to house domain. They can easily access their house network domain securely as well as not to be bothered with messing up with devices from other domain.

1.2.1.2 Proprietary remote control

The absence of universal remote control in the market increases the number of remote controls per household. Each and every electrical device nowadays is bundled together with its own set of remote controls. This results in many remote controls residing in the house which then will lead to misplacing and confusion.

Hence, this project acts as a milestone in realizing the concept of having universal remote control to operate as much as electrical appliances as possible. Equipped with the

ability to connect 8 devices through Bluetooth simultaneously, Bluetooth-enabled mobile phones will be made capable to act as the standard remote control. It is also very flexible as every mobile phone that is equipped with the Bluetooth technology will be able to connect into the house domain and take control of the electrical appliances. This will help the users in not to face problems such as misplacing their remote controls or getting confused on which remote control that operates a particular appliance.

1.2.1.3 Aid for physically-challenged people

Another matter that should be looked into is that the idea of home automation is starting to take place worldwide. The idea of home automation does not only focus on giving normal people the freedom to control electrical devices for their own convenient, but it also focus on giving aid to the elderly, disabled and invalids to live their life with ease. To an invalid, the average staircase can present a formidable obstacle. It is also very troublesome and tiring for them to operate devices and appliances as normal people do.

Thus, it is vital to assist them as much as possible. By having a remote control, movements can be reduced and this special group of people will not have to go through many obstacles in life.

1.2.2 Significant of the Project

This project is designed to solve the limitations of using Infrared enabled remote control. By having a Bluetooth-enabled remote control, physical obstacles will not limit the users' range of control. They are able to control their appliances and devices almost anywhere in within the domain range which is only being limited by the Bluetooth range, of the respected classes. With the proposed system, such scenario as switching off the lights, turning on the air-conditioner and switching on the television to watch movie can be done by a touch of buttons on their mobile phones. On the other hand, people with limited movements will be able to operate things better with less supervision of others. The

proposed system that is powered by Bluetooth devices will definitely gives the stated advantages in homes and living places.

1.3 Objective

The objectives of the project are as below:

- To design a Bluetooth-enabled architecture that is able to support a number of devices.
    - To expand the capabilities and eliminate the limitations of current infrared remote controls.
- To develop a standard or universal remote control system to control electrical appliances in within a small domain.
- To design a front end system that can easily be installed in mobile phones.

1.4 Scope of Study

The scope of study to be done in this project is focused more on using a J2ME-enabled mobile phone with JSR-82 compatibility and Linux operating system to maintain open source architecture as much as possible. The system will however make use of LEDs, a buzzer and miniature fans to mimic the real-size electrical appliances. The system will also be coded using Java language with the support of their J2SE and J2ME platform.

1.5 Relevancy of the Project

Currently, the market of home automation in Malaysia is still at the beginning level. There are not many companies are willing to venture in such services. In realizing the state of Perak's K-Perak 2010 vision, having such home brew application will help the locals to easily adopt the system and adapt their life to it. By having a universal remote control, the physical amount of remote control can be reduced, which also means that the

consumption of plastic materials can be reduced. Human wise, the end product of this project gives the invalids and elderly people to accomplish things by their own by at least reduce if not eliminate the hassles that they have been facing all this while.

The introduction of Bluetooth in this project as the wireless medium of communication will address the limitations of Infrared remote controls. This will definitely give users more flexibility in operating things using the remote control. By using the Bluetooth technology, the implementation cost of having a universal remote control system will also be reduced as well.

1.6 Feasibility of Project within the Scope and Time Frame

This project was being developed within a specified time frame given. There were limitations and constraints faced throughout the development of this project. However, the main goal to achieve at the end of this project is to create a prototype of a Bluetooth remote control system that will utilize the Java programming language in such open source environment.

CHAPTER 2
LITERATURE REVIEW

This chapter contains the previous and existing studies; and applications that are related to this project.

2.1 Bluetooth

Bluetooth is an industrial specification for wireless personal area networks (PANs), also known as IEEE 802.15.1. It is a short-range communications technology intended to replace the cables connecting portable and/or fixed devices while maintaining high levels of security. The key features of Bluetooth technology are robustness, low power, and low cost. The Bluetooth specification defines a uniform structure for a wide range of devices such as mobile phones, laptops and Personal Digital Assistant (PDA)s to connect and communicate with each other.

Bluetooth uses omni directional wireless transmission of both voice and data in the 2.4 GHz Industrial, Scientific and Medical (ISM) band. The operating range depends on the device class:

- Class 3 – have a range of up to 1 meter or 3 feet
- Class 2 – have a range of 10 meters or 30 feet
- Class 1 – have a range of 100 meters or 300 feet

Bluetooth uses a flexible, multiple piconet structure for communication. It supports both point-to-point and multipoint connections for full-duplex networks. Currently up to seven slave devices can be configured to use a master radio in one device. Several of the piconets can be established and linked in scatternets to allow flexibility among configurations. Devices in the same piconet have priority synchronizations, but other devices can enter the network at any time. In a full-duplex network, a multiple piconet structure with 10 fully loaded, independent piconets, can maintain aggregate data transfer

speeds of up to 6 Mbps. Bluetooth piconet and scatternet topologies are illustrated in Figure 2.1.

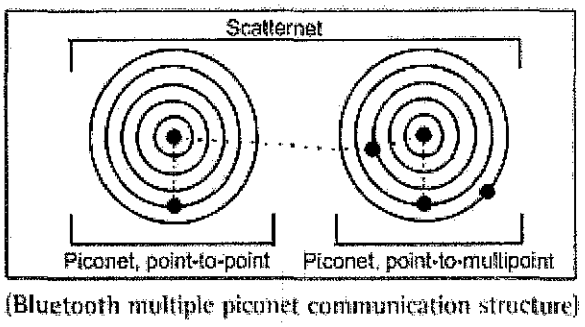


Figure 2.1: Bluetooth communication structure
(Source: Courtesy of Quatech)

This project demonstrated the use of Bluetooth communication piconet structure in allowing electrical appliances to be controlled using Bluetooth-enabled mobile phone. As Bluetooth-enabled mobile phones are now available in the market, they are definitely the best device to be used as a remote control as people tend to cling on them almost everywhere they go. The system will require the users to pair their Bluetooth-enabled mobile phone with Bluetooth-enabled server for authentication purposes. This will act as one sense of security measurement for Bluetooth connectivity where similar pin number is required to pair both devices up. When the authentication is successful, users are then able to control their electrical appliances via the system interface installed in their mobile phones.

2.2 Home Automation

As the word suggest, Home Automation is automating our home. According to Kwang et. al (2003), Home automation is a house or living environment that contains the technology to allow devices and systems to be controlled automatically. From lighting and climate control to home cinema and video surveillance, a growing range of home networking, remote control and automation technologies promise unprecedented control at our fingertips. Figure 2.2 below illustrates how a basic home automation concept works.

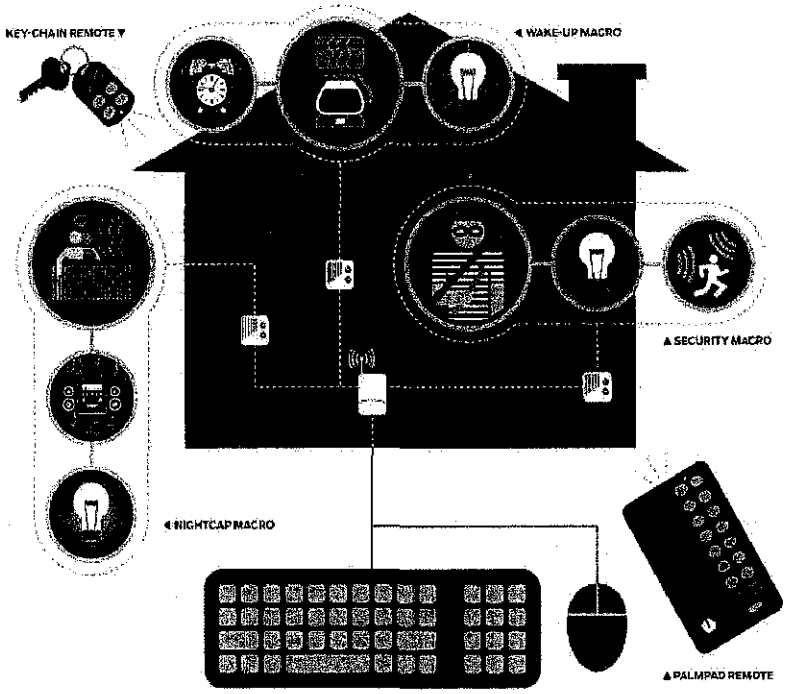


Figure 2.2: Basic Home Automation

(Source: Popular Mechanics, September 2005)

However, Dodge (2006) stated that home automation is more than glorified light-dimming via remote control, promising to integrate control of security, lighting, HVAC, energy, entertainment and appliances. For instance, controllers can automatically raise or lower blinds depending on the temperature or turn off the lights in a vacation home from a thousand miles away. People nowadays are indeed looking for such services as it is proven by a statistic done by Consumer Electronic Association in the United States as shown in Figure 2.3.

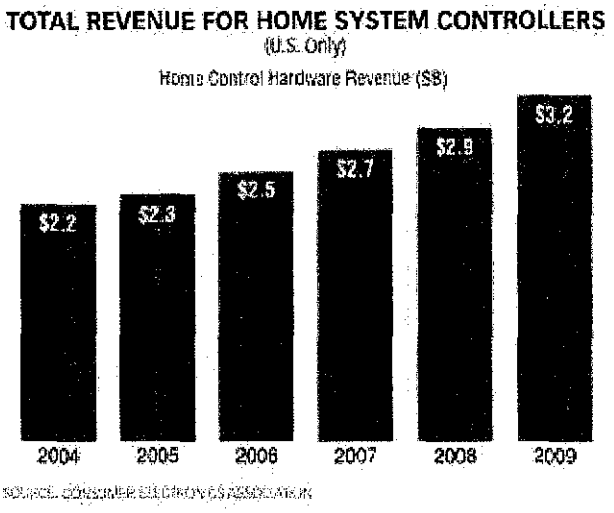


Figure 2.3: Total Revenue for Home System Controllers

(Source: Consumer Electronics Association)

There are three major reasons for people to invest time and money in a home automation setup. The first is convenience; the second is security and safety; the third is energy savings. By having such system, it will be much easier for users to control the appliances residing in their home especially to the elderly and physically challenged group of people. On the security front, homes get broken into when they're empty, and they can get vandalized. As for energy savings, these systems can certainly help. The house can be divided into zones and the electricity on each zone can be controlled appropriately, for instance.

2.2.1 Home Automation Technologies

2.2.1.1 X10

According to Wikipedia, **X10** that is developed back in 1975 is an industry standard for communication among devices used for home automation. Using conventional home electrical wiring, X10 transmits digital packets through up to 256 compatible devices on a single power circuit. A control centre for an X10 system may be a standalone hardware unit or a PC running Linux or Windows. Remote controls and keypads can also be used

to control light dimmers, TVs, VCRs, security alarms, door locks and surveillance equipment.

With no specialized wiring required, X10 is an affordable and reliable beginning to any digital home project. An X10 starter kit usually contains PC and hand-held remote controls as well as modules for two appliances, two lamps and a ceiling-mounted light. To use a module, users simply plug it into a power socket and then plug the appliance into the module. The appliance is then controlled via an infrared remote control or a power-point control module such as a keypad or PC adapter. Once the software is installed, the system can be run from a Windows, Mac, Linux, OS/2 or Amiga computer.

2.2.1.2 Universal Plug and Play (UPnP)

In an attempt to create "intelligent" appliances that communicate via a home network, Microsoft launched their Universal Plug and Play (UPnP) technology in 1999. UPnP offers Ethernet connectivity to household appliances. These can range from lighting dimmers and climate control systems to security and audiovisual appliances. This means, in theory, that any networked Windows PC can control UPnP devices around the home. It also means that users may eventually have remote access to do things like record a TV show on their home VCR via a Web browser.

Although slow to be adopted by manufacturers, some recent UPnP devices such as standalone media players have emerged. Version 2 of the UPnP protocol is in development. This is expected to be more widely implemented than its predecessor by incorporating support for technologies such as IPv6 and .NET services.

2.2.1.3 xAP

Likened by some to UPnP, but with a smaller overhead, xAP is a network protocol designed to be independent of operating system and programming language. Although available to any mode of transmission, it is currently only implemented via serial port or Ethernet connections. The goal of xAP is to provide interconnectivity between all

household devices including lights, telephones, Hi-Fi units, heating systems and computers. Although in its infancy, xAP has a dedicated developer community and may emerge as a contender in the future of home automation.

2.2.1.4 Jini

Sun Microsystems' Jini technology can network any device with a Java Virtual Machine over Ethernet, Firewire or HomeRF (a proprietary radio frequency wireless networking technology). Although it has the backing of vendors such as Sony and Philips, Jini remains within the realm of Java purists and programmers prepared to build their own interfaces and hand-code appliance drivers.

2.3 Wireless Technology

According to Wacker et. al (2004), existing approaches for home automation can be divided in two categories according to the communication media used. It can be done based on wired networks and on wireless networks. Since wireless networking started to be accepted globally, it is fair to have a glance on the wireless technologies that are currently available in the industry.

2.3.1 802.11

The group of standards known collectively as 802.11, Wi-Fi, or more broadly as "wireless", offer Ethernet connectivity to computers equipped with wireless network cards or WNICs. Generally, wireless is used to connect notebooks to existing wired networks via access points or specialised routers, although ad-hoc networks can be formed between multiple computers with WNICs. Most wireless Ethernet occupies the same 2.4GHz spectrum as cordless devices (excluding 802.11a) and can be used by compatible devices for digital home networking. The bandwidth of wireless networks ranges from 11Mbps (802.11b) to 54Mbps (802.11a and 802.11g). Although this is

slower than the 100Mbps available to most wired networks, it is fast enough for Web browsing and streaming multimedia content. Some proprietary standards have also emerged that can double these speeds to up to 108Mbps.

2.3.2 Ultrawideband

Ultrawideband (UWB) is a wireless technology that provides a maximum throughput of 1 gigabit per second. Like 802.11, UWB uses radio frequencies but in a much wider spectrum than the 2.4GHz range used by conventional wireless and cordless devices. As these frequencies are restricted by most international authorities, the future of UWB is uncertain. If the fears of UWB interfering with existing systems are overcome, then it may emerge as the wireless technology of the future.

2.3.3 HomeRF

HomeRF is a wireless networking standard developed by a working group of vendors in 1998. Version 2.0 is capable of speeds up to 20Mbps. As it offers similar capabilities within the same frequency range as 802.11b, HomeRF was once considered a competitor to Wi-Fi. With the emergence of Wi-Fi speeds of 56Mbps and greater, however, HomeRF is largely a redundant technology.

2.3.4 Bluetooth

Bluetooth is another radio frequency technology used by compatible computers, mobile phones and other handheld appliances for data transfer. Although capable of 723kbps over a range of up to 10m, Bluetooth is intended to be a short range and low speed standard to connect devices. While not suitable as the centrepiece of a digital home, Bluetooth does provide the option of using a PDA or mobile phone to interact with computer-controlled devices over a wireless or wired Ethernet network.

2.3.5 IrDA

IrDA is an infra-red technology for transferring data between devices such as laptops, PDAs and digital cameras. It is very short range but offers speeds up to 1.152 Mbps (in version 1.1). Like Bluetooth, it is not particularly useful as a basis for networking, but does offer remote control possibilities to networked devices.

2.3.6 Z-Wave

Z-Wave is a proprietary wireless technology that is like a radio frequency version of X10. Using inexpensive RF enabled devices attached to appliances around the home, Z-Wave can be used to automate lighting, thermostats, alarms and other mains power operated appliances. The latest version includes support for UPnP and .NET extensibility, making it a candidate for an integrated digital home system.

2.3.7 Readywire

Similar to X10, Readywire is capable of delivering home automation controls to up to 250 devices in 15 domains. Additionally, the USB adapter is equipped with an ARM 946 processor and up to 8MB of RAM. This delivers 168bit encryption to safeguard appliances from hijacking. It can also drive 15 near-CD quality audio streams and seven full-duplex voice lines using standard power lines.

2.4 J2ME

Java 2 Micro Edition (J2ME) platform will be use as it consists of a set of technologies and specifications developed for small devices like PDAs, mobile phones, pagers and other consumer electric and embedded devices. The other Java programming environments are Java 2 Enterprise Edition (J2EE platform), Java 2 Standard Edition (J2SE) and Java Card (for SIM cards). Below is the figure of Java end-to-end review

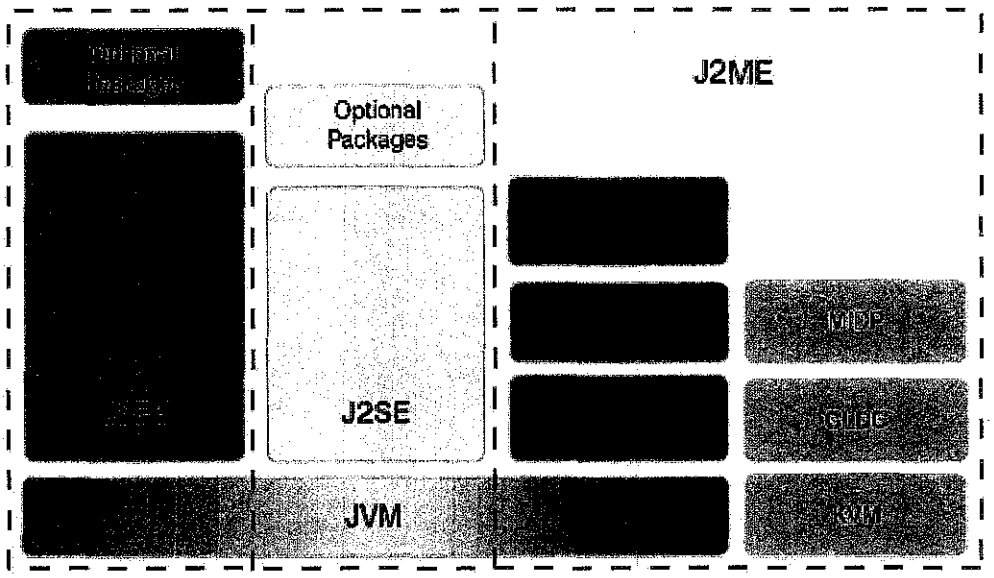


Figure 2.4: The relationship between J2EE, J2SE, and J2ME
(Source: IBM, 2001)

The reason of using the Java platform for wireless device development is that we are able to produce portable code that can run on multiple platforms. Java is one of the most widely used programming languages in the world. It is particularly appropriate for computers implementing internet-based and intranet-based applications and any other software for devices that communicate over the networks, including cell phones, pagers and PDA (Knudsen, 2003)

2.5 Parallel Port Interface

Parallel port is also known by its technical name, DB25, is a 25 pin male or female connector. Engdahl (2003) mentioned that the PC parallel port can be very useful I/O channel for connecting circuits to PC. The simplicity and ease of programming makes parallel port popular in electronics hobbyist world. A parallel port carries one bit on pin that serves particular purposes as shown in Figure 2.5. With this feature, it is possible to control more than 1 device with each parallel port connector.

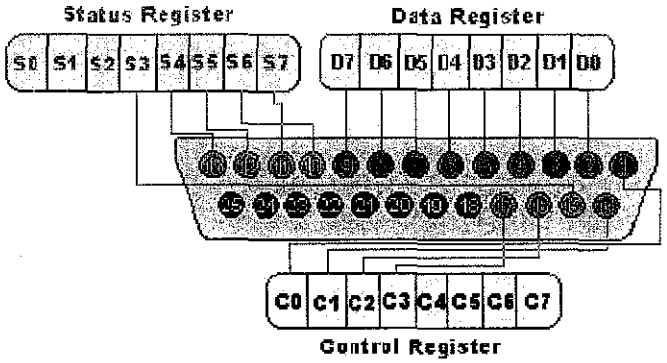


Figure 2.5: DB25 Female Connector

The pins on a parallel port are group into three registers or purposes which are

1) Status Register
2) Data Register
3) Control Register

The pins that are under the data registers, pins 2 to 9 are the ones to be manipulated in order to send I/O commands to the circuit. The other two registers are not allowed to be directly manipulated by users. Pin 18 to 25 serve the purpose as grounding for the circuit. Table 2.1 below shows the detail function for all pins.

Pin No (DB25)	Signal name	Direction	Register - bit	Inverted
1	nStrobe	Out	Control-0	Yes
2	Data0	In/Out	Data-0	No
3	Data1	In/Out	Data-1	No
4	Data2	In/Out	Data-2	No
5	Data3	In/Out	Data-3	No
6	Data4	In/Out	Data-4	No
7	Data5	In/Out	Data-5	No
8	Data6	In/Out	Data-6	No
9	Data7	In/Out	Data-7	No
10	nAck	In	Status-6	No
11	Busy	In	Status-7	Yes
12	Paper-Out	In	Status-5	No
13	Select	In	Status-4	No
14	Linefeed	Out	Control-1	Yes
15	nError	In	Status-3	No
16	nInitialize	Out	Control-2	No
17	nSelect-Printer	Out	Control-3	Yes
18-25	Ground	-	-	-

Table 2.1: Parallel Port Signal Lines

2.6 Review of Related Projects

A project done by Herman Shee (2005) entitled *The Development of a System to Control Electrical Appliances through Bluetooth-enabled Device* was designed to be a prototype of such new idea of home automation. The project focuses on replacing the legacy technology of remote control which is the Infrared technology with the Bluetooth technology. The mobile application utilized the use of Java 2 Micro Edition (J2ME) on a Nokia Series 60 phone. The project was successful with such native interfaces for both server and client applications.

However, the project was conducted by manipulating a single lamp. The assumption made was controlling a single electrical appliance will prove that other appliances are able to be controlled as well by expanding the circuitry. Besides that, the mobile application was not user-friendly enough for the users. It was mainly because of the project's initial objective which is proving the concept of Bluetooth remote control. As for the operating system, the author was using Windows XP SP2 to host the Bluetooth server. Communicating directly to the I/O port is not fully recommended by Microsoft in that release. Thus, the author needed to run couples of other applications prior to running the Bluetooth server. This was to allow the server to send and receive I/O command to and status from the parallel port respectively.

The project also indirectly suggested the use of built-in parallel port available on CPUs or laptops instead of using expansion slots such as PCI cards. This is due to the port base address constraint found in UserPort software, an application that is needed to be run in prior to the Bluetooth server. This reduces the convenience towards users who do have any built-in parallel port on their system.

Chakrabarti et. al (2004) presented a project titled *A Remotely Controlled Bluetooth Enabled Environment* which introduced another way of remotely controlling devices with the use of Bluetooth and web page. The webpage was design using Java language and contains the list of appliances that can be controlled through it. The

webpage was also able to provide the current state of those appliances back to the users. The project also suggested the usage of Bluetooth chips on separate devices for them to be able to be linked in Scatternet form.

Since the project uses webpage to control the appliances, it jeopardizes the fact of being mobile for the users to use it. It means that, users are expected to control their appliances from in front of the computer. Other than that, the usage of Bluetooth chip on every device definitely is not cost effective from the users' point of view. Users will have to bear with the high cost of implementation if they decide on adopting this system.

A project done by Edlington (2005) entitled *X10 Appliances Control Using Mobile Phone* was designed to make home automation easy to control when a user is not at home. By using mobile phones, users can control the system using SMS, WAP or other mobile development technologies. These technologies will be the intermediary between the users and the PC which in turn controls the devices attached to X10 modules. The PC will then return the status of every remote request that the user has made.

Having this kind of approach is not really efficient and user-friendly as it will burden the mobile phone users through the SMS charges besides having to spend more in buying the X10 products adapters.

Hal Furton (2001) took a different approach on home automation as his project codenamed "Domo" was designed for Linux using Ruby and X10 and Slink-e products. The project has a bigger scope that consists of internet, multimedia, telephony, electrical appliances and thermo capabilities.

Other than facing similar barrier of using X10 products, this project suggested another barrier which is the discontinuation of Slink-e products. Nirvis Inc., the manufacturer of Slink-e product has stopped their sales as of 2002 due to patent issue. Those who have just decided to adopt Furton's method will not be able to do so. The

project also suggested on using IrDA technology on its remote control module as the Slink-e product only caters for IrDA communication.

CHAPTER 3
METHODOLOGY

In this chapter, the methodology that is used for the project will be described briefly. In consideration of the time frame given, Rapid Application Development was chosen for this project.

3.1 Procedure Identification

3.1.1 Research

To collect information regarding this project, the followings approach were taken:

- Literature reviews – Find out about home automation development and the technologies involved in them. Gather information regarding wireless technology on remote controls.
- Device assessment – Find out about the Bluetooth support on mobile phones. Evaluate the communication interface devices to determine which device is suitable to be used in the project.

3.1.2 Development Methodology

For the development of this project, RAD (Rapid Application Development) is used considering the time frame given is quite short. The RAD category that best suits this project would be Iterative Development. The development methodology will be represented in Figure 3.1 .

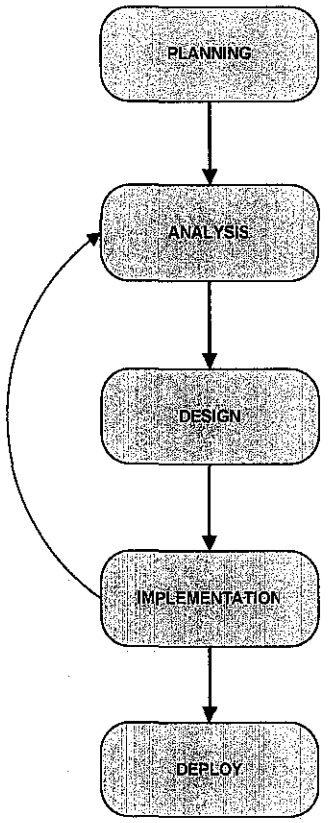


Figure 3.1: Iterative Methodology

The iterative methodology breaks the overall system into a series of versions that are developed sequentially. The analysis phase identifies the overall concept, and the project team, users and system sponsor then categorize the requirement into a series of versions. The most important and fundamental requirements are bundled into the first version of the system. The analysis phase then leads into design and implementation, but only with the set of requirements identified for version 1.

3.1.2.1 Planning

This stage was conducted during the early part of this semester. The activities done during this stage was focusing on brainstorming for the project. Students were given 3 weeks to decide on a topic and that was when they are expected to propose a project.

After some meetings with lecturers and doing some researches, the author decided on proceeding with home automation system using Bluetooth and Java technology. Problem identification has been made to determine the relevancy of the project and it has been discussed in Chapter 1.

3.1.2.2 Analysis

For the analysis phase, research was done to find out more about the case being studied which is the home automation concept. Some researches were done in determining the feasibility of using Bluetooth technology on mobile phone to act as a remote control. It is found that there are a lot of projects done related to home automation previously but none came close to reducing cost of implementation. Additional information has been added to the initial problem statements to ensure the to-be application shall be able to strengthen the currently available remote control application. The literature review has been discussed in Chapter 2 which consists of researches and reviews on projects related to this project. Devices that are used for this project were also being assessed. After some assessments, it was possible to develop the whole project under the open source architecture. However, all possible findings concerning the project are carefully reviewed to ensure the best solution is proposed.

3.1.2.3 Design

3.1.2.3.1 System Architecture Design

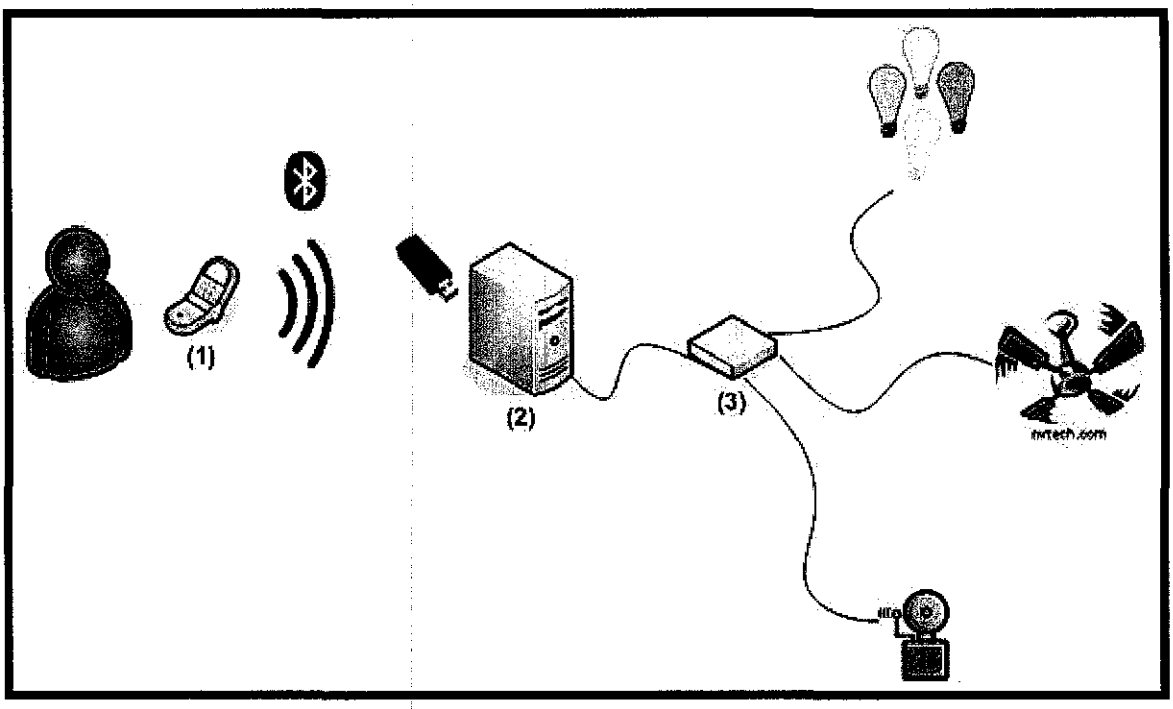


Figure 3.2: The Proposed System Architecture

(1) Mobile phone

- Nokia 6230 / 6680
- J2ME

(2) Bluetooth-enabled Server

- JDK 1.6.0
- Fedora Core 6
- AvetanaBT

(3) Circuit Interface

- Pspice

The proposed system architecture as shown in Figure 3.2 is the proposed architecture for this project. A J2ME application was installed in the mobile phone as the interface for the remote control. The mobile phone then communicates with the server using Bluetooth. The server which has a Bluetooth dongle receives the signal from the mobile phone and then control the parallel port pins via Java interface. Before that, it returns the status of pins back to the phone in order to let the users know the state of the appliances, whether it is in ON or OFF mode. When the user decides to switch the mode, the phone will send the command signal back to the server. Then, the server updates the parallel port pin with the new data bit and send it to the circuit interface. When the operation is done, the server the updates the status on the mobile phone and notify the user of the application about the status of the particular appliance.

3.1.2.3.2 Circuit Design

Designing the circuit was the first task being conducted. The designed circuit was consisting of a parallel port, and six LEDs, six resistors, a 5V fan and a buzzer. The parallel port acts as a communication interface between the PC and the circuit and it is used to control the voltage for all the electronic components. They put out ideally 0V when they are in low logic level (0) and +5V when they are in high logic level (1). When value 1 is being sent out to the data pin where the components are connected, that respective component will be turned on. When value 0 is being sent to that same pin, the component will be switched off. Figure 3.3 shows the schematics of the circuit.

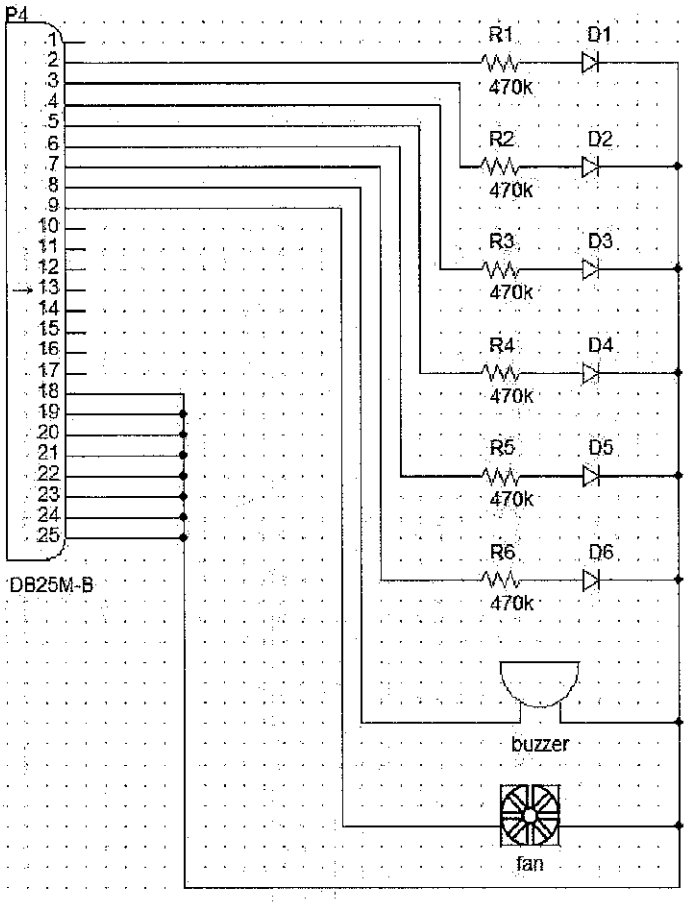


Figure 3.3: Circuit Schematics

3.1.2.3.3 Mobile Phone Interface Design

To enable the use of the program to control appliances in a particular room domain inside the house, the program needs to locate the Bluetooth room server first. To enable this, start up screen have been created to 'Locate Room' and provide a list of Room Server to choose on. User will then select which room to control. Once the room is chosen, the phone will attempt to connect with the computer using the Bluetooth connection on the phone. The current state of appliances is shown in the next screen. The next option is to switch and send the command to the computer server to change the state of the appliance. When the 'Switch' option is selected, a message will appear asking whether to allow the connection to Bluetooth and a simple confirmation needs to be selected. This type of message varies on different types of phones in which the program is installed.

3.1.2.3.4 Bluetooth Server Design

The server program will open the Bluetooth connection from the server program and wait for the client connection to connect. Once connected, the server will read the current state of the parallel port and send it to the mobile phone. After each time a command is received from the phone program, server will change the current state of the appliance whether to ON or OFF the appliance.

3.1.2.4 Implementation

The application part of the project was developed as well as the electronic circuit that holds the components together. The mobile application was written in Java language. As for the circuit, circuit design software was used to develop the overall circuit. Pspice is able to provide electronic components in diagram mode as well as able to test the circuit whether it is workable or not.

3.1.2.4.1 Circuit Development

This project utilizes all eight data pins on the parallel port. 470Ω resistors are being used to limit the current that flows into the LED, fan and buzzer. The fan and buzzer does not need a resistor because it can handle 5V that is being supplied to it.

3.1.2.4.2 Mobile Phone Interface Development

J2ME is the chosen language to deploy this part of the project. J2ME is minimal version of Java for mobile devices such as mobile phones or PDAs (Personal Digital Assistant). MIDP (Mobile Information Device Profile) applications written in J2ME are cross platform programs and will work on most modern phones. Because a program written in J2ME would be able to be used on most phones, the library is more limited to enable

compatibility. In order to smooth things up, Netbeans 5.0 and Wireless Toolkit are being used to manage application packages and emulators.

3.1.2.4.3 Bluetooth Server Development

The Bluetooth server was being developed using J2SE. This reduces the possibility of incompatibility issues between the PC and the mobile phone since the mobile phone. There are other Open-source applications that are being used for this server as well.

3.1.2.4.3.1 BlueZ stack

This is the official Bluetooth stack for Linux. The stack was initially developed by Max Krasnyansky at Qualcomm in 2001. Later Qualcomm decided to release it under the GPL open source license. It was then included as the official stack in the Linux kernel from the 2.4.6 release. Any recent kernels have the BlueZ stack build in, either as loadable modules or compiled into the kernel. BlueZ provides vast support for the entire core Bluetooth layers and protocols, like: L2CAP, RFCOMM, SDP, SCO, BNEP etc. It also ships with a large amount of tools and sample programs to test the Bluetooth equipment, which makes it easier for a new developer to make out new code from the small code fragments of the samples.

3.1.2.4.3.2 Parport

This is a piece of coding that was written by Juan Gabriel Del Cid Portillo in 2005. It is a Java class that enables applications to read and write bytes to and from the parallel ports on the computer. By using this, developer does not have to implement Java Comm API that is far more complex and tedious.

3.1.2.4.3.3 AvetanaBT

The software is based on the most widely spread Bluetooth protocol stacks and does not use special Bluetooth hardware or software. It allows programmers to easily use and offer Bluetooth services. Avetana Bluetooth is a Java JNI implementation of JSR-82 for J2SE and different stack implementations.

The implementation is actually quite platform independent and supports different stacks on various platforms:

- Widcomm (Windows)
- Apple System stack (Macintosh)
- BlueZ (Linux)

3.1.2.4.4 Testing

Testing is an initial phase to ensure that the end product is flawless during deployment. Thus, it was executed back to back in within the implementation phase. This task is being split into two more phases, which are the unit testing and system testing.

3.1.2.4.4.1 Unit Testing

Throughout the project, testing has been carried out on every part of the system as each part was implemented.

3.1.2.4.4.1.1 Circuit

Every electronic component has gone through independent testing that they work. After the complete circuit is all set, it is tested using 9V battery to make sure that the circuit works and the current goes through every component. Next, the printer port is assembled to the circuit and then being tested using a simple C language application. The final step

was to test the circuit using a simple Java application that utilizes Parport class made by Duan (2005).

3.1.2.4.4.1.2 Server

Bluetooth-enabled computer acting as the Bluetooth server should be able to establish the Bluetooth connection and waiting to be connected by the client. In order to test that, a J2SE application was created to initiate connection with the Bluetooth server, using another PC. This is because the server and client must not use the same Bluetooth device or in other words, executed in the same PC. Once the connection is able to establish, the same client preferences is being deployed into the mobile application.

3.1.2.4.4.2 System Testing

This phase of testing is to test the system integration between the server, client and circuitry. During this phase of testing, testing should be done after each unit testing has been performed. This is to make sure that what have been developed manage to integrate with each other to perform as one big system. Other than that, it is to make sure what have been debugged and repaired, do not break the whole system's functionality and performance.

3.2 Tools Required

3.2.1 Hardware

- J2ME-Bluetooth enabled mobile phone
- USB Bluetooth dongle
- Circuit Interface
- Parallel port interface card

3.2.2 Software

3.2.2.1 Operating System

3.2.2.1.1 Fedora Core 6

This distribution has been chosen to become the operating system since there is an active support from this university's students. There is currently a repository being set up to ease up the update process of this distribution.

3.2.2.2 Phone Application Development

3.2.2.2.1 Netbeans IDE 5.0 bundled with Netbeans Wireless Toolkit 5.0

This IDE assists developer in creating application packages for better management. Equipped with its own Wireless Toolkit for IDE compatibility purposes, it provides developers with the common J2ME emulator found in J2ME Wireless Toolkit pack from Sun Microsystem.

3.2.2.3 Bluetooth Server

3.2.2.3.1 BlueZ stack

It is the official Bluetooth stack for Linux distributions.

3.2.2.3.2 Java Development Kit 1.6.0

The JDK is a development environment for building applications, applets, and components using the Java programming language. It includes tools useful for developing

and testing programs written in the Java programming language and running on the Java platform.

3.2.2.3.3 Libparport.so component

ParallelPort is a simple Java class that enables reading and writing bytes to and from the parallel ports on the computer.

3.2.2.3.4 AvetanaBT

Open source implementation of the JSR-82 Bluetooth API for Java on Linux.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 System Logic Flow

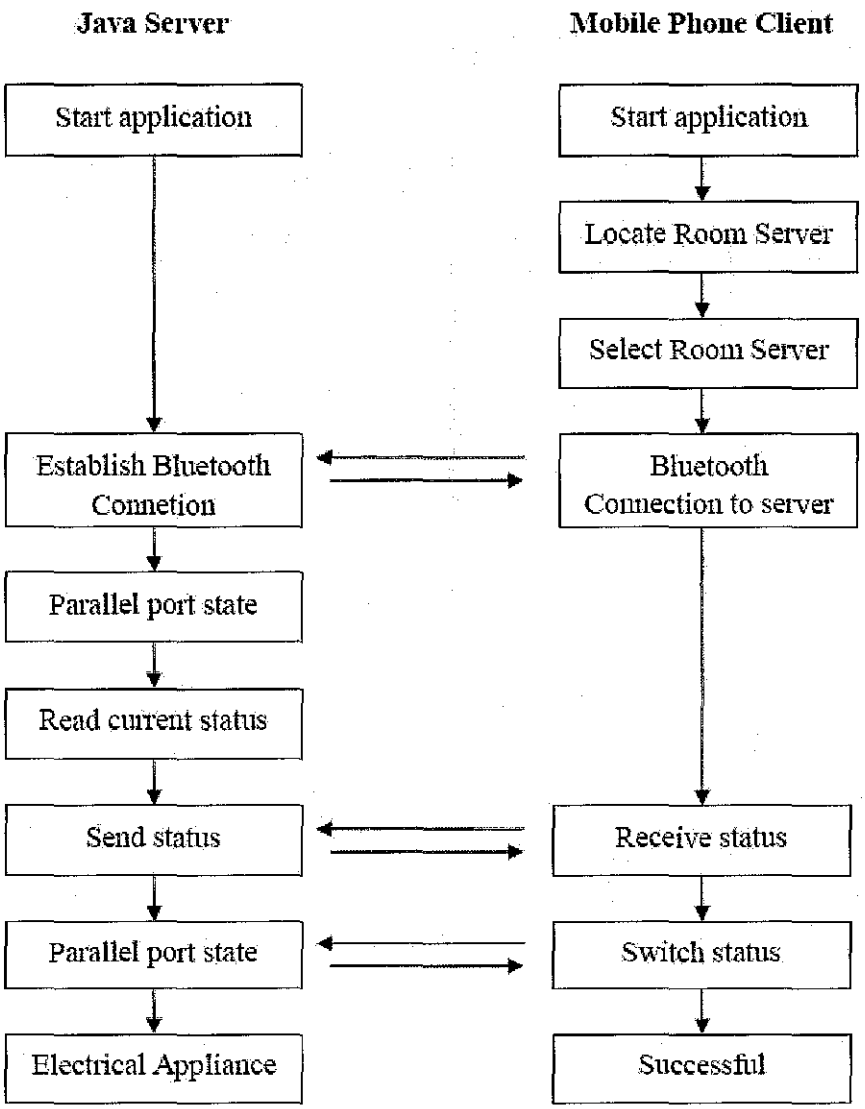


Figure 4.1: System Logic Flow

Figure 4.1 shows the overall flow of the communication process between server and client right from the start. The server needs to open the Bluetooth connection so that the

phone client will be able to locate and acquire the server to establish the connection as shown in Figure 4.2.

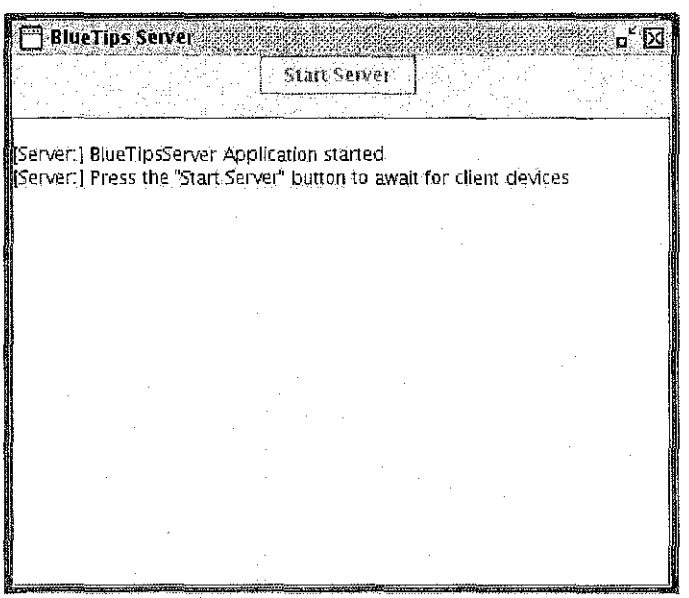


Figure 4.2: Server Interface - Waiting for Connection

Once the server is up and running, the mobile client should start searching for the server and establish a connection with the particular server that is available.

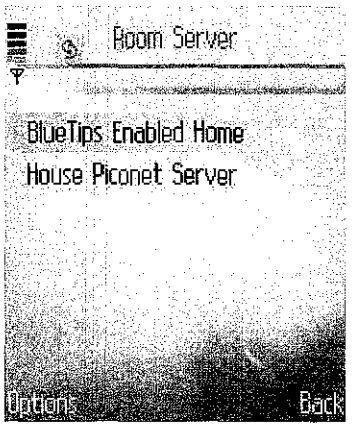


Figure 4.3: Mobile Interface – Room Server

Once the client established a connection with the server, the server will acknowledge the connection and start retrieving the parallel port status as shown in Figure 4.4.

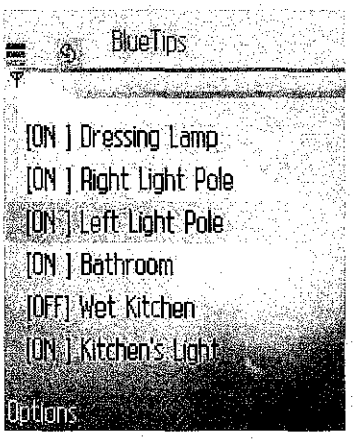


Figure 4.4: Mobile Interface - Appliance Status Retrieval

BlueTips server will receive the command to switch the state of the appliance from the mobile phone and run the command through the server program as in Figure 4.5 and Figure 4.6. After receiving the command, the server will then send it to the control unit via parallel port and switch the particular appliance. The state will remain until the next command is received to switch the state.

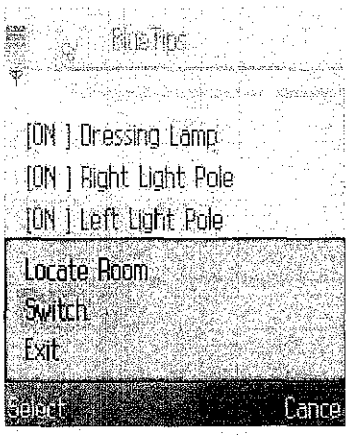


Figure 4.5: Mobile Interface – Appliance Status Switch

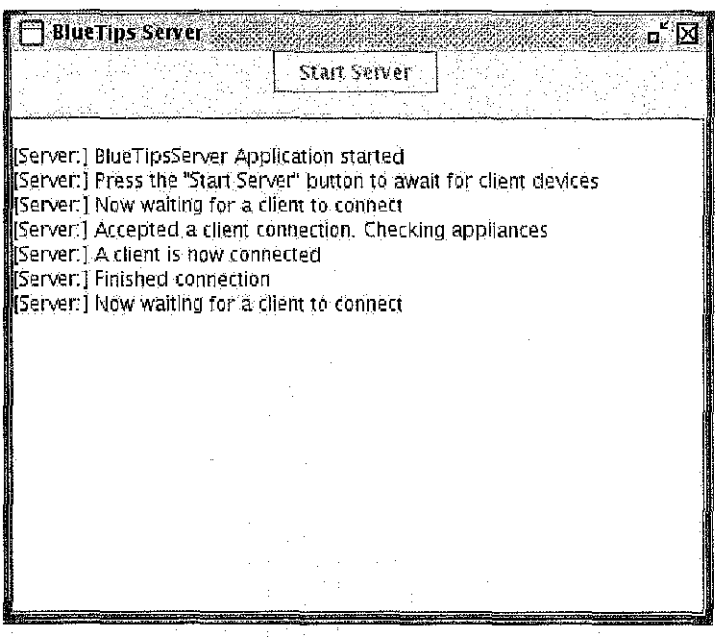


Figure 4.6: Server Interface – Execution of Status Update

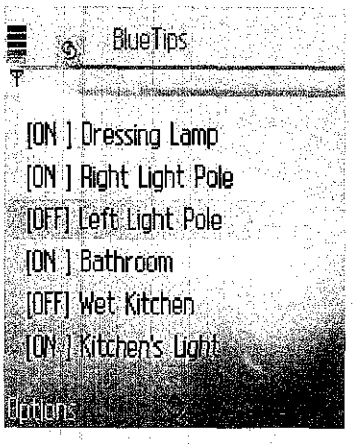


Figure 4.7: Mobile Interface – Appliance Status Updated

4.2 Circuit Realization

Currently, the circuit that has been developed is indeed a simple circuit which consists of a 25-pin D-shaped male connector, six LEDs, six 470Ω resistors, a 5V miniature fan and a buzzer. The resistors were assigned to limit the current that will flow through the LEDs. Bread board was used instead of a PCB to ease up the swap of components and debugging process.

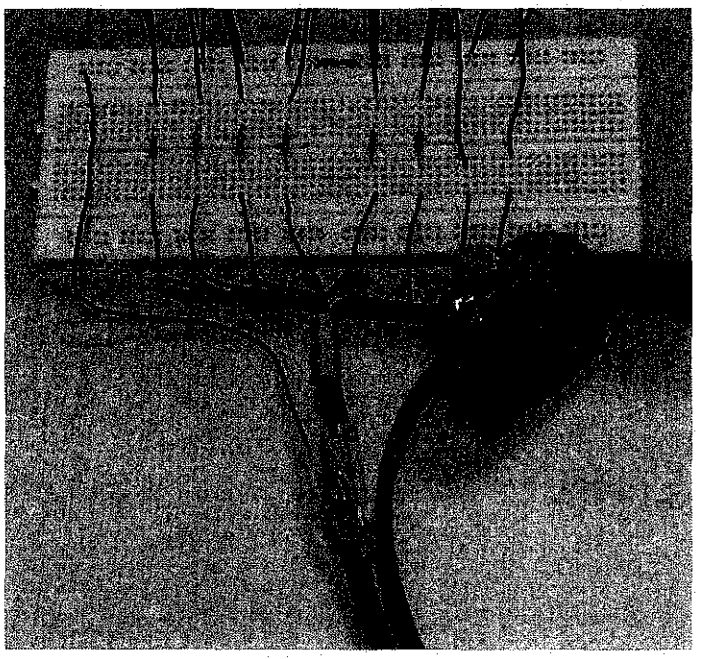
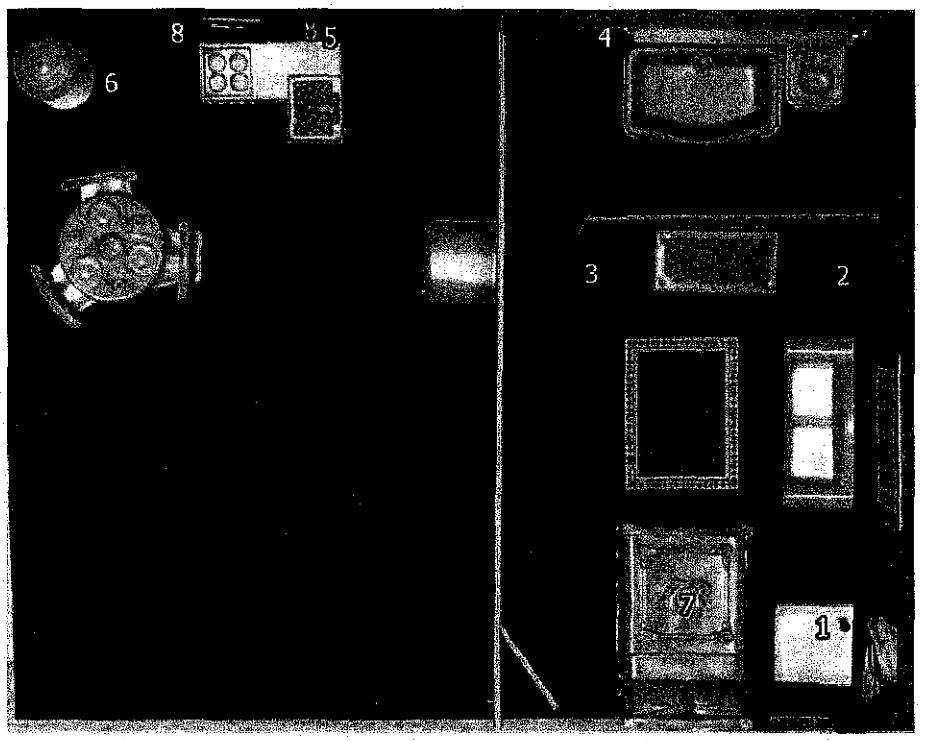


Figure 4.8: Product Image - Circuit View



1 : Dressing Lamp
2 : Right Lamp Pole
3 : Left Lamp Pole
4 : Bathroom
5 : Wet Kitchen
6 : Kitchen's Light
7 : Alarm
8 : Ventilator

Figure 4.9: Product Image – Home Model

4.3 Bluetooth Server Application

4.3.1 Parport IO Component

The following function is the sample of the written program using J2SE. Before we can use the function, we will first need to copy the file libparport.so file into the system directory.

```
public class ParallelPort {
   private int portBase;
   public ParallelPort (int portBase)
   {
      this.portBase = portBase;
   }
```

In order to write commands to the parallel port data pins, this function will be executed by accepting base 2 numbers.

```
public void write (int oneByte)
{ParallelPort.writeOneByte (this.portBase, oneByte);}
```

This command will be used load the libparport.so library to the system that has been developed.

```
{System.loadLibrary("parport");}
```

4.3.2 AvetanaBT

AvetanaBT needed to be patched in order to be used in any linux distributions that uses kernel 2.6 and later. For those distributions that come with new kernels, all distribution-related packaged have been migrated to the new library.However, some 3rd party software still refer to the old library such as this AvetanaBT. Upon compilation, the compiler will halt with an error caused by missing symbol "sdp_cstate_t".

In order to get AvetanaBT compiled, the below statement is needed to be inserted in the structs declaration of BlueZ.cpp source file which is located in the "c" directory of the distribution package.

```
typedef struct {
        uint8_t length;
        unsigned char data[16];
} __attribute__ ((packed)) sdp_cstate_t;
```

4.3.3 BlueTips Server

This J2SE application handles both Bluetooth communication between the mobile phone application and the computer; and the data transfer towards circuitry via parallel port. Both tasks must be declared and initialized as stated below.

```
//0xa400 is the printer port base address
private static ParallelPort lpt1 = new ParallelPort(0xa400);

// 11111 is the UUID being set in both server and client mobile
// application to allow Bluetooth communication
UUID uuid = new UUID("11111", true);
```

The next statements allow the server send the appliances current state by the mean of base 2 numbers to the mobile phone application.

```
DataOutputStream out = btConn.openDataOutputStream();
out.writeInt(x);
out.flush();
```

`processConnection()` function is to handle data stream or actions to be performed summoned by the mobile phone application.

```
void processConnection(StreamConnection conn) {
try {
                DataInputStream in = conn.openDataInputStream();
                try{
                        x = in.readInt();
                } catch (IOException ex) {
```

```
                    System.out.println("Unable to handle incoming
data");
                    ex.printStackTrace();
                }
                in.close();
.
.
.
}
```

4.4 Mobile Phone Application

As mentioned earlier in previous chapter, the Bluetooth server and client need to have the same set of UUID to allow communication between them. The UUID assignment is being done in SPP_Server class

```
public final static UUID uuid = new UUID("11111", true);
```

read_SPP_message() function in SPP_Client class is done to retrieve the appliance state sent from the server.

```
public void read_SPP_message(ServiceRecord r)
{
.
.
      con = (StreamConnection) Connector.open( url);
      // read command (current status) using input stream
      in = con.openDataInputStream();
      x=in.readInt();
.
.
}
```

Below is the portion of the function that converts command from the mobile application to the base 2 number before sending it to the server.

```
public int pow(int a, int b)
{
      .
      .
      for (int x=1;x<=power;x++)
      {
            value = value*2;
      }//end of for.
      .
      .
}
```

4.5 Project Cost

Listed below in Table 4.1 is all the costs related to deployment of the product for this project.

Components	Cost (RM)
Server	
Bluetooth Dongle	60
PCI I/O Card (Printer Port)	50
Circuit Components	
DB25 Male Connector	1.20
LED	0.10 x 8 = 0.80
470Ω Resistor	0.20 x 8 = 1.60
Bread Board	15
5V Fan	3
Buzzer	1.2
Total	**132.80**

Table 4.1: Cost of Project

4.6 Recommendations

4.6.1 Controlling Electrical Appliances

The circuit needs to be expanded in order to cater for real electrical appliances. Zayed (2004) has suggested such circuit in order to allow parallel port to control electrical appliances.

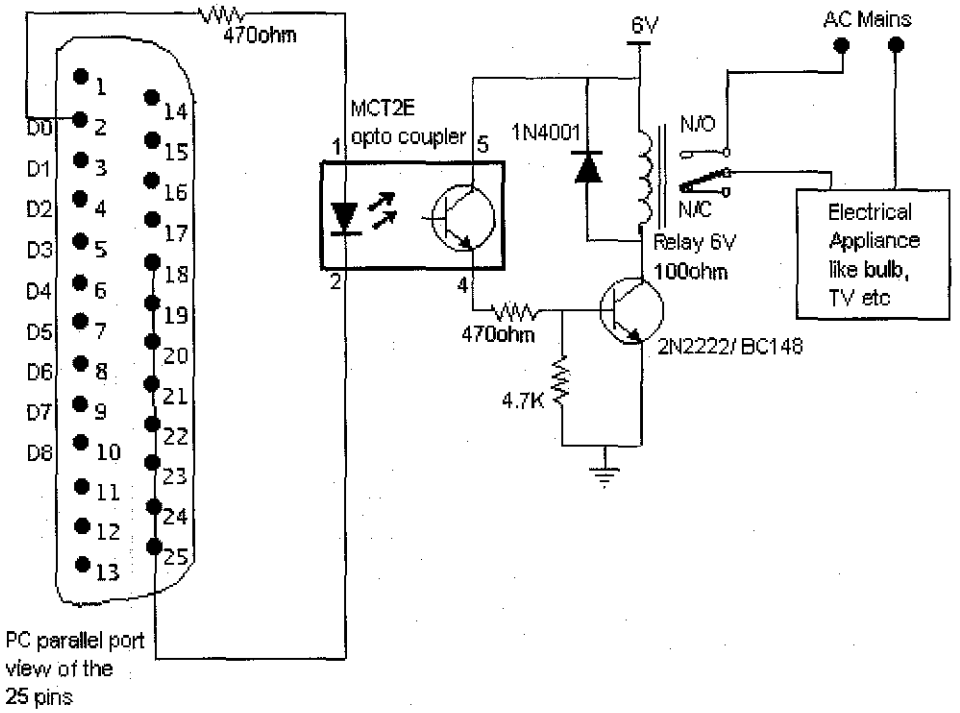


Figure 4.10: Electrical Appliance Circuit Interface Design

According to Herman Shee (2005), in order to supply the required voltage of 240V to switch the appliance, relay is needed to supply and extend those power supplied from the parallel port with the used of batteries. The opto-coupler is used to protect the port parallel. The opto-coupler's input is a light emitting diode. R1 is used to limit the current when the output from the port is on. That 1kohm resistor limits the current to around 3 mA, which is well sufficient for that output transistor driving. The output side of the opto-coupler is just like a transistor, with the collector at the top of the circuit and the emitter at the bottom.

When the output is turned on by the input light from the internal LED in the opto-coupler, current flows through the resistor and into the transistor, turning it on. This allows current to flow into the relay and current goes through R2 to the transistor base.On the other hand, when transistor is off, no current flows into the relay and switches off the circuit. The diode provides an outlet for the energy stored in the coil, preventing the relay from back feeding the circuit in an undesired manner.

The transistor in the circuit can be used for controlling output loads to maximum of around 100 mA. The circuit is powered from external power supply which is not connected to the PC. This arrangement prevents any currents on the external circuits from damaging the parallel port.

4.6.2 Multimedia features on PC

There a lot of possible things to be controlled prior to having a mobile phone connected to a server. Having a mobile phone that behave as a computer mouse has been made possible by countless of projects done in the University and out in the Internet. A mobile phone that is able to control presentation slides, handle MP3 playlists as well as turning the computer on and off makes the idea of universal remote control, more pleasing.

4.6.3 Client Application GUI

Currently, most part of the client application is text-based. The user interface can be further improved using a free patch, J2ME-Polish that is available at www.j2mepolish.org .J2ME Polish is a collection of tools for developing J2ME applications. This package can enhance a user interface (GUI), by designing it outside of the application code with simple text-files (CSS).

4.6.4 Server Support

For the server point of view, it could be interesting to see if it can be organized with a better object oriented design as for right now, it is a little bit messy. Despite already having a GUI for user to interact, it is believed that a better GUI will reduce the hassles of making the server to run active.

Besides that, for further improvements, the server should be able to handle multi connections from other devices. Although it is possible to connect more than one mobile

phone right now, those mobile phones were connected using the same set of UUID. This suggests security issues that need to be dealt with as it proves that other people are able to connect to the Bluetooth server if they practically have the similar client application.

4.6.5 Security

It is also recommended to embed security features on both server and client applications since having these new technologies at home provides many new ways for adversaries to invade an individual's personal life. Both applications can be equipped with a login page of some sort that acquire the users to key in their security codes other than their pairing pin codes. This can avoid unauthorized users to meddle with the application.

CHAPTER 5
CONCLUSION

The outcome of the project has proven that Bluetooth can be used to connect a mobile phone to the PC in within a restricted domain. Theoretically, the circuitry part needs to be expanded to change from electronic components to electrical appliances. LEDs, a miniature fan and a buzzer were used to resemble the real life appliances especially lamps, fans and security alarm.

J2ME enables Java application to run on small, resource-constrained computing devices. It has become a standard in current mobile phone developments that most mobile phone manufacturers bundle their mobile phones with J2ME support. The introduction of J2ME into this project enables the client application to be easily installed in various mobile phones. The client application for this project has been tested on Nokia 6230 (Series 40) and Nokia 6680 (Series 60) mobile phones.

Nowadays, Bluetooth can be found on mobile phones, PDAs, laptops and even PCs. Despite the focus of this project which is to implement piconet topology on a domain, during the testing phase, the mobile phone was able to detect both Bluetooth connections transmitted from various devices that were a linux-based PC, a Windows-based laptop, and a Sony Ericsson W700i mobile phone.

All in all, the project looks very promising. The Bluetooth technology seems to fit well into context and location-based applications like this, as proven in this project. It has been an interesting and fun project to work with Bluetooth and to build a small remote control application although it was quite time-consuming.

REFERENCE

Dodge, John. 2006, *Home Automation searches for the mainstream market.* Electronic Business

Kwang, Y.L. and Jae, W.C., 2003, *Remote-Controlled Home Automation System via Bluetooth Home Network.* Pusan National University, Korea .

Randall, N., 2006, "Home Automation Setups," *Smart Computing* (August 2006) 58-61

Curry, J.O., 2005, "Remote Possibilities," *Popular Mechanics*. 30 August 2006, <http://www.popularmechanics.com/technology/computers/1788752.html?page=1&c=y>

Wacker, A.; Heiber, T. and Cermann, H., 2004, *A key-distribution scheme for wireless home automation networks.* Universität Stuttgart, Stuttgart, Germany

Dennis, A.; Wixom, B.H. and Tegarden, D., 2002, *Systems Analysis & Design An Object-Oriented Approach with UML*, New York, John Wiley & Sons

Knudsen, J., 2003, *Wireless Java: Developing with J2ME, Second Edition*, California, Apress

Herman Shee, H.S, 2005, *The Development of a System to Control Electrical Appliances through Bluetooth-enabled Device*, Universiti Teknologi PETRONAS, Perak, Malaysia

Chakrabarti, S.; Wu, Liyun; Vuong, Son and Leung, V.C.M., 2004 *A Remotely Controlled Bluetooth Enabled Environment.* University of British Columbia, Vancouver, Canada

Quatech, *Bluetooth Communication Overview.* 10 August 2006, <http://www.quatech.com/support/comm-over-bluetooth.php>

Sundsted, T., 2001, *J2ME Grows Up.* 20 August 2006, <http://www.ibm.com/developerworks/java/library/j-j2me/>

Edlington, P., 2005, *X10 Appliance Control using Mobile Phone*, Lancaster University, United Kingdom

Del Cid Portillo, J.G., 10 January 2007, <http://www.geocities.com/Juanga69/parport/>

Zayed, T., 2004, *Control Electrical Appliances using PC.* 20 March 2007, <http://www.codeproject.com/csharp/control_e_appliances.asp?df=100&forumid=55160&exp=0&select=1286423>

http://www.bluetooth.com

http://en.wikipedia.org

APPENDICES

BlueTipsServer.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.microedition.io.*;
import java.io.*;
import parport.*;
import javax.bluetooth.*;

public class BlueTipsServer implements ActionListener, Runnable{
      // Bluetooth singleton object
      public static BlueTipsServer instance;
      private static ParallelPort lpt1 = new ParallelPort(0xa400);
      public String s;
      //StreamConnection btConn = null;
      public DataInputStream in;
      //initialize now;
      int x;
      int swap = 0;
      int firstRun = 0;
      LocalDevice device;
      DiscoveryAgent agent;
      String HTBTurl = null;
      Boolean mServerState = false; // stop is default state
      int statusValue = 255;
      Thread mServer = null;
      String msgOut = "srv out msg";
      String msgIn = "no msg rcv";
      StreamConnectionNotifier btServerNotifier;
      UUID uuid = new UUID("11111", true);

    JLabel spacerlabel = new JLabel(" ");

    JButton startButton = new JButton("Start Server");
    JTextArea textarea = new JTextArea("",20, 40);


    public BlueTipsServer(){


      instance = this ;
        //Give it the Java look and feel
        JFrame.setDefaultLookAndFeelDecorated(true);

        JFrame frame = new JFrame("BlueTips Server");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JScrollPane scrollPane = new JScrollPane(textarea);
        textarea.setEditable(false);

        Container cp = frame.getContentPane();
        cp.setLayout(new BoxLayout(cp, BoxLayout.Y_AXIS));

        startButton.setAlignmentX(Component.CENTER_ALIGNMENT);
        startButton.addActionListener(this);
                cp.add(startButton);

        spacerlabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        cp.add(spacerlabel);
```

```
spacerlabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        cp.add(spacerlabel);

        scrollPane.setAlignmentX(Component.CENTER_ALIGNMENT);
        cp.add(scrollPane);

        frame.pack();
        frame.setVisible(true);

        updateStatus("[Server:] BlueTipsServer Application started");
        updateStatus("[Server:] Press the \"Start Server\" button to
await for client devices");


    }

        private void startServer() {
            if (mServer !=null)
                return;
            //start the server and receiver
            mServer = new Thread(this);
            mServer.start();
        }

        private void endServer() {
            if (mServer == null)
                return;
            try {
                mServer.join();
            } catch (Exception ex) {};
            mServer = null;

        }


     // control flag for run loop
  // set true to exit loop
  public boolean done = false;

        public void run(){
        try {
            //the application is utilizing 11111 as its UUID
            UUID uuid = new UUID("11111", true);
                  device = LocalDevice.getLocalDevice(); // obtain
reference to singleton
            device.setDiscoverable(DiscoveryAgent.GIAC); // set
Discover mode to LIAC
            }catch (Exception e)
             { System.err.println("Cant init set discover");
               e.printStackTrace();
            }
            String url = "btspp://localhost:" + uuid +
";name=BTTP;authenticate=false;master=false;encrypt=false";

        try{

                  // obtain connection and stream to this service
                    btServerNotifier = (StreamConnectionNotifier)
Connector.open( url );

                } catch ( Exception e) {
                    e.printStackTrace();
                }
```

```
                while (mServerState )
                {

                     StreamConnection btConn = null;
                     try {
                                                updateStatus("[Server:] Now
waiting for a client to connect");
                                                btConn =
btServerNotifier.acceptAndOpen();
                                                updateStatus("[Server:]
Accepted a client connection. Checking appliances ");

                                                // retrieve the remote device
object
                                                RemoteDevice rdev =
RemoteDevice.getRemoteDevice( btConn );

                                                //open status log
                                                DataInputStream dis = new
DataInputStream(new FileInputStream("status.dat"));
                                                x = dis.readInt();
                                                dis.close();

                                                // sending appliances state
to client
                                                DataOutputStream out =
btConn.openDataOutputStream();
                                                out.writeInt(x);
                                                out.flush();

                                                //update the circuit
                                                lpt1.write(x) ;
                                   } catch (IOException ioe) {}

                                  if (btConn != null)
                                 processConnection(btConn);
                }//end of while
        }//end of run

        void processConnection(StreamConnection conn) {
         updateStatus("[Server:] A client is now connected");
         try {
                 DataInputStream in = conn.openDataInputStream();
                 try{
                         x = in.readInt();
                } catch (IOException ex) {
                    System.out.println("Unable to handle incoming
data");
                    ex.printStackTrace();
                }
                  in.close();
                  System.out.println("The receive message is '" + x +
"'");
                  //update circuit status
                  lpt1.write(x) ;
                } catch (Exception e)
                {
                  e.printStackTrace();
                }

                //save status of appliances
                try {
```

```
                        DataOutputStream dos = new DataOutputStream(new
FileOutputStream("status.dat"));
                        dos.writeInt(x);
                              dos.close();
                }catch(Exception e){}

                try {
                    conn.close();
                        updateStatus("[Server:] Finished connection");
                }catch (Exception e ){ }

        }//end of processConnection

    public void actionPerformed(ActionEvent e) {
                if ((e.getActionCommand()).equals("Start Server")    )
{
                    startButton.setEnabled(false);
                    mServerState = true; // set server state started
                    startServer();
                }//end of if
    }//end of actionPerformed

    public void updateStatus(String message){
        textarea.append("\n" + message);
    }//end of updateStatus


}//end of BlueTipsServer
```

SPP_Midlet.java

```
/*
* @author Redza Shafique Md. Ridzuan - (C)opyright 2007
* Version 1.0

* This is a project created for Final Year Project
* University of Technology Petronas
* requirement for the Degree in Technology (BIS)
*/
package btgallery.spp_gui;

import java.io.*;
import javax.bluetooth.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

import btgallery.bluelet.*;
import btgallery.spp_bt.*;

public class SPP_MIDlet extends MIDlet implements CommandListener
{
  public static SPP_MIDlet instance;
  public static Display display;
  public int pinArray[] = new int[8];
  public int tempPin;
  public int which;
  public int connCounter = 0;
  public ServiceRecord maintain;
  // GUI component/screen
  public SPP_Screen spp_screen = null;

  // GUI component/screen
  public RemoteDeviceUI remotedeviceui = null;
```

```
  // utilize Bluelet to do device disvcovery and service discovery
  BLUElet bluelet = null;
  // SPP client logic
  SPP_Client spp_client  = null;

  // SPP server logic
  SPP_Server spp_server = null;
  private Splash splash;

  public SPP_MIDlet()
  {
    instance = this;
    splash = new Splash(this);
  }

  /**
   * Implements MIDlet lifecycle
   */
  public void startApp()
  {
    display = Display.getDisplay(this);

    bluelet = new BLUElet( this, this );
    bluelet.startApp();

    spp_client = new SPP_Client();
    spp_server = new SPP_Server(); // server not yet started in
constructor
    spp_screen = new SPP_Screen();


    Display.getDisplay(this).setCurrent(splash);

    //start locate room server


    //showSplashScreen( display, spp_screen );
    //showSplashScreen( d, spp_screen );
    //display.setCurrent( spp_screen );
  }

//Activate Main Menu Displayable
      public void activateMainMenu()
      {
      bluelet.startInquiry( DiscoveryAgent.GIAC, new UUID[]{
SPP_Server.uuid } );
        display.setCurrent( bluelet.getUI() );
      }
  /**
   * Implements MIDlet lifecycle
   */
  public void pauseApp()
  {
    bluelet.pauseApp();
  }

  /**
   * Implements MIDlet lifecycle
   */
  public void destroyApp(boolean unconditional)
  {
    bluelet.destroyApp( unconditional );
  }
```

```
  /**
   * Exit MIDlet
   */
  public static void quitApp()
  {
    instance.destroyApp(true);
    instance.notifyDestroyed();
    instance = null;
  }

  /**
   * Handle user action and input.
   * All GUI Command are directed to this function.
   * @param c
   * @param d
   */
  public void commandAction(Command c, Displayable d)
  { if ( c.equals( BLUElet.COMPLETED ) )
    {
        display.setCurrent( spp_screen );

    // WTK2.2 beta 2 patch note:
    // - above solution works with WTK 2.2 beta 1, but doesn't work
with beta 2. it will hang the emulator
    // - below does work with WTK 2.2 beta 2, by using a background
thread to send the message
   Thread t = new Thread(  new ReadCommand() );
   t.start();

 }  else if ( c.equals( BLUElet.SELECTED ) )
    {
      display.setCurrent( remotedeviceui );

    } else if (d == spp_screen && c.getLabel().equals("Locate Room"))

    {
        bluelet.startInquiry( DiscoveryAgent.GIAC, new UUID[]{
SPP_Server.uuid } );
        display.setCurrent( bluelet.getUI() );

  } else if ( d == spp_screen && c.getLabel().equals("Server") )

    {
      // start local SPP server
        spp_server.run_server();
  } else if (d == spp_screen && c.getLabel().equals("Switch"))
  {     which = spp_screen.getSelectedIndex();
        Thread t = new Thread(  new SwitchCommand() );
        t.start();

  } else if ( c.equals( BLUElet.BACK ) )
    {
      bluelet.startInquiry( DiscoveryAgent.GIAC, new UUID[]{
SPP_Server.uuid } );
        display.setCurrent( bluelet.getUI() );
    } else if (d == spp_screen && c.getLabel().equals("Back"))
    {
      display.setCurrent( bluelet.getUI() );
      quitApp();
    } else if (d == spp_screen && c.getLabel().equals("Close"))
    {
        // Exit application.
        try
        {
```

```
            // attemp to close the server connection before existing...
            spp_server.done = true;
            spp_server.server.close();
          }
          catch (IOException ex)
          {
          }
          quitApp();
      }

    }

    public int pow(int a, int b)
    {
      int base = a;
      int power = b;
      int value = 1;
      for (int x=1;x<=power;x++)
      {
              value = value*2;
      }//end of for
      return value;
    }
    /**
     * An utility function to display a log message
     * @param s String
     */
    public static void log(String s)
    {
      System.out.println(s);
    }
    /**
     * An utility function that show a alert box that display an
exception message.
     * @param e
     * @param next_screen
     */
    public static void alert( Exception e, Displayable next_screen )
    {
      Alert alert = new Alert( "", "Exception: "+e.getClass().getName()+"
"+e.getMessage(), null, AlertType.INFO );
      alert.setTimeout( Alert.FOREVER );
      display.setCurrent( alert, next_screen );

    }
    /**
     * An utility function that show a alert box that display a message.
     *
     * @param m String
     * @param next_screen Screen
     */
    public static void alert( String m, Displayable next_screen )
    {
      Alert alert = new Alert( "", m, null, AlertType.INFO );
      alert.setTimeout( Alert.FOREVER );
      display.setCurrent( alert, next_screen );

    }

   public class ReadCommand implements Runnable
    {
      public void run()
      {//==================test mode================
        if (connCounter == 0)
```

```
      {
      //int temporaryInt = valueOf(temporaryString);
      // service found, read command from this service using SPP
connection
      spp_client.read_SPP_message( bluelet.getFirstDiscoveredService()
);
      maintain = bluelet.getFirstDiscoveredService();
      }
      else
       spp_client.read_SPP_message( maintain );
      tempPin = spp_client.x;
      //spp_screen.cleanScreen();
      for(int i=0;i<8;i++)
      {
            pinArray[i]=tempPin % 2;
            tempPin = tempPin / 2;
            //System.out.println("Pin number " + i + "=" + status[i]);
                if (pinArray[i] == 0)
                    spp_screen.showoff(i);
                else
                    spp_screen.showon(i);
      }//end of for
      connCounter ++;
        display.setCurrent(spp_screen);
   // ============================================

        }
    }

  public class SwitchCommand implements Runnable
  {
    public void run()
    {
       int newStatusValue = 0;
       int swapValue;
      // service found, send command to this service using SPP
connection


      if (pinArray[which]==0)
          pinArray[which] = 1;
      else
          pinArray[which] = 0;


      for(int i=0;i<8;i++)
      {

                if (pinArray[i] == 0)
                    spp_screen.showoff(i);
                else
                {
                    //this is to calculate back the value to be sent to
the circuit
                    newStatusValue = newStatusValue + pow(2,i);
                    spp_screen.showon(i);
                }
      }//end of for
      display.setCurrent(spp_screen);
      spp_client.x = newStatusValue ;
      if (connCounter == 0)
        spp_client.send_SPP_message(
bluelet.getFirstDiscoveredService(), "" );
      else
```

```
        spp_client.send_SPP_message( maintain, "" );

    }
  }

}
```

SPP_Screen.java

```
package btgallery.spp_gui;

import javax.microedition.lcdui.*;
import btgallery.*;

public class SPP_Screen extends List
{
      private Image images;

  public SPP_Screen()
  {

    super("BlueTips",List.IMPLICIT);
    addCommand(new Command("Locate Room", Command.OK, 1));
    //addCommand(new Command("Server", Command.OK, 2));
    addCommand(new Command("Switch",Command.OK,2));
    setCommandListener( SPP_MIDlet.instance );
  }

  public void cleanScreen()
  {
  super.deleteAll();
  }

  public void showoff(int pin)
    {
      switch (pin){
      case 0: super.deleteAll();
              super.append("[OFF] Dressing Lamp", null);
              break;
      case 1: super.append("[OFF] Right Light Pole", null);
              break;
      case 2: super.append("[OFF] Left Light Pole", null);
              break;
      case 3: super.append("[OFF] Bathroom", null);
              break;
      case 4: super.append("[OFF] Wet Kitchen", null);
              break;
      case 5: super.append("[OFF] Kitchen's Light", null);
              break;
      case 6: super.append("[OFF] Alarm", null);
              break;
      case 7: super.append("[OFF] Ventilator", null);
              break;
      }

    }

  public void showoff()
    {

      super.deleteAll();
      super.append("[OFF] Light", null);
```

```
    }

  public void showon(int pin)
    {
      switch (pin){
      case 0: super.deleteAll();
              super.append("[ON ] Dressing Lamp", null);
              break;
      case 1: super.append("[ON ] Right Light Pole", null);
              break;
      case 2: super.append("[ON ] Left Light Pole", null);
              break;
      case 3: super.append("[ON ] Bathroom", null);
              break;
      case 4: super.append("[ON ] Wet Kitchen", null);
              break;
      case 5: super.append("[ON ] Kitchen's Light", null);
              break;
      case 6: super.append("[ON ] Alarm", null);
              break;
      case 7: super.append("[ON ] Ventilator", null);
              break;
      }

    }


    public void showon()
    {

      super.deleteAll();
      super.append("[ON]  Light", null);
    }
  public void showonBen()
  {
    super.deleteAll();
    super.append("YATTAAAA!!!!", null);
  }

}
```

Splash.java

```
package btgallery.spp_gui;
import javax.microedition.lcdui.*;
import java.io.IOException;
import btgallery.*;


public class Splash extends Form implements Runnable, CommandListener
{

      private SPP_MIDlet theMidlet;
      private ImageItem imageItem;
      private Command Continue;
      private long cycleStartTime;
      private boolean splashIsShown = true;

      public Splash(SPP_MIDlet midlet)
      {
            super("");
```

```
            theMidlet = midlet;
            initSplash();

            //create the timer thread
            Thread t = new Thread(this);
            t.start();

            append(imageItem);
            Continue = new Command("Start", Command.OK, 1);
            addCommand(Continue);
            setCommandListener(this);
      }


      public void initSplash()
      {
            Image dictionaryImage = null;
            try
            {

                  //Construct the imageItem item
                  dictionaryImage =
Image.createImage("/btgallery/images/bluetips.png");
            }

            catch(IOException ioe)
            {
                  append("Unable to load image");
            }

            imageItem = new ImageItem(null, dictionaryImage,
ImageItem.LAYOUT_CENTER,null);


            cycleStartTime = System.currentTimeMillis();
      }


      public void run() {
            doTimeConsumingInit();
      }


      private static final int TIME_LOAD = 10000;

      public void doTimeConsumingInit()
      {

         try
         {
            while (splashIsShown)
            {
                  //5 seconds
                  long timeLength= (System.currentTimeMillis() -
cycleStartTime);
                  if (timeLength > TIME_LOAD)
                  {
                        splashIsShown = false;
                        theMidlet.activateMainMenu();
                  }
            }
            }
```

```
            catch(Exception ex)
            {
              System.out.println("App exception: " + ex);
              ex.printStackTrace();
            }

      }


      public void commandAction(Command command, Displayable
displayable)
      {
            if (command == Continue)
            {
                  splashIsShown = false;
                  theMidlet.activateMainMenu();
            }
      }



}
```

SPP_Server.java

```
package btgallery.spp_bt;

import javax.microedition.lcdui.*;
import javax.bluetooth.*;
import javax.microedition.io.*;
import java.io.*;
import btgallery.*;
import btgallery.spp_gui.*;

public class SPP_Server implements Runnable
{
  LocalDevice device;
  DiscoveryAgent agent;

  //public final static UUID uuid = new
UUID("1020304050607080 90A0B0C0D0E0F010", false);
public final static UUID uuid = new UUID("11111", true);
  // major service class as SERVICE_TELEPHONY
  private final static int SERVICE_TELEPHONY = 0x400000;

  // control flag for run loop
  // set true to exit loop
  public boolean done = false;

  // BT server connection
  public StreamConnectionNotifier server;

  public SPP_Server()
  {
  }

  public void run_server()
  {
    try
    {
      // initialize the JABWT stack
```

```
      device = LocalDevice.getLocalDevice(); // obtain reference to
singleton
      device.setDiscoverable(DiscoveryAgent.GIAC); // set Discover mode
to LIAC

      // start a thread to serve the server connection.
      // for testing purposes, only one server thread to start
      // see run() for the task of this thread
      Thread t = new Thread( this );
      t.start();

    } catch ( BluetoothStateException e )
    {
      e.printStackTrace();
    }

  }

  public void run()
  {
    // human friendly name of this service
    String appName = "SPPServerTesting";

    // connection to remote device
    StreamConnection c = null;
    try
    {
      String url = "btspp://localhost:" + uuid.toString() +";name="+
appName;
      log("server url: " + url );

      // Create a server connection object, using a
      // Serial Port Profile URL syntax and specific UUID
      server =  (StreamConnectionNotifier)Connector.open( url );

      // Retrieve the service record template
      ServiceRecord rec = device.getRecord( server );

      // set ServiceRecrod ServiceAvailability (0x0008) attribute to
indicate our service is available
      // 0xFF indicate fully available status
      // This operation is optional
      rec.setAttributeValue( 0x0008, new DataElement(
DataElement.U_INT_1, 0xFF ) );

      // Print the service record, which already contains
      // some default values
      Util.printServiceRecord( rec );

      // Set the Major Service Classes flag in Bluetooth stack.
      // We choose Telephony Service
      rec.setDeviceServiceClasses( SERVICE_TELEPHONY  );

    } catch (Exception e)
    {
      e.printStackTrace();
      log(e.getClass().getName()+" "+e.getMessage());
    }

    while( !done)
    {
      try {
        log("local service waiting for client connection...");
```

```
            // start accepting client connection.
            // This method will block until a client
            // connected
            c = server.acceptAndOpen();

            log("accepted a client connection. Checking appliance ");
            // retrieve the remote device object
            RemoteDevice rdev = RemoteDevice.getRemoteDevice( c );
            //input testing
            String s="254";
            DataOutputStream out = c.openDataOutputStream();
            out.writeUTF( s );
            out.flush();

            log("Current state of Appliance:" + s);

            // close current connection, wait for the next one

            try{
            // obtain an input stream to the remote service
            DataInputStream in = c.openDataInputStream();
            // read in a string from the string
            s = in.readUTF();
            log("Current state of Appliance:" + s);
            } catch (Exception e)
            {
                e.printStackTrace();
                c.close();
            }
          } catch (Exception e)
          {
            e.printStackTrace();
            SPP_MIDlet.alert(e, SPP_MIDlet.instance.spp_screen );
          }

        } // while

      }

      /**
       * An utility function to display a log message
       * @param s String
       */
      public void log( String s )
      {
        SPP_MIDlet.log( s );
      }

}
```

SPP_Client.java

```
package btgallery.spp_bt;

import java.io.*;
import javax.bluetooth.*;
import javax.microedition.io.*;

import btgallery.spp_gui.*;
import btgallery.bluelet.*;

public class SPP_Client
```

```
{

 public RemoteDeviceUI remotedeviceui = null;
 public String s;
 public int x;
 public StreamConnection con = null;
 public DataInputStream in;


  public SPP_Client()
  {
  }

  /**
   * Send a message to server using Serial Port Profile.
   * Connect to incoming service record, read current parallel port
status
   * and send command to switch the appliance to the server.
   * Device and service discovery is part of Serial Port client but it
is
   * done by Bluelet component. See SPP_MIDlet for usage of Bluelet.
   */

  public void read_SPP_message(ServiceRecord r)
  {
    // obtain the URL reference to this service on remote device
    String url =
r.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false );

    try
    {
      // obtain connection and stream to this service
      con = (StreamConnection) Connector.open( url);
      log("Connected to server");

      // read command (current status) using input stream
      in = con.openDataInputStream();


      x=in.readInt();
      con.close();
      in.close();

    } catch (Exception e)
    {
      e.printStackTrace();
      SPP_MIDlet.alert( e, SPP_MIDlet.instance.spp_screen );
    }

  }

  public void send_SPP_message(ServiceRecord r, String msg)
  {

    // obtain the URL reference to this service on remote device
    String url =
r.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false );

    try
    {
      // obtain connection and stream to this service
      con = (StreamConnection) Connector.open( url);
```

```
      DataOutputStream out = con.openDataOutputStream();
      // write data into serial stream
      //out.writeUTF( msg );
      out.writeInt(x);
      out.flush();

      // this wait is artificial, the purpose to do wait until the
      // server side really receive the message before close the
connection
      // in theory, this is not necessary, but sometimes the connection
dropped
      // on the server side when to close
      //Thread.sleep(1000);

      // finish, close output stream
      out.close();
      //open input stream to again check the current status and switch
the command
      in = con.openDataInputStream();
      //s = in.readUTF();
      x = in.readInt();
      log("Switch OK!");
      log("Command has successfully been sent!");

      con.close();
      in.close();
    } catch (Exception e)
    {
      e.printStackTrace();
      SPP_MIDlet.alert( e, SPP_MIDlet.instance.spp_screen );
    }
  }

  public void log( String s )
  {
    SPP_MIDlet.log( s );
  }

}
```

RemoteDeviceUI.java

```
package btgallery.bluelet;

import javax.microedition.lcdui.*;
import javax.bluetooth.*;

import btgallery.*;

public class RemoteDeviceUI extends List
{

  public RemoteDeviceUI()
  {
    super("Room Server", List.IMPLICIT);

    addCommand( new Command( "Select", Command.SCREEN, 1 ) );
    addCommand( new Command( "Search Again", Command.SCREEN, 2 ) );
    addCommand( BLUElet.BACK );

    setCommandListener( BLUElet.instance );
  }
```

```
  /**
   * Set a one-line message to screen.
   * @param str String
   */
  public void setMsg( String str )
  {
    super.deleteAll();
    append( str, null );

  }
  /**
   * refresh the list with blutooth devices
   */
  public void showui()
  {
    super.deleteAll();

    if (BLUElet.devices.size() > 0)
    {
      for (int i = 0; i < BLUElet.devices.size(); i++)
      {
        try
        {
          RemoteDevice device = (RemoteDevice)
BLUElet.devices.elementAt(i);
          String name = device.getFriendlyName(false);
          append(name, null);

        } catch (Exception e)
        {
          e.printStackTrace();
        }
      }
    } else
    {
      append("[No Device Found]", null);
    }
  }

}
```

BLUElet.java

```
package btgallery.bluelet;

import java.util.*;
import javax.bluetooth.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

import btgallery.*;

public class BLUElet  implements CommandListener
{
  // Commands used in callback to idenfity BLUElet events.
  // COMPLETED - When both device and service discovery are completed.
  public static Command COMPLETED =  new Command( "COMPLETED",
Command.SCREEN, 1 );
  // SELECTED - When user has selected a Bluetooth device for service
search
```

```
  public static Command SELECTED =  new Command( "SELECTED",
Command.SCREEN, 1 );
  // BACK - When user press Back button on Bluetooth Devices screen
(RemoteDeviceUI)
  public static Command BACK = new Command( "Back", Command.BACK, 1 );


  // MIDlet reference
  public static MIDlet host;
  // allback CommandListener
  public static CommandListener callback;
  // self instance of BLUEletUI
  public static BLUElet instance;
  // reference to GUI display
  public static Display display;
  public int myCounter = 0;
  public ServiceRecord temp;
  public static Vector devices = new Vector();
  public static Vector deviceClasses = new Vector();
  public static Vector services = new Vector();
  public static int selectedDevice = -1;
//  public static int selectedService = -1;

  // discovery mode in device inquiry
  public int discoveryMode;
  // list of UUID to match during service discovery
  public UUID[] serviceUUIDs = null;

  // Bluetooth return code from device inquiry operation
  // see DiscoveryListener
  public int deviceReturnCode;
  // Bluetooth return code from service discovery operation
  // see DiscoveryListener
  public int serviceReturnCode;

  public RemoteDeviceUI remotedeviceui = null;
  private LocalDevice device;
  private DiscoveryAgent agent;

  /**
   * Creae a new BLUElet.
   * @param host MIDlet
   * @param listener CommandListener
   */
  public BLUElet(MIDlet host, CommandListener listener)
  {
    this.host = host;
    this.callback = listener;
    instance = this;
  }

  /**
   * Mirror MIDlet.startApp(), should be called by MIDlet startApp().
   */
  public void startApp() {

    display = Display.getDisplay(host);

    remotedeviceui = new RemoteDeviceUI();
    remotedeviceui.showui();
  }

  /**
   * Mirror MIDlet.pauseApp(), should be called by MIDlet pauseApp().
```

```
    */
   public void pauseApp()
   {
     // do nothing
   }

   /**
    * Mirror MIDlet.destroyApp(), should be called by MIDlet
destroyApp().
    */
   public void destroyApp(boolean unconditional)
   {
   }


   /**
    * Utility function to write log message.
    * @param s String
    */
   public static void log(String s)
   {
     System.out.println(s);
   }

   /**
    * Obtain reference to device selection screen component.
    * You should show this screen when user invoke device search.
    * @return Screen
    */
   public Screen getUI()
   {
     return remotedeviceui;
   }

   /**
    * Get all discovered services from selected remote device.
    * Your application call this method after your app receive COMPLETED
callback
    * event. This will return all services that match your UUIDs in
startInquiry().
    * @return ServiceRecord[]
    */
   public ServiceRecord[] getDiscoveredServices()
   {
     ServiceRecord[] r = new ServiceRecord[ services.size() ];
     services.copyInto( r );
     return r;
   }

   /**
    * Get the first discovered service from selected remote device.
    * Application call this method after app receives COMPLETED
    * callback event. This will return the first service that match
    * UUIDs in startInquiry().
    *
    * @return ServiceRecord null if no service discovered
    */
   public ServiceRecord getFirstDiscoveredService()
   {
/*       if ( services.size() > 0 )
           return (ServiceRecord) services.elementAt(0);
       else
           return null;
 **/
```

```
      if (myCounter == 0)
    {
        if ( services.size() > 0 )
        {
            temp = (ServiceRecord) services.elementAt(0);
            myCounter ++;
            //return (ServiceRecord) services.elementAt(0);
        }

        //else
           // return null;
    }
      //else
          return temp;

  }

  /**
   * Return the Bluetooth result code from device inquiry.
   * This is the result code obtained in
DiscoveryListener.inquiryCompleted().
   * Application cal call this method after a COMPLETED callback event
   * is received.
   * @return int
   */
  public int getDeviceDiscoveryReturnCode()
  {
    return deviceReturnCode;
  }


  /**
   * Return the Bluetooth result code from service discovery.
   * This is the result code obtained in
DiscoveryListener.serviceSearchCompleted().
   * Application cal call this method after a COMPLETED callback event
   * is received.
   * @return int
   */
  public int getServiceDiscoveryReturnCode()
  {
    return serviceReturnCode;
  }

  /**
   * Return user selected remote device that is used for service
discovery.
   * Application can call this after app received SELECTED callback
   * event.
   * @return RemoteDevice null if user didn't select anything
   */
  public RemoteDevice getSelectedDevice()
  {
    if ( selectedDevice != -1 )
      return (RemoteDevice) devices.elementAt(selectedDevice);
    else
      return null;
  }

  /**
   * Start device inquiry. Your application call this method to start
inquiry.
   * @param mode int one of DiscoveryAgent.GIAC or DiscoveryAgent.LIAC
   * @param serviceUUIDs UUID[]
```

```
   */
  public void startInquiry( int mode, UUID[] serviceUUIDs )
  {
    try
    {
      this.discoveryMode = mode;
      this.serviceUUIDs = serviceUUIDs;

      // clear previous values first
      devices.removeAllElements();
      deviceClasses.removeAllElements();

      //
      // initialize the JABWT stack
      device = LocalDevice.getLocalDevice(); // obtain reference to
singleton
      device.setDiscoverable(DiscoveryAgent.GIAC); // set Discover Mode
      agent = device.getDiscoveryAgent(); // obtain reference to
singleton


      boolean result = agent.startInquiry( mode, new Listener() );

      // update screen with "Please Wait" message
      remotedeviceui.setMsg("[Please Wait...]");

    } catch ( BluetoothStateException e )
    {
      e.printStackTrace();
    }

  }

  /**
   *
   * @param c Command
   * @param d Displayable
   */
  public void commandAction(Command c, Displayable d)
  {
    if ( d == remotedeviceui && c.getLabel().equals("Search Again") )
    {
      startInquiry( discoveryMode, serviceUUIDs );

    }
    else if ( d == remotedeviceui && c.getLabel().equals("Select") )
    {
      // get selected device
      selectedDevice = remotedeviceui.getSelectedIndex();
      RemoteDevice remoteDevice = (RemoteDevice) devices.elementAt(
selectedDevice );

      // remove all existing record first
      services.removeAllElements();

      try
      {
        agent.searchServices(null,
                             serviceUUIDs,
                             remoteDevice,
                             new Listener() );

        // tell callback device selected
        display.callSerially(new Worker(ID_DEVICE_SELECTED));
```

```
      } catch (BluetoothStateException ex)
      {
        ex.printStackTrace();
      }

    } else if ( d == remotedeviceui && c.getLabel().equals("Back") )
    {
      callback.commandAction( BACK, remotedeviceui);

    }

  }

  /**
   * Bluetooth listener object.
   * Register this listener object to DiscoveryAgent in device inqury
and service discovery.
   */
  class Listener implements DiscoveryListener
  {

    public void deviceDiscovered(RemoteDevice remoteDevice,
                                 DeviceClass deviceClass)
    {
      log("A remote Bluetooth device is discovered:");
      Util.printRemoteDevice( remoteDevice, deviceClass );
      devices.addElement( remoteDevice );
      deviceClasses.addElement( deviceClass );
    }

    public void inquiryCompleted(int complete)
    {
      log("device discovery is completed with return code:"+complete);
      log(""+devices.size()+" devices are discovered");

      deviceReturnCode = complete;

      if ( devices.size() == 0 )
      {
        Alert alert = new Alert( "Bluetooth", "No Bluetooth device
found", null, AlertType.INFO );
        alert.setTimeout(3000);
        remotedeviceui.showui();
        display.setCurrent( alert, remotedeviceui );

      } else
      {
        remotedeviceui.showui();
        display.setCurrent( remotedeviceui );
      }

    }

    public void servicesDiscovered(int transId, ServiceRecord[]
records)
    {
      // note: we do not use transId because we only have one search at
a time
      log("Remote Bluetooth services is discovered:");
      for ( int i=0; i< records.length;  i ++ )
      {
        ServiceRecord record = records[i];
        Util.printServiceRecord( record );
```

```
        services.addElement( record );
      }
    }

    public void serviceSearchCompleted(int transId, int complete)
    {
      // note: we do not use transId because we only have one search at
a time
      log("service discovery completed with return code:"+complete);
      log(""+services.size()+" services are discovered");

      serviceReturnCode = complete;

      // we cannot callback in this thread because this is a Bluetooth
      // subsystem thread. we do not want to block it.
      display.callSerially( new Worker( ID_SERVICE_COMPLETED ) );

    }

  } // Listener

  private final static int ID_SERVICE_COMPLETED = 1;
  private final static int ID_DEVICE_COMPLETED = 2;
  private final static int ID_DEVICE_SELECTED = 3;

  /**
   * Worker thread that invoke callback CommandListener upon Bluetooth
event occurs.
   */
  class Worker implements Runnable
  {
    int cmd = 0;

    public Worker( int cmd )
    {
      this.cmd = cmd;
    }
    public void run()
    {
      switch (cmd) {
        case ID_SERVICE_COMPLETED:
          callback.commandAction( COMPLETED, remotedeviceui);

          break;
        case ID_DEVICE_COMPLETED:
          callback.commandAction( COMPLETED, remotedeviceui);

          break;
        case ID_DEVICE_SELECTED:
          callback.commandAction( SELECTED, remotedeviceui);

          break;
        default:
          break;

      }
    }
  }


}
```