

CHAPTER 3

METHODOLOGY

3.0 Introduction

Inform Departure and Arriving of Busses Using Bluetooth project will be divided into 2 main parts, server and client. The server part will run in a PC and the client will apply in the mobile phones that support the Java environment. The design and development process will be done using Netbeans 5.5 IDE. NetBeans 5.5 is capable to compile and run the Java programming program and capable in developing Graphical User Interface for PC and mobile application.

To simply install the Java environment in the workstation, bundle software from NetBeans is used. This installer comes with Netbeans 5.5 installer and Java Development Kit Update 6.0. So, the process to set up the Java environment becomes more simple and fast.

The installer for this software can be downloads from the NetBeans Official Web Site [3]. To develop a Bluetooth program in Java, BlueCove API is used. BlueCove is a JSR-82 implementation on Java 2 Standard Edition (J2SE) that currently interfaces with the Microsoft Bluetooth stack found in windows XP SP2 or newer. This API is originally developed by Intel Research and currently maintained by volunteer.

3.0.1 Project Flow Diagram

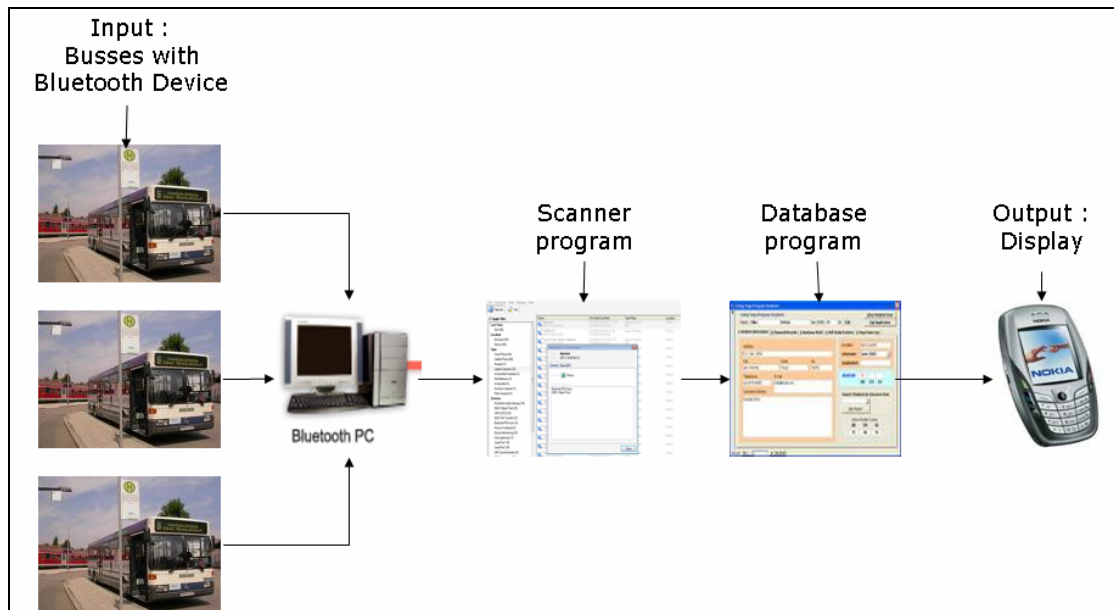


Figure 3.1: Overall flow of Bluetooth Bus Info

Figure 3.1 show the overall flow for Bluetooth Bus Info system. The input for this system is the busses with Bluetooth device. First of all, the details data about the journey of the busses is already save in the database. When the system is running, the Bluetooth device scanner will continuously scan the nearby area to detect the active Bluetooth devices. If the Bluetooth devices are detected, the system will compare the Bluetooth Address from the devices with the Bluetooth address save in database.

If the detected Bluetooth Address is match with Bluetooth Address in the database, the system will show the details data in the PC side GUI. From the Figure 3.1, the Bluetooth PC is the centre point for the system. This PC consists the software to scan the nearby Bluetooth devices and the database server.

The output for this system will be display in the GUI and mobile phone display. Phone side output only available when the user on phone side request the data from the PC side software.

3.0.2 Overall flow of Bluetooth Bus Info

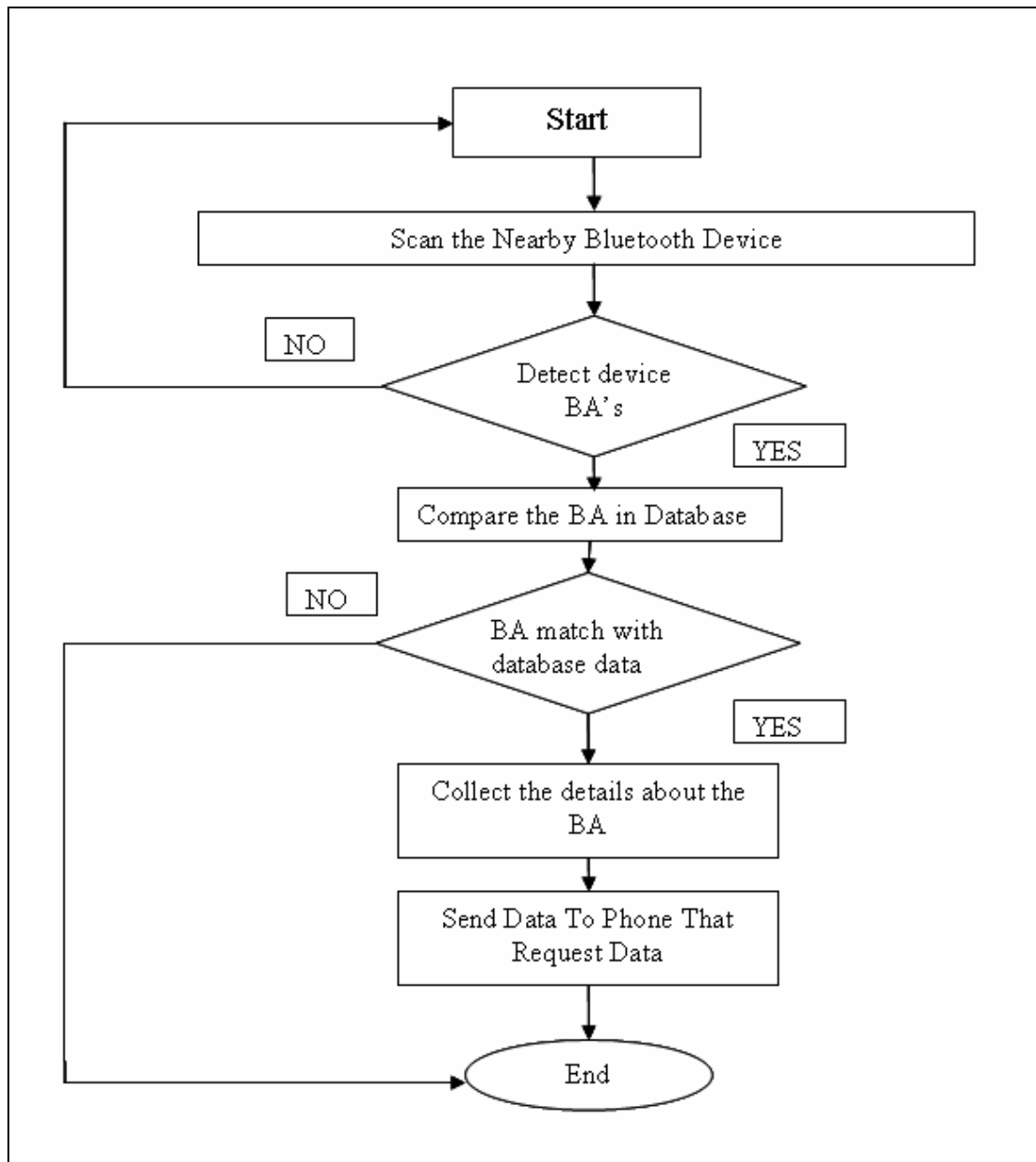


Figure 3.2: Flow Chart for Bluetooth Bus Info System

Figure 3.2 show the overall flow for Bluetooth Bus Info. BA is stand for Bluetooth Address. This figure also shows the flow for PC side program. When the system is started, the Bluetooth device scanner will automatically scan the nearby area to search for Bluetooth devices around the bus station. If devices are detected, the system will proceed to next step; compare the Bluetooth Address with Bluetooth

Address in the database. If no devices is detected, the system will continuously scan the area until new devices is detected.

If the Bluetooth devices address is matched with Bluetooth devices in the database, the program will collect the journey details from the database and show it on the GUI display. If Bluetooth address is not match with database data, the system will ignore the address and waiting for another data query. Data query is the method to search data from the database.

If the phone side is request the PC side program for the details data, the program will search the data and sent it to the phone side. The data will be display on the phone LCD display.

3.1 Develop the Bluetooth Device Scanner for PC application

The Bluetooth device scanner is a backbone to this project. This program will scan the nearby Bluetooth devices and display the Bluetooth address, current time and date in the GUI. Bluetooth Device scanner also acts as the server program to receive the request from the phone. This program will use the BlueCove API as the bridge to make the interface to Microsoft Bluetooth Protocol Stack.

To develop the Bluetooth Device Scanner we need to use the classes in the BlueCove API. The classes are `javax.bluetooth.LocalDevice`, `javax.bluetooth.RemoteDevice`, and `javax.bluetooth.DeviceClass`. These classes provided the device management capabilities that are part of the Generic Access Profile (GAP). To get the current time and date, `java.util` class is used. The source code for Bluetooth Device scanner can be view in **Appendix I**.

3.1.1 Local Device

The local Bluetooth device is represented by `javax.bluetooth.LocalDevice`. This class provides methods to manage and retrieve local device and information about it such as its Bluetooth address, device class, and discovery agent. Some of the methods provided by this class include:

- `static LocalDevice getLocalDevice()` – static method to retrieve the `LocalDevice` object that represents the local Bluetooth device.
- `java.lang.String getBluetoothAddress()`– retrieves the Bluetooth address of the local device. A Bluetooth address is represented as `java.lang.String` that represents a 12 characters long hexadecimal value.
- `DiscoveryAgent getDiscoveryAgent()`– returns the discovery agent for this device.
- `boolean setDiscoverable(int mode)` – sets the discoverable mode of the device.

3.1.2 Remote Device

The remote local Bluetooth device is represented by `javax.bluetooth.RemoteDevice`. This class provides methods to retrieve the `RemoteDevice` object associated with a Bluetooth connection, methods to learn the address and name of the remote device, and security-related methods. Some of the methods provided by this class include:

- `java.lang.String getBluetoothAddress()` – retrieves the Bluetooth address of the remote device. A Bluetooth address is represented as `java.lang.String` that represents a 12 characters long hexadecimal value.

3.1.3 Get Current Time and Date

Get current time is a method to get the local time. This class is from java.util. The use of this class in this program is to get the time when a Bluetooth device is detected. Besides time, this class also provide a method to get the current local date.

3.2 Graphical User Interface (GUI)

The graphical user interface is a computer interface that uses graphic icons and controls in addition to text. The user of the computer utilizes a pointing device, like a mouse, to manipulate these icons and controls. This is considerably different from the command line interface (CLI) in which the user types a series of text commands to the computer.

A graphical user interface (GUI) allows for interaction with a computer or other media formats which employ graphical images, widgets, along with text to represent the information and actions available to a user. The actions are usually performed through direct manipulation of the graphical elements.

3.2.1 Getting Started With NetBeans 5.0

To install the NetBeans 5.0 in the workstation, reader can get the installer from NetBeans website [4]. Follow the instruction to install the software. For tutorial or help, please refer to this website [5]. To develop an application program for Java in NetBeans IDE, please refer to **Appendix II**.

3.2.2 PC side GUI

To make the system a user friendly program, a GUI for this system will be develop. GUI for the PC side will be divided into 4 parts. Scan tab will be the place to start the scanning process and start the server for receives connection from the phone. Departure Database will conduct the data for the departure information. In this tab, the user will have an access to edit, add and delete data to the departure database. The same design will be found in the Arriving Database tab. Departure and Arriving tab will show the detail about the busses information based on the data in the database.

3.2.3 Bluetooth Device Scanner GUI

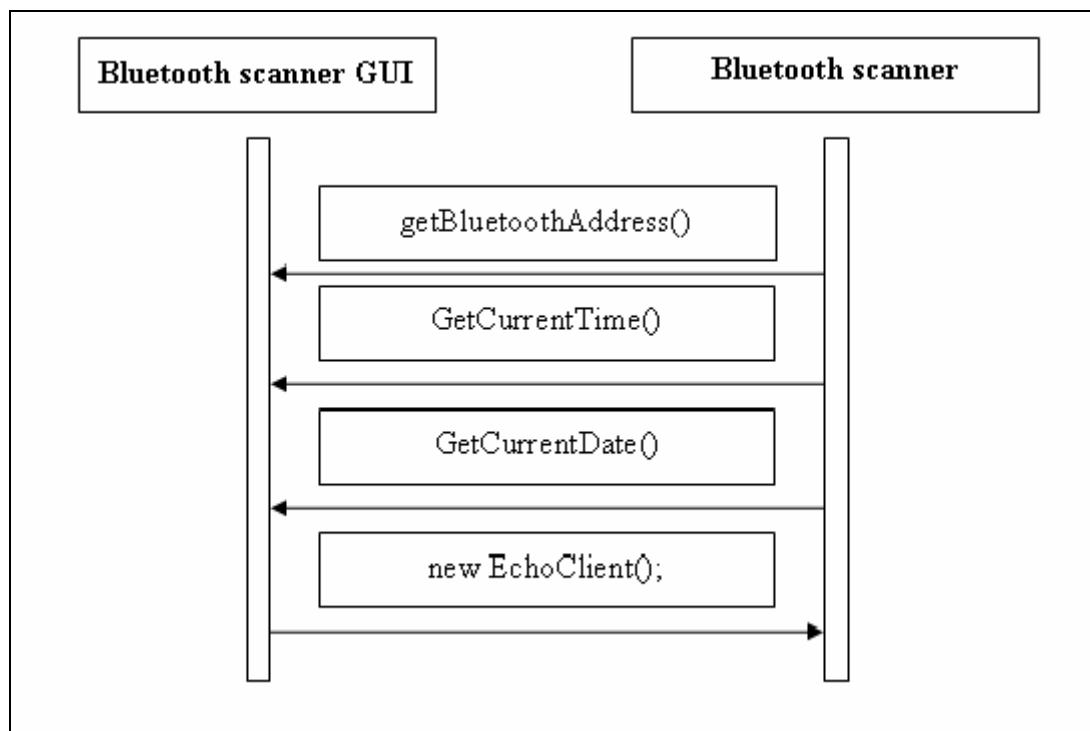


Figure 3.3: Sequence Diagram for Bluetooth Device Scanner GUI

Figure 3.2 shows the sequent diagram in the Bluetooth scanner GUI. This sequence happen when the Start button is pressed. When the button is pressed, the GUI program will call the Bluetooth device scanner program. The scanner program will run and the server will started and wait for client (phone) request. The result from the scanning process will be send back to the GUI and display it in the text area.

3.2.4 Database

For storing details about busses data likes the destinations, time of departure or arriving in this project, MySql database is choose. To monitoring the data, a GUI for database is created. Before creating the GUI for database, a database needs to set up first. Below is the step to install MySql database in window workstation.

- Download mysql-5.0.27-win32 installer from MySql website [6].
- Extract the .zip file and double click setupmysql.EXE
- Follow the instruction and click finish when the installation is complete.
- Find the MySql Command Line Client.
- Reader will be prompt to fill the password to the database. The reader can set the password in the installation process. Enter the password and press enter.
- The MySql environment is already set up in the workstation.

3.2.4.1 Connecting the IDE with MySql database.

To build the GUI for the database, we need to connect the database with the IDE. Because of the process is a little bit tricky, the step to connect the database with the IDE will be explained in **Appendix III**.

After MySql is connected to NetBeans 5.5 IDE, a table to store the data must be created before we can display the data to the GUI. This is the simple way to display all the data from database. Step to create the table in the database also available in **Appendix III**.

3.2.4.2 Build the database GUI

Database GUI will use same frame but in the different panel. Create the new panel and start building the database GUI. The GUI for database will have 3 panels and 1 table. Each panel will represent add, edit and delete data to database.

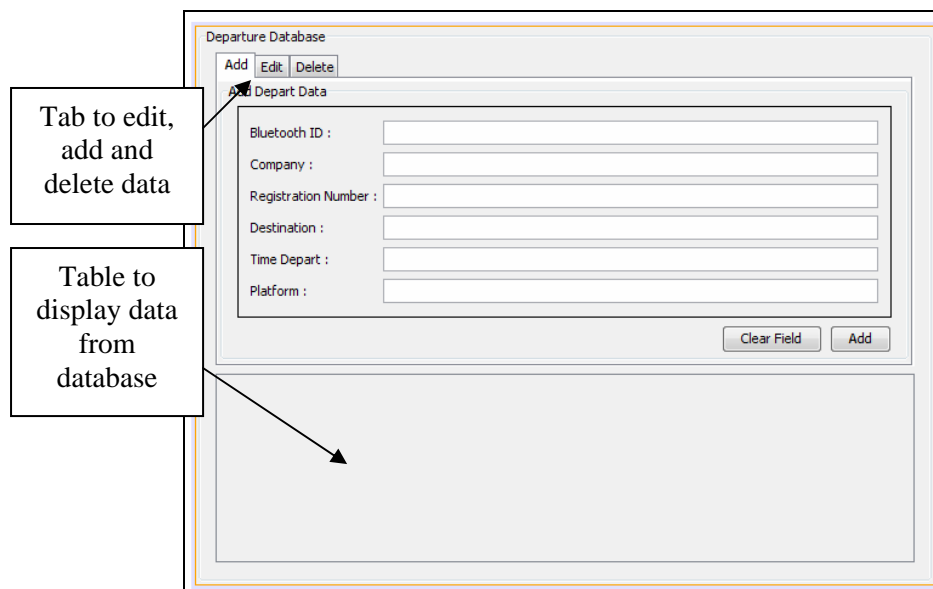


Figure 3.4: Sample of Database GUI

3.2.4.2.1 Add data to database

To add data to database using GUI should be done by choose the Add tab.

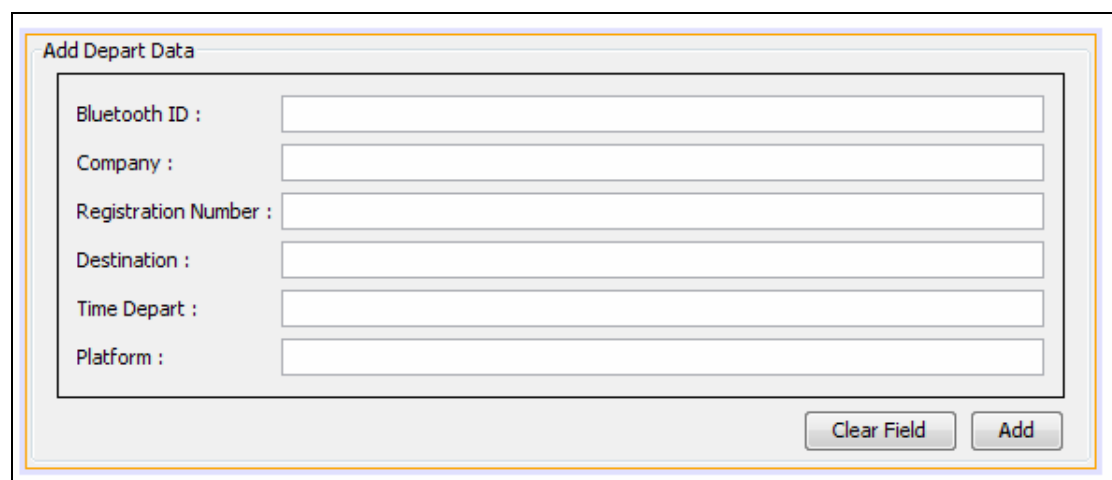


Figure 3.5: Add Data to Database GUI

The "addActionPerformed" method is created in the source file. Here is the code to perform this action.

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {  
    String dbluetoothID=jTextField29.getText();  
    String company=jTextField4.getText();  
    String regno=jTextField5.getText();  
    String destination=jTextField6.getText();  
    String timedepart=jTextField7.getText();  
    String platform=jTextField8.getText();  
  
    String insertStr="";  
  
    try{  
        insertStr="insert into depart (dbluetoothID, company, regno, destination,  
timedepart, platform ) values ("  
            +quote(dbluetoothID)+","  
            +quote(company)+","  
            +quote(regno)+","  
            +quote(destination)+","  
            +quote(timedepart)+","  
            +quote(platform)  
            +")";  
  
        int done=stmt.executeUpdate(insertStr);  
        if (jTextField29.getText()== "")  
            getContentPane().removeAll();  
        initComponents();  
    }  
    catch(Exception e){  
        e.printStackTrace();  
    }  
}
```

3.2.4.2.2 Edit data from database

To edit data from the database should be done in edit tab.

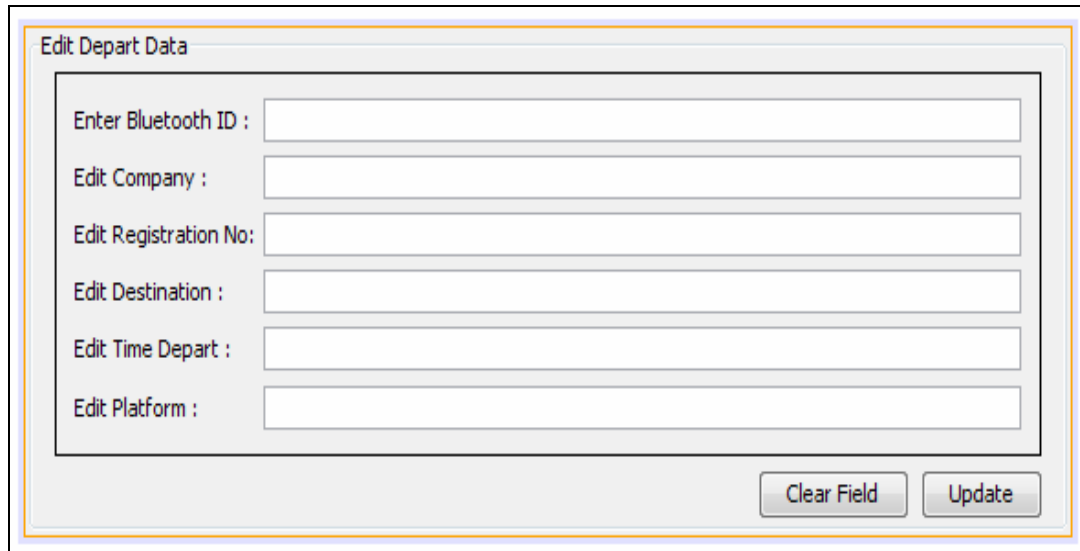


Figure 3.6: Edit Data from Database GUI

The code for the Update button will be shown below

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {  
    String dbluetoothID=jTextField9.getText();  
    String company=jTextField10.getText();  
    String regno=jTextField25.getText();  
    String destination=jTextField26.getText();  
    String timedepart=jTextField27.getText();  
    String platform=jTextField28.getText();  
  
    String insertStr="";  
  
    try{  
        insertStr="update depart set " +  
            "company="+quotate(company)+  
            ",regno=" +quotate(regno)+  
            ",destination="+quotate(destination)+
```

```

        ",timedepart="+quotate(timedepart)+
        ",platform="+quotate(platform)+
        "where dbluetoothID =" +quotate(dbluetoothID)+"";

int done=stmt.executeUpdate(insertStr);

getContentPane().removeAll();
initComponents();

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

```

3.2.4.2.3 Delete data from database

To delete the data from the database will be done in delete tab.

The image shows a GUI window with a light gray background and a thin border. At the top left, the text "Delete Depart Data By:" is displayed. Below this, there are two input fields. The first is labeled "Bluetooth ID :" and the second is labeled "Registration Number :". Between these two fields, the word "OR" is centered. At the bottom right of the window, there are two buttons: "Clear Field" and "Delete".

Figure 3.7: GUI for Delete Data from Database

The delete action will be performed when the delete button is pressed. The code for this action is shown below

```

private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {
    String dbluetoothID=jTextField11.getText();
    String regno=jTextField12.getText();

    String insertStr="";
    String insertStr1="";

    try{
        insertStr="delete from depart where dbluetoothID ="
            +quotate(dbluetoothID)
            +"";

        insertStr1="delete from depart where regno ="
            +quotate(regno)
            +"";

        int done=stmt.executeUpdate(insertStr);
        int done1=stmt.executeUpdate(insertStr1);

        getContentPane().removeAll();
        initComponents();

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

```

For arrive database the same method is used. For the full source code can be view in the **Appendix II**.

3.2.4.2.3 Database Table

The usage of the database table is to show the data from the database. This table is link to add, edit and delete panel. If a data is added the data in the table also change. The Figure 3.14 show the table when data from database in called.

BluetoothID	Company	Regno	Destination	Time Depart	Platform
000e6db1ac04	kukum	JHJ9981	Kangar	08:30	Platform 1
0012d13f4ac2	kukum	nbn9981	kubang gajah	08:00	platform 3
001370285bb9	kukum	mbg9981	automart	07:45	Platform 2
0016b8c2503f	kukum	bha9981	muhibah	08:00	platform 2

Figure 3.8: Table show the data from Departure table from Database

To set up the table to receive the data from database, a program to connect directly to database is created. The full source code for this program can be view in **Appendix II**.

3.2.4.3 Information Panel

Information panel is a place to inform the passenger about their journey information. In this panel, the data from database is called and will be send to client and display it on the hand phone screen. This panel will get the Bluetooth address from the scanner and automatically find the data from the database.

The source code for this panel can be view in **Appendix II**. Below is a screenshot of the information panel. The arriving and departure information panel share the same design and function, but will be run and display in the different screen.

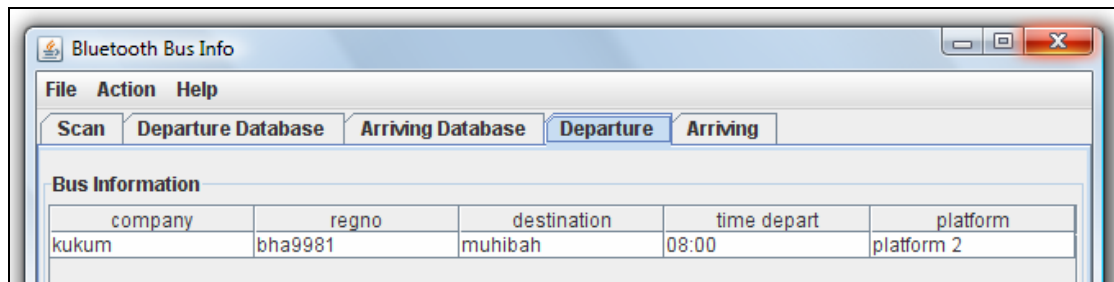


Figure 3.9: Departure Information Panel

3.3 Bluetooth client program

All the process above is a Bluetooth bus info server side program. The server side provide the client the information to be displayed. The client will request the server about the information of the busses. The server will send them a set of information to the client using Bluetooth connection. In this section the process to build the client side will be explain.

3.3.1 Introduction to J2ME application

Java Micro Edition is Java for small electronic device that support the Java environment. Java™ Platform, Micro Edition (Java ME) is the most ubiquitous application platform for mobile devices across the globe. It provides a robust, flexible environment for applications running on a broad range of other embedded devices, such as mobile phones, PDAs, TV set-top boxes, and printers. The Java ME platform includes flexible user interfaces, a robust security model, a broad range of built-in network protocols, and extensive support for networked and offline applications that can be downloaded dynamically. Applications based on Java ME software are portable across a wide range of devices.

The Java ME platform is deployed on billions of devices, supported by leading tool vendors, and used by companies worldwide. In short, it is the platform of choice for today's consumer and embedded devices.

To build a wireless application on the hand phone, Sun Wireless Toolkit formerly known as J2ME is used. The Sun Wireless Toolkit is a set of tools for creating Java applications that run on devices compliant with the Java Technology for the Wireless Industry (JTWI, JSR 185) specification and the Mobile Service Architecture (MSA, JSR 248) specification. It consists of build tools, utilities, and a device emulator. For this project, the application on the hand phone will use the Java for CLDC and MIDP 2.0 that fit with the hand phone's hardware capabilities. For more detail about the CLDC and MIDP 2.0, please refer to this site [7].

3.3.2 Build The Application on mobile phone

To build the application for phone, we need to download the Wireless Toolkit CLDC from Sun Microsystems site. The Sun Microsystems has built a wireless toolkit that supports the NetBeans software. That means the developer can use the NetBeans software to build the application for the phone. To NetBeans, the Wireless Toolkit CLDC is known as Mobility Packed Extension 5.5. This extension can be downloaded from Sun Microsystems and NetBeans site [8]. Source code for phone side program can be view in **Appendix IV**.

3.3.2.1 Flow Chart client part (on phone)

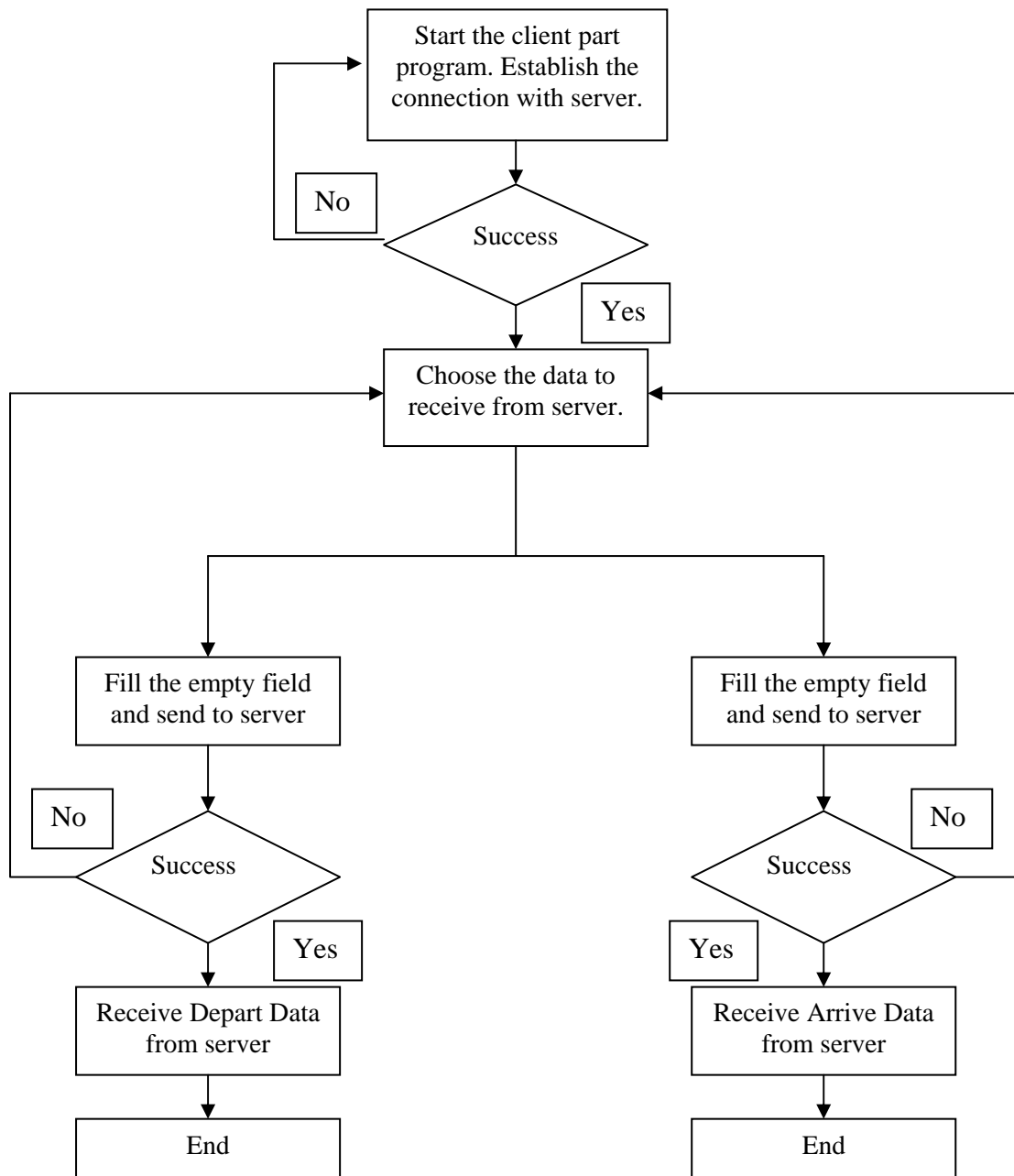


Figure 3.10 Flow chart of client (phone) program.

3.3.2.2 Program to connect with server.

Bluetooth Bus Info Client is passed the server's service record, which it uses to create a RFCOMM stream connection. The connection is mapped to an OutputStream and InputStream for sending and receiving messages. The opening of the connection is carried out in a thread since Connector.open() may block for a long period, and that shouldn't affect the GUI. Step to develop MIDP in NetBeans can be view in **Appendix V**.

```
// globals
private ServiceRecord servRecord;
private ClientForm clientForm;
private StreamConnection conn; // for the server
private InputStream in; // stream from server
private OutputStream out; // stream to server
private boolean isClosed = true;
// is the connection to the server closed?
public void run()
{
// get a URL for the service
String servURL = servRecord.getConnectionURL(
ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
if (servURL != null) {
clientForm.setStatus("Found Echo Server URL");
try {
// connect to the server, and extract IO streams
conn = (StreamConnection) Connector.open(servURL);
out = conn.openOutputStream();
in = conn.openInputStream();
clientForm.setStatus("Connected to Echo Server");
clientForm.setEnabled(true); // communication allowed
isClosed = false; // i.e. the connection is open

{ System.out.println(ex);
clientForm.setStatus("Connection Failed");
}
}
else
clientForm.setStatus("No Service Found");
} // end of run()
```

The first argument of the call to `ServiceRecord.getConnectionURL()` specifies the level of security for the connection. For this project, level of security is set for no security. The second argument relates to the master-slave communications protocol at the Bluetooth level, and should usually be set to false.

Once the connection URL has been extracted from the service record, a stream connection is obtained with `Connector.open()`, and an `InputStream` and `OutputStream` are layered on top of it. I use `InputStream` and `OutputStream` for the same reason as in the server additional message passing functionality is easily implemented with `read()` and `write()`.

3.3.2.3 Request data from server

A message is sent out when `ClientForm` calls `echoMessage()`. The method waits for an response then returns it to `ClientForm`.

```
public String echoMessage(String msg)
{
    if (isClosed) {
        disableClient("No Connection to Server");
        return null;
    }
    if ((msg == null) || (msg.trim().equals("")))
        return "Empty input message";
    else {
        if (sendMessage(msg)) { // message sent ok
            String response = readData(); // wait for response
            if (response == null) {
                disableClient("Server Terminated Link");
                return null;
            }
            else // there was a response
                return response;
        }
        else { // unable to send message
            disableClient("Connection Lost");
            return null;
        }
    }
} // end of echoMessage()
```

If there's a problem, the client is notified using `disableClient()`, and `null` is returned. The `readData()` and `sendMessage()` methods employed in `echoMessage()` are the same as the ones used in `ThreadedEchoHandler`.

3.3.2.4 Closing Down

`closeDown()` sends a "bye\$\$" message to the server, then closes the connection.

```
public void closeDown()
{
    if (!isClosed) {
        sendMessage("bye$$"); // tell server that client is leaving
        try {
            if (conn != null) {
                in.close();
                out.close();
                conn.close();
            }
        }
        catch (IOException e)
        { System.out.println(e); }
        isClosed = true;
    }
} // end of closeDown();
```

CHAPTER 4

IMPLEMENTATION ISSUE

4.0 Introduction

This chapter will discuss about the implementation issue from the Java Bluetooth API classes. Bluetooth API will be used to develop Bluetooth device scanner and connection between PC with mobile phone. Another API used in this program is Java Utilities. Java Utilities provides the method to get the date and time to be implementing to this software. The details about these classes will be discuss below.

4.1 Using the Local Device methods.

Local Device class is the method to extract all the data about the Bluetooth data from the local host. In this program, local device will be the built in Bluetooth device that attach to the workstation. Program below show how to implement the local device class to the Bluetooth Device Scanner.

```
import javax.bluetooth.*;
:
:
    try {
// Get the local device
LocalDevice localDevice = LocalDevice.getLocalDevice();
```

```

// Get the DiscoveryAgent object for device and service discovery
discoveryAgent = localDevice.getDiscoveryAgent();
// Start the device inquiry
discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
}
catch (Exception e)
{System.out.println(e);}

```

4.2 Using the Remote Device methods.

In the Java Bluetooth API, remote device is the Bluetooth devices that try to make connection with local device. This class will extract data likes Bluetooth Address, Bluetooth friendly name and Bluetooth device class.

```

public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
    String bt_address = new String(btDevice.getBluetoothAddress());
    System.out.println(btDevice.getBluetoothAddress());
}

```

4.3 Using the java.util methods to get local time

Java Utilities class can be used to get the local time and date from the workstation. The method to get the local time and date is show below

```

import java.util.*;

private String GetCurrentTime()
{
    String[] ids = TimeZone.getAvailableIDs(60 * 60 * 1000);
    if (ids.length != 0)
    {

```

```

        SimpleTimeZone gmt = new SimpleTimeZone(60 * 60 *
        1000, ids[0]);
        Calendar calendar = new
        GregorianCalendar(TimeZone.getTimeZone("Asia/Kuala_Lu
        mpur"));
    java.util.Date today = new java.util.Date();
    calendar.setTime(today);

    String patternDate = new String("EEE-MMM-dd");
    String patternTime = new String("H:mm:ss");

        SimpleDateFormat formatter = new
        SimpleDateFormat(patternTime, Locale.ROOT);
    formatter.setCalendar(calendar);
    return (String)(formatter.format(today));
    }
    return null;
}

```

4.4 Using the java.util to get local date

```

import java.util.*;
private String GetCurrentDate()
{
    String[] ids = TimeZone.getAvailableIDs(60 * 60 * 1000);

    if (ids.length != 0)
    {
        SimpleTimeZone gmt = new SimpleTimeZone(60 * 60 * 1000, ids[0]);
        Calendar calendar = new GregorianCalendar(gmt);
        java.util.Date today = new java.util.Date();
        calendar.setTime(today);
    }
}

```

```

String patternDate = new String("dd.MM.yyyy");

SimpleDateFormat formatter = new SimpleDateFormat(patternDate,
        Locale.ROOT);
formatter.setCalendar(calendar);

return (String)(formatter.format(today));
}

return null;
}

```

4.5 Server development

Bluetooth Bus Info Server is a threaded RFCOMM-based echo service, identified by a UUID (a unique Bluetooth ID) and service name ("echoserver"). The waiting for client connections is done in a thread so that it doesn't cause the toplevel GUI to block. When a client does connect, a ThreadedEchoHandler thread is spawned to deal with it.

```

public EchoServer()
{
this.ecm = ecm;
handlers = new Vector();
try { // make the server's device discoverable
LocalDevice local = LocalDevice.getLocalDevice();
local.setDiscoverable(DiscoveryAgent.GIAC);
}
catch (BluetoothStateException e) {
System.out.println(e);
return;
}
/* Create a RFCOMM connection notifier for the server, with
the given UUID and name. This also creates a service
record. */
try {
server = (StreamConnectionNotifier) Connector.open(
"btspp://localhost:" + UUID_STRING +
";name=" + SERVICE_NAME);
}
}

```



```
}  
catch (IOException e) {  
System.out.println(e);  
return;  
}  
} // end of EchoServer()
```

The server's device must be discoverable by a client:

```
LocalDevice local = LocalDevice.getLocalDevice();  
local.setDiscoverable(DiscoveryAgent.GIAC);
```

The `DiscoveryAgent.GIAC` (General/Unlimited Inquiry Access Code) constant means that all remote devices (all the clients) will be able to find the device. There's also a `DiscoveryAgent.LIAC` constant which limits the device's visibility. The RFCOMM stream connection offered by the server requires a suitably formatted URL. The basic format is: `btspp://<hostname>:<UUID>;<parameters>`

The UUID field is a unique 128-bit identifier representing the service; I utilize a 32 digit hexadecimal string (each hex digit uses 4 bits). The URL's parameters are "`<name>=<value>`" pairs, separated by semicolons. Typical `<name>` values are "name" for the service name (used here), and security parameters such as "authenticate", "authorize", and "encrypt". In WTK 2.2., the value of the "name" string must be in lowercase or a client won't recognize the service name at discovery time.

The other main URL type is for L2CAP connections, where messages are sent as packets, in a similar style to UDP datagram. The creation of the `StreamConnectionNotifier` instance, server, by the `Connector.open()` call also generates an implicit service record. The record is a description of the Bluetooth service as a set of (id, value) attributes. It can be accessed by calling `LocalDevice.getRecord(): ServiceRecord record = local.getRecord(server);` The `ServiceRecord` class offers get/set methods for accessing and changing a record's attributes.

4.6 Waiting for a Client

EchoServer waits for client connections in a separate thread (EchoServer implements Thread) so that the enclosing GUI isn't blocked.

```
// global
private boolean isRunning = false;
public void run()
// Wait for client connections, creating a handler for each one
{
isRunning = true;
try {
while (isRunning) {
StreamConnection conn = server.acceptAndOpen();
ThreadedEchoHandler hand = new ThreadedEchoHandler(conn, ecm);
handlers.addElement(hand);
hand.start();
}
}
catch (IOException e)
{ System.out.println(e); }
} // run()
```

The call to `acceptAndOpen()` makes the server block until a client connection arrives, and also adds the server's service record to the device's Service Discovery Database (SDDB). When a client carries out device and service discovery it contacts the SDDBs of the devices that it's investigating. When a client connection is made, `acceptAndOpen()` returns a MIDP `StreamConnection` object, which is passed to a `ThreadedEchoHandler` instance so it can deal with the client communication.

4.7 Connecting to the Client

`ThreadedEchoHandler`'s `run()` method creates input and output streams, and starts processing the client's messages.

```
// globals
private InputStream in;
private OutputStream out;
public void run()
{
ecm.incrCount(); /
try {
```

```

// Get I/O streams from the stream connection
in = conn.openInputStream();
out = conn.openOutputStream();
processClient();
}
catch(Exception e)
{ System.out.println(e); }
ecm.decrCount(); // remove this handler from the top-level count
} // end of run()

```

An `InputStream` and `OutputStream` are extracted from the `StreamConnection` instance, and used in `processClient()`. The use of `run()` means that any I/O blocking will be in a separate thread from the server and its GUI. It's possible to map a `DataInputStream` and `DataOutputStream` to the `StreamConnection` instance, so that basic Java data types (integers, floats, doubles, strings) can be read and written. I've used `InputStream` and `OutputStream` because their byte-based `read()` and `write()` methods can be easily utilized as 'building blocks' for implementing different forms of message processing.

4.8 Talking to a Client

The `processClient()` method waits for a message to arrive from the client, converts it to uppercase, and sends it back. If the message is "bye\$\$", then the client wants to close the link.

```

private void processClient()
{
isRunning = true;
String line;
while (isRunning) {
if((line = readData()) == null)
isRunning = false;
else { // there was some input
if (line.trim().equals("bye$$"))
isRunning = false;
else {
ecm.showMessage(clientName + ": " + line);
// show in the GUI
String upper = line.trim().toUpperCase();
if (isRunning)

```

```

sendMessage(upper);
}
}
}
System.out.println("Handler finished");
} // end of processClient()

```

The messy details of reading a message are hidden inside `readData()`, which either returns the message as a string, or null if there's been a problem. A message is transmitted with `sendMessage()`.

4.9 Reading a Message

When a client sends a message to the handler ("hello"), it is actually sent as a stream of bytes prefixed with its length ("5hello"). The number is encoded in a single byte, which puts an upper limit on the message's length of 255 characters. Since a message always begins with its length, `readData()` can use that value to constrain the number of bytes it reads from the input stream.

```

private String readData()
{
    byte[] data = null;
    try {
        int len = in.read(); // get the message length
        if (len <= 0) {
            System.out.println("Message Length Error");
            return null;
        }
        data = new byte[len];
        len = 0;
        // read the message, perhaps requiring several read() calls
        while (len != data.length) {
            int ch = in.read(data, len, data.length - len);
            if (ch == -1) {
                System.out.println("Message Read Error");
                return null;
            }
            len += ch;
        }
    }
    catch (IOException e)
    { System.out.println("readData(): " + e);
      return null;
    }
}

```

```
}  
return new String(data); // convert byte[] to String  
} // end of readData()
```

`InputStream.read()` is called repeatedly until the necessary number of bytes have been obtained. The bytes are converted into a `String`, and returned; the message length is discarded.

4.10 Sending a Message

`sendMessage()` adds the message's length to the front of a message, and it is sent out as a sequence of bytes:

```
private boolean sendMessage(String msg)  
{  
    try {  
        out.write(msg.length());  
        out.write(msg.getBytes());  
        return true;  
    }  
    catch (Exception e)  
    { System.out.println("sendMessage(): " + e);  
      return false;  
    }  
}
```

4.11 Closing Down the Handler

The server terminates the handler by calling its `closeDown()` method. The input and output streams are closed first, then the underlying `StreamConnection`.

```
public void closeDown()
{
    System.out.println("Close down echo handler");
    isRunning = false;
    try {
        if (conn != null) {
            in.close();
            out.close();
            conn.close();
        }
    }
    catch (IOException e)
    { System.out.println(e); }
}
```

`isRunning` is set to `false`, which will cause the I/O loop in `processClient()` to finish.