

IMPLEMENTATION OF SHORT RANGE COMMUNICATION SERVICE

JITENDRA SINGH RAJPUT¹, ARUN PARAKH² & SURESH GHODE³

^{1,3}Research Scholar, (Digital Techniques & Instrumentation), Department of Electrical Engineering,
Shri G. S. Institute of Technology & Science, Indore, Madhya Pradesh, India

²Assistant Professor, Department of Electrical, SGSITS, Indore, Madhya Pradesh, India

ABSTRACT

Communication service between computers in short-range become an essential requirement in recent years. We have devised a protocol to establish a communication service in such an environment where neither physical network nor Wi-Fi is available. We have used Bluetooth technology to implement this concept. Bluetooth has client-server architecture; the one that initiates the connection is the client, and the one who receives the connection is the server. Bluetooth is a great protocol for wireless communication because it's capable of transmitting data at nearly 3 MB/s, while consuming 1/100th of the power of Wi-Fi. This paper explains how we can make chat application using Bluetooth programming in C. We made this application on Linux operating system using the BlueZ Bluetooth protocol stack and libraries.

KEYWORDS: Bluetooth, BlueZ, Client-Server, Protocol Stack

INTRODUCTION

In this paper we present a low cost, portable system with wireless transmission of data (text) from PC to PC on Linux operating system by Bluetooth dongle. Bluetooth is a method for data communication that uses short-range radio links to replace cables between computers and their connected units. Industry-wide Bluetooth promises very substantial benefits for wireless network operators, end workers, and content developers of exciting new applications. This paper explains the protocol stack and architecture of Bluetooth technology. It also describes that how we can make chat-server with Bluetooth programming in Linux operating system with C programming. We wrote software programs using C language for detecting nearby devices and communicating with those devices (i.e. sending and receiving data).

The Bluetooth standard and technology came about initially when Ericsson Mobile Communications carried out a study to find a low power and low cost radio interface between mobile phones and their accessories. The study showed that a short-range radio link solution was feasible. To develop the technique and to get broad market support, Ericsson, together with Intel, IBM, Toshiba and Nokia Mobile Phones formed a Special Interest Group (SIG) in 1998. This group was to form a standard for the air interface and the software that controls it, such as to achieve interoperability between different devices from different producers [9].

Bluetooth operates in the 2.4 GHz Industrial, Scientific and Medical (ISM) band using frequency hopping spread spectrum (FHSS) and binary Gaussian Frequency Shift Keying (GFSK) modulation. A pseudorandom hopping algorithm is used to select the next frequency out of 79 possible 1 MHz wide channels. Bluetooth devices wishing to communicate with each other form a piconet, which consists of a master, who controls all communication on the piconet, and up to 7 slaves. The slaves may only communicate with the master, and not with each other. The master and slaves will take turns transmitting, with a slave responding only when polled by the master.

The name itself was derived from that of the Danish Viking King Harald Bluetooth who ruled Denmark from 940 to 981. Harald Bluetooth was well known for getting people to communicate and one of his most significant achievements was the uniting of Denmark and Norway. Since Bluetooth was developed with the idea of connecting devices and people in mind, his name was adopted [12].

The contribution of the work is to develop a prototype for short range communication service which enables the communication between multi users without pre exist a costly infrastructure.

This paper is organized as follows. The following section describes the Bluetooth technology and protocol stacks. Section III presents the design and implementation. Section IV analyses the experiments and results. Finally, section V gives the conclusion and future work.

BLUETOOTH TECHNOLOGY AND PROTOCOL STACK

Bluetooth Network Topologies

Bluetooth devices can be organized into groups of two to eight devices, which together form a piconet. Each piconet will contain at least one master, and all other units participating in the piconet will be slaves. The master of a piconet controls the communications within a piconet [10].

The number of units in a piconet is deliberately limited to eight (one master, seven slaves) in order to keep a high capacity link between all the units. It also limits the overhead required for addressing. Note that the master/slave role only lasts for the duration of the piconet. Once the piconet is cancelled, these roles are also cancelled. Any unit can become master or slave. By definition, the unit that establishes the piconet becomes the master.

Two or more piconets can be interconnected to form a scatternet. The connection point between two piconets is a Bluetooth unit that is a member of both piconets. One device can be a master in one piconet, and a slave in another. A device can also be a slave on more than one piconet, but it cannot be a master in more than one piconet, as this will mean that the two are actually one piconet (having a single master) [12].

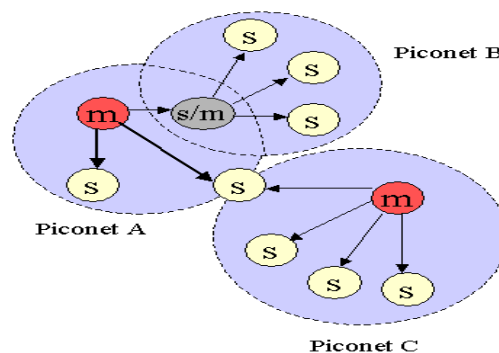


Figure 1: Shows a Scatternet Comprising of Three Piconets[12]

Bluetooth Software Stack

Bluetooth can be defined as a layered protocol architecture consisting of protocols such as the cable replacement protocol, the telephony protocol, the adopted protocol and the core protocols.

The core protocols are four protocol layers consisting of [1, 9]:

Baseband. Specifies details of the air interface, including frequency, the use of frequency hopping, modulation scheme, and transmit power. Also specifies the protocol for connection establishment within a piconet, addressing, packet format, order of transmission, timing sequence, power control and channel coding.

Link manager protocol (LMP). Responsible for link setup between Bluetooth devices and ongoing link management such as encryption and security. Allows service discovery, which detects other Bluetooth devices when in range.

Logical link control and adaptation protocol (L2CAP). Adapts upper layer protocols to the baseband layer. It is also responsible for the multiplexing data to upper layer. This is done by assigning a particular PSM (multiplexer number).

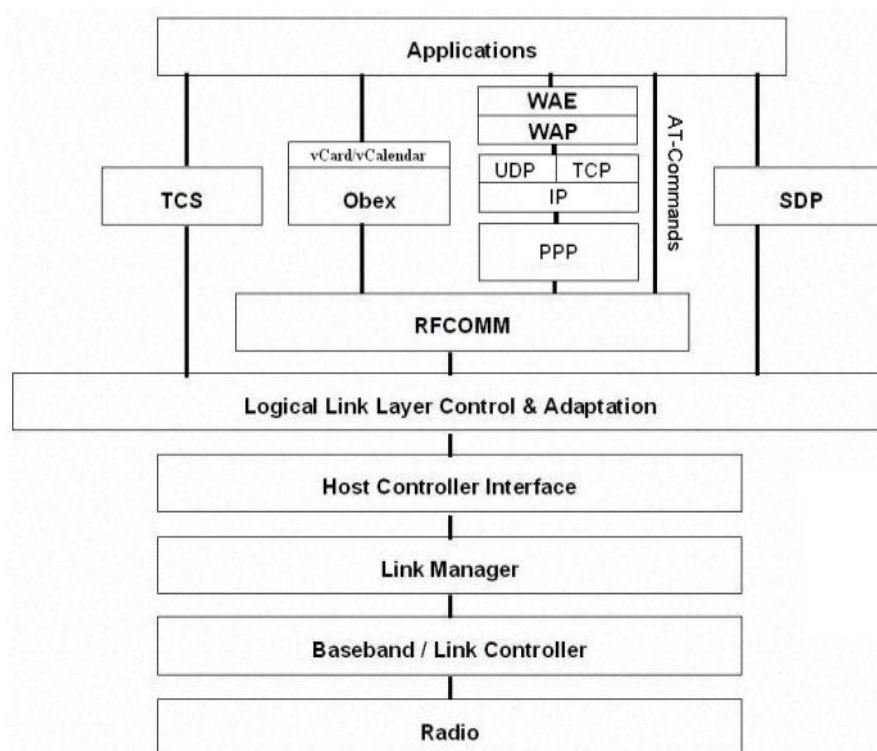
Service discovery protocol (SDP). Query devices of possible services available. Does not require a connection to be made.

We are interested in three particular layers, as they are the communication layers that we are implement our application in. These layers are:

L2CAP layer acts as an adaptation layer to the upper layer, but it also allows connectionless data transfer.

RFCOMM layer is a cable communication protocol designed to emulate serial ports.

HCI (Host Controller Interface) sits between the L2CAP and LMP. It is the first interface between programmer and the protocol, allowing access to the hardware capabilities. It is also responsible for the multiplexing data to upper layer.



Source: Bluetooth Specifications [<http://www.bluetooth.com>]

Figure 2: Bluetooth Protocol Stack

DESIGN AND IMPLEMENTATION

We decided to program at the L2CAP layer because RFCOMM only provides one-to-one connections while L2CAP allows multiplexing of data via the PSM number. This means that two Bluetooth devices can have more than one connection to each other, allowing different Bluetooth programs to run simultaneously.

We can transmit messages (text) from one PC to other using our program which is written in c language.

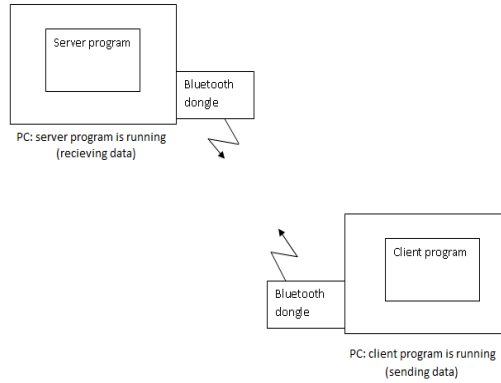


Figure 3: Communication between Two PC over Bluetooth

When we run the server program, the server goes in listening mode and it receives the data which is send by the client. The client knows the address of Bluetooth device with whom it wants to communicate.

The different parts of Bluetooth programming can be separated into several components:

- Choosing a device to communicate
- Figuring out how to communicate with it
- Making an outgoing connection
- Accepting an incoming connection
- Sending data
- Receiving data

Below is the flow chart which summarizes the programming steps for making outgoing and incoming connection.

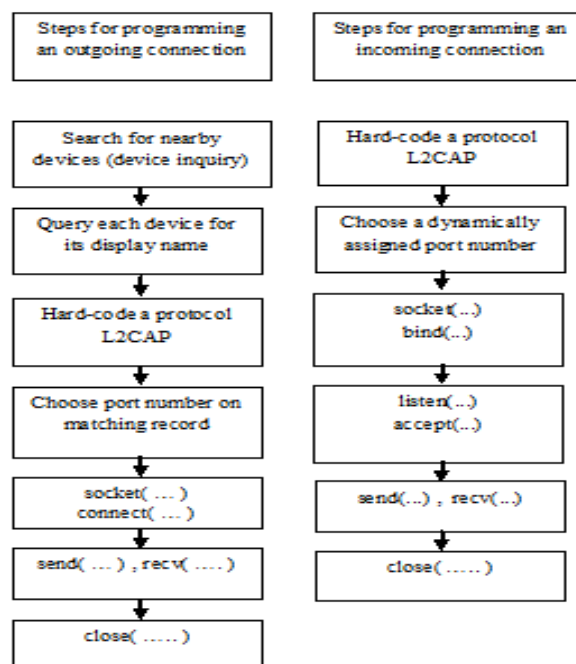


Figure 4: Shows Programming Steps of Making Outgoing and Incoming Connection

The explanation of programming steps is given below:

The first step is to allocate or create a socket.

int socket (int domain, int type, int protocol)

For L2CAP sockets, the three parameters to the socket function call should always be AF_BLUETOOTH, SOCK_STREAM, and BTPROTO_L2CAP. The first parameter, AF_BLUETOOTH, specifies that it should be a Bluetooth socket. The second, SOCK_STREAM, requests a socket with streams-based delivery semantics. The third, BTPROTO_L2CAP, specifically requests an L2CAP socket. The socket function creates the L2CAP socket and returns an integer that is used as a handle to that socket.

Once created, a socket must be connected in order to be of any use. The procedure for doing this depends on whether the application is accepting incoming connections (server sockets), or whether it's creating outgoing connections (client sockets). Client sockets are simpler, and the process only requires making a single call to the connect function

int connect (int sock, const struct sockaddr *server_info, socklen_t infolen)

Server sockets are a bit more complicated and involve three steps instead of just one. After the server socket is created, it must be bound to a local Bluetooth adapter and port number with the bind function

int bind (int sock, const struct sockaddr *info, socklen_t infolen)

The first parameter, sock, is the server socket created by connect. info should point to a struct sockaddr_rc addressing structure filled in with the local Bluetooth adapter to use, and which port number to use. After the socket is successfully bound, it should be placed into listening mode by using the listen function

int listen (int sock, int backlog)

In between the time an incoming Bluetooth connection is accepted by the operating system and the time that the server application actually takes control, there may be other incoming connection requests. These new ones are placed into a backlog queue. The backlog parameter specifies how big this queue should be. Usually, a value of 1 is fine. Once these steps have completed, the server application is ready to accept incoming connections using the accept function

int accept (int server_sock, struct sockaddr *client_info, socklen_t *infolen)

The accept function waits for an incoming connection and returns a brand new socket. The returned socket represents the newly established connection with a client, and is what the server application should use to communicate with the client. If client_info points to a valid struct sockaddr_rc structure, then it is filled in with the client's information. Additionally, infolen will be set to sizeof(struct sockaddr_rc). The server application can then make another call to accept and accept more connections, or it can close the server socket when finished.

Bluetooth Dongle

We have used the two Bluetooth dongle of Enter company in our work. The specification is shown below:

Specification

- Support networking, Dial-up, Fax, LAN access, and headset.
- Operation system: Windows 98, 98se, Me, 2000, XP, Vista, WIN7, Linux.
- Wireless connects to Bluetooth devices such as mobile phone, PDA or PC, for data transfer.
- Supporting Bluetooth voice data.
- Data Sending & Receiving Range : 0 ~ up to 100 m.

Linux and USB Dongle

The Linux operating system is part of GNU, so it is freely available and downloadable from the internet. GNU software means that the public is allowed to distribute, use, and modify the software.

After related study, we found a Bluetooth stack for Linux called Bluez. Bluez had become the official Bluetooth stack for Linux. Just like Linux, Bluez is also GNU software.

The advantages of using Linux with the Bluetooth USB device are:

- The cost for a Bluetooth dongle is relatively low.
- The Linux operating system and Bluez Bluetooth stack are free.
- Uses C programming language that comes with most Linux Operating System. Such as GCC or CC.
- Bluez implements most layers in the stack such as SDP, RFCOMM, L2CAP, HCI layers.

In this work we have used ubuntu Linux Version 12.04 it has inbuilt Bluez.

Linux and BlueZ

Recently, Bluez has been made the official Linux Bluetooth stack. Thus, a more recent Linux distribution with version 2.4.6 or higher would include Bluez packages. This also suggests that Bluez has been recognized as a reliable stack compared to other Linux Bluetooth stacks, and it will receive more attention and improvements by the open source community. Bluez was originally developed by Qualcomm Incorporated [19].

Bluez implements almost all of the available stacks that is defined by the Bluetooth standard. The lists of implemented stacks taken from the Bluez official website are shown in table-1.

Table 1: BlueZ Stacks and their Use

Bluez Stack	Use of Stack
Bluetooth Core	HCI device and connection manager.
L2CAP	Reliable datagram protocol.
RFCOMM	Serial port emulation. Reliable connection oriented streaming protocol.
SCO	Synchronized connection (voice).

Bluez supports link layer security, and multiple connections. Those are crucial for our project. On top of those, it also allows multiple Bluetooth devices, which may be required for future extensions to this work.

EXPERIMENT AND RESULTS

We have developed C programs for communicating between PC to PC. We use a BlueZ software stack and libraries for compile our application. We made three programs; one for searching nearby device, one for server application and the other for client application.

After the implementation of the specified system following experiments are performed and the results are obtained as shown in figure 5,6,7,8.

Searching a Nearby Device

This program shows the nearby Bluetooth devices, there addresses and user defined name as shown in figure 5.

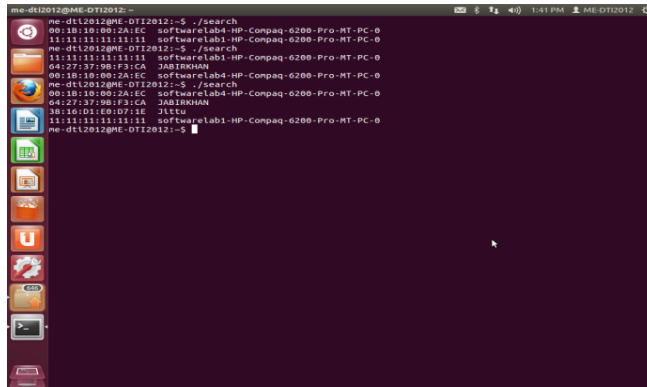


Figure 5: Screenshot of Nearby Bluetooth Devices Program

Server Program Waiting for Messages (Text Data)

When we compile and run our server program it goes to listening mode and wait for messages from client. The screenshot is shown in figure 6.

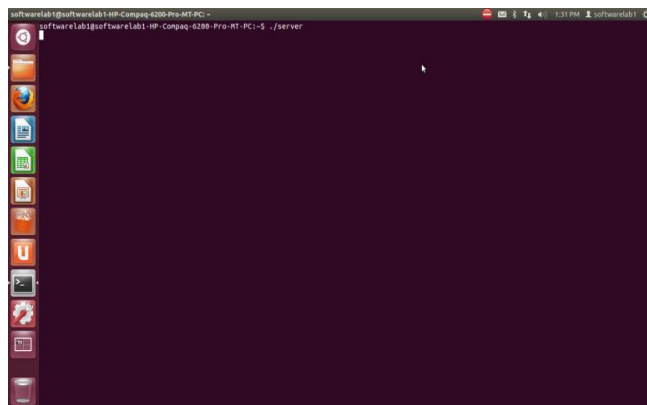


Figure 6: Screenshot of Server Program

Client Program for Sending Messages (Text Data)

When we compile and run our client program it ask for the address of device with whom it wants to communicate then it send the messages to server. The client screen is shown in figure 7

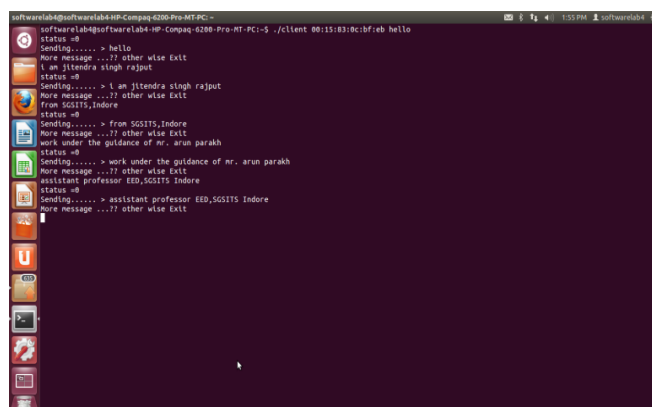
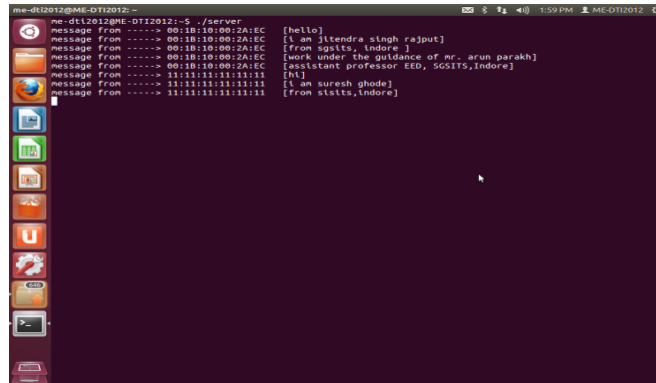


Figure 7: Screenshot of Client Program

After establishing connection between client and server, the communication begins. The server can continuously send data and the server receives the data. The communication is bidirectional it means that we can use the server and client application on the same PC. The same programme is run on the other PC so we can get the bidirectional communication as shown in figure 8.



```

me-dt12012@ME-DT12012:~$ ./server
message from ----> 00:10:10:00:2A:EC [hello]
message from ----> 00:10:10:00:2A:EC [I am jitendra singh rajput]
message from ----> 00:10:10:00:2A:EC [from sgits, Indore ]
message from ----> 00:10:10:00:2A:EC [work under the guidance of mr. arun parakh]
message from ----> 00:10:10:00:2A:EC [assistant professor EED, SGSITS,Indore]
message from ----> 11:11:11:11:11:11 [hi]
message from ----> 11:11:11:11:11:11 [I am suresh ghode]
message from ----> 11:11:11:11:11:11 [from sgits,Indore]

```

Figure 8: Screenshot of Server Program after Communication Established

CONCLUSIONS AND FUTURE PLAN

Programming client/server applications in Bluetooth is a tricky process due to the wide variety of stacks, protocols, profiles, and versions. We have successfully completed the communication between different Bluetooth devices.

Our implementation gives freedom from paid network or high initial cost infrastructure. This is a portable solution for short range communication service. It means we don't require any heavy cost LAN or Wi-Fi or GSM type infrastructure. Our application doesn't require any specific software to be installed that further reduces the requirement of resources like CPU time, main memory.

Security is an important aspect of Bluetooth programming. It is of great interest to study of Bluetooth security manager. Looking into how the Bluetooth security manager is implemented on Bluetooth devices is also an exciting subject.

In future this application will be equipped with better GUI and security. We are also suggesting its next version as a mobile phone compatible.

Limitation of our work at present is limited number of communication so in future we try to address this issue.

REFERENCES

1. Bluetooth based home automation system N. Sriskanthan*, F. Tan, A. Karande [School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore, Singapore 639798 Received 17 September 2001; revised 8 May 2002; accepted 10 May 2002]
2. Home Appliance Control System over Bluetooth with a Cellular Phone Hiroshi Kanma, Noboru Wakabayashi, Ritsuko Kanazawa, Hiromichi Ito, Manuscript received July 20, 2003 0098 3063 © 2003 IEEE
3. Bluetooth Special Report Website",
<http://www.zdnet.co.uk/news/specials/1999/04/bluetooth/>(current April. 20, 2001)
4. G. Held, Data Over Wireless Networks Bluetooth TM, WAP, and Wireless LANS, McGraw-Hill, United States of America, 2001.
5. Bluetooth for Programmers Albert Huang, Larry Rudolph
6. Distributed Wireless Control Using Bluetooth Johan Eker, Anton Cervin, Andreas Hörjel IFAC Conference on New Technologies for Computer Control, Hong Kong, P.R. China, November 2001.

7. GSM-Bluetooth based Remote Monitoring and Control System with Automatic Light Controller Vini Madan Research Scholar IGIT, IP University New Delhi, S.R.N Reddy Associate Professor IGIT, IP University New Delhi International Journal of Computer Applications (0975 – 8887) Volume 46– No.1, May 2012
8. Wireless Technologies for Industrial Applications Version 2.1 – 19/03/2012 Mats Andersson, CTO connectBlue AB, Sweden
9. Ericsson et al., Core, Specification of the Bluetooth System, Volume 1, version 1.1 February 2001. [<http://www.bluetooth.com/developer/specification/specification.asp>] (dec 2001)
10. Ericsson et al., Profiles, Specification of the Bluetooth System, Volume 2, version 1.1 February 2001. [<http://www.bluetooth.com/developer/specification/specification.asp>] (jan 2003)
11. Ericsson et al., Bluetooth Summary [<http://www.ericsson.com/bluetooth/bluetoothf/beginnersg/>] (Oct 2005)
12. Au-System, “Bluetooth Whitepaper”, [http://www.palowireless.com/infotooth/documents/Bluetooth_Whitepaper_-_AU_System.zip]
13. Matt Ziegler, “An Overview of Bluetooth: Architecture, Power Consumption and Performance”, [<http://www.ece.virginia.edu/~mmz4s/papers/>]
14. J.C. Haartsen, "The Bluetooth Radio System", IEEE Personal Communications, Vol.7, No.1, Feb. 2000, pp. 28-36.
15. “The Use of Bluetooth in Linux and Location Aware Computing.” Huang, A. Master's thesis. MIT CSAIL. 2005. Available at <http://csail.mit.edu/~albert/pubs/2005-ashuang-sm-thesis.pdf>
16. “Survivable Bluetooth location networks,” F.J. Gonzalez-Castano and J. Garcia-Reinoso, IEEE International Conference on Communications, Vol. 2, pp. 1014-1018, May 2003
17. “Indoor location estimation using multiple wireless technologies” D. Paiidya, R. Jain, and E. Lupu, IEEE International Symposium on Personal, Indoor and Mobile Radio Communication, Proceedings, Vol.3, pp. 2208-2212, 2003
18. “An indoor Bluetooth-based positioning system: concept, Implementation and experimental evaluation,” W. Zhuang, Chi-Hsiang Yeh, O. Droegehorn, C.-T. Toh, and H. R. Arabnia, International Conference on Wireless Networks, June 2003
19. “Bluez Project, “Programming Using Bluez”,<http://bluez.sourceforge.net/howto/node43.html>
20. “Implementing Automatic Location Update for Follow-Me Database Using VoIP and Bluetooth Technologies,” Yi-Bing Lin, Hsu-Yung Cheng, Ya-Hsing Cheng, and P. Agrawal, IEEE Transactions on Computers, Vol. 51, pp. 1154-1168, Oct. 2002

