PDA User Interface for a Data Acquisition System

Thomas Rodriguez

CEN 3213 - Embedded Systems Programming - Dr. Zalewski

Florida Gulf Coast University

March 28, 2007
Section 1. Introduction

We have all experienced it, the anxious feeling waiting for the next hurricane season update. Did the system get stronger? Will it affect us? Thanks to one simple item, that many of us take for granted, can we know something about the tropical systems: Data Acquisition. “Data acquisition is the sampling of the real world to generate data that can be manipulated by a computer.” (Wikipedia). It is basically acquiring (the word acquisition is based on acquire) data from the world in real-time. What do we use the data for? To answer this question, I post another question: why do we need a history class? The answer is to predict or hypothesize the near future. Will I need a jacket today? Will I need a raincoat? By acquiring data in advance and study its patterns we can predict to a certain point what will happen soon. Here is an example of one such system.

A thermometer outside could be one instrument of a data acquisition system, with the other part being at least a computer that is connected to it to record the information. Thanks to this information the computer records almost continuously, we are able to “manipulate”, or interpret, the data for calculations, estimations, and observations. Since it does this almost incessantly, it becomes part of a special type of computer system, called and Embedded System.

An Embedded System is a computer system that is part of a larger system. For example, that same thermometer is in most cases not alone, instead it is part of a small or large weather station. This whole system may include a pressure gauge, humidity gauge, and much more. Since this whole weather system needs to be precise and accurate, the programming made for acquiring, storing, and displaying data has to be carefully thought out to not overload the computer with five or six inputs when its processor can only grab one at a time. There are several programming constructs that may help us handle the inputs in an appropriate manner. The most used and popular is thread usage. But this project does not concentrate on the use of threads so much as the minimal use of the processor for completing and displaying tasks.

Since almost every Real-Time Embedded System has some degree of Data Acquisition, it is easy to see why programming for a system like this requires storing data in an efficient way. And since data acquisition deals with real world data, which in most cases is messy and sometimes random, we must cautiously approach a program model to make sure we interpret the data in the correct form and (and here is the point) in the most efficient way possible. We do not want a relatively small computer to
crash when we are trying to acquire vital information. So this project concentrates on how to best use the computer’s processor, whether it be a fast one or a slow one.

Section 2. Problem Specification

Section 2.1 Device Description

Let’s begin with the device that we will be using for this project on data acquisition. For reading and displaying information we will be using a PDA. The reason is because that way we are able to read information about RFID tags while being in the field. The specifications on the PDA, which in this case is a Dell Axim x50, are show in Table 2.1.1. A picture of it is displayed in Figure 2.1.2.

Axim X50 Advanced

<table>
<thead>
<tr>
<th>Processor:</th>
<th>520 MHz Intel XScale PXA270 with WMMX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display:</td>
<td>3.5 inch 240 x 320 QVGA screen</td>
</tr>
<tr>
<td>Memory:</td>
<td>64 MB RAM (55 MB available); 128 MB flash ROM (90 MB available)</td>
</tr>
<tr>
<td>Size &amp; Weight:</td>
<td>4.68&quot; x 2.87&quot; x 0.63&quot;, 167 grams (5.9 ounces)</td>
</tr>
<tr>
<td>Expansion:</td>
<td>CompactFlash Type 2 and SDIO slots</td>
</tr>
<tr>
<td>Docking:</td>
<td>36-pin connector, standard USB cradle</td>
</tr>
<tr>
<td>Communication:</td>
<td>Integrated Bluetooth 1.2 and 802.11b WiFi</td>
</tr>
<tr>
<td>Audio:</td>
<td>Internal monaural speaker; microphone; 3.5mm headphone/headset jack</td>
</tr>
<tr>
<td>Battery:</td>
<td>Standard 3.7 volt, 1100 milliamp-hour Lithium-Ion battery; optional 2200 mAh Lilon battery</td>
</tr>
<tr>
<td>Input:</td>
<td>6 remappable application buttons; 5-way directional pad; touchscreen</td>
</tr>
<tr>
<td>Software:</td>
<td>Windows Media Player 10, 802.1x security client, Outlook 2002</td>
</tr>
</tbody>
</table>

*Table 2.1.1(From brighthand.com)*
Section 2.2 Problem Description

The data acquisition part of this project has already been completed by other students in previous semesters. The concentration of this project is making a Graphical User Interface (GUI) for reading and displaying data that a data reader should bring in. Our friends at Cante Anandromous Fish Research Center (CAFRC) and the United States Geological Survey (USGS) have allowed us to use a system in our vicinity as a starting point to test our data acquisition project. The following is a basic summary of Alex Haro’s “Manual for Operation of TIRFID PIT Tag Systems.”

General Connections for Single Reader/Datalogger TIRFID System

The TIRFID System is a system that acquires live data of fish tagged with an RFID tag known as a PIT tag (Passive Integrated Transponder). This tag is placed under the skin of the fish and usually lasts (battery) about the life of the fish. This tag transmits a radio frequency that the RFID (Radio Frequency IDentification) system will read and record every time a fish passes...
through a constructed antenna. Each PIT tag has a unique 16 digit tag code which is recorded as part of the data transmitted. How is the system connected to record this data?

The system starts with an antenna, a looped wire inside of a constructed PVC pipe, usually submerged underwater. This antenna is then connected to a Tuning Module which is connected using a twinaxial cable to the “Reader”, which is composed of the Reader Module and Control Modules. This is connected to a 12 VDC battery so that it may run. The “Reader” is then connected to a DataLogger, in our case the PDA. Figure 2.2.1 shows the basic connection of this system.

Our system is located in Charlotte Harbor and has two different antennas. Figure 2.2.2 shows us where the TIRFID system antennas are located.

Our purpose then is to take this system and create a friendly user interface for the data logging stage. We must remember, though, our system is an embedded one, which requires us to “keep it simple” and not use too much of the processor. Luckily, our PDA does not have a terrible processor, the processor usage is not as important as in other smaller and slower processors with this PDA. But our goal is to make a working GUI for any handheld device, with a good or bad processor. So the user interface must still be as processor efficient as possible.
Section 3. Design Specification

Section 3.1 Design Description

Program writing and compiling is very difficult and limited if it is all done straight from the PDA. For this reason, many embedded programmers use other computers to write and compile the necessary code for the program and create an executable file of some sort. This file is then put on the PDA or other device that can then run the executable code. This is very useful and convenient. But not all programming languages can do this so easily. Luckily, Java is one of those languages that can.

Java is a remarkable programming language known for two things in particular: its Object-Oriented Features and its Portability. The Object-Oriented features of this high level programming language allow us to program easily without having to think too much about computer specific necessities. The portability that the Java programming language provides is also very useful because it allows us to compile on one computer and run the generated .class files on another computer. Java uses something called the JVM, or the Java Virtual Machine, to read the .class file and translate it into machine code. This machine code is always different in each computer, but the JVM helps us by converting non-machine-specific code to machine specific code. Therefore, we can put this .class file virtually on any computer, PDA, or other device. But is there a JVM for the PDA?

There are a few. One of them is J2ME, ME for Micro Edition. This is a compact edition of the J2SE or J2EE. This basically does not contain many Java libraries for graphics and multithreading capabilities. The reason is because most PDA’s that the Java programs are running on are not capable of using them. Luckily, our PDA is a little advanced and can handle the graphics. So I will use another JVM that does include these libraries, and this one is called the Mysaifu JVM. I do not recommend one in specific, so you may use either of these two or of the many other JVM’s out on the market. Here is a link to download and use J2ME:

http://java.sun.com/javame/downloads/index.jsp

Here is a link to download and use the Mysaifu JVM:
To use Mysaifu, we must download the file onto the PDA under the My Documents folder. Once it’s there, simply open the Mysaifu JVM under Programs. The window that appears looks like Figure 3.1.1. Then type the name of the class file in the “Name” space. Now click on the button that says “Options”.

Figure 3.1.1

The window in Figure 3.1.2 will appear. The last text box in the list is titled “Current Directory”. Here click “Browse”. Now chose the folder that the .class file is in, which in this project will be in My Documents. Then click “OK” and then “Execute”.

Figure 3.1.2
Now the .class file should run. Now that we know the required materials and instructions to get our class file running, how is the GUI going to look like?

Our GUI has to fit best for our users, hence the term Graphical User Interface. I emailed Mr. Kirby Wolfe, the Senior Biologist on this project, and asked him what he would like to see in his GUI. He said that “it would be nice to see a count of how many tags have been read [and] a means to stop start the program.” So these are things that need to be included. The current executable file has a few parameters that it need so that it can work properly. Using some swing components, Figure 3.1.3 is how the GUI is going to somewhat look like.

![Figure 3.1.3](image)

The code is supposed to run the .exe file with the correct parameters in place in order for the system to work properly. How are we going to run an executable file from a Java program and passing the correct parameters?

The Java programming language allows us to run a windows executable file by simply using the Runtime class. First you must import it and then you may use it. Here is the basic idea using some pseudo code.

```java
try {
    Runtime rntm = Runtime.getRuntime();
    Process p = rntm.exec("Program.exe");
    InputStream in = p.getInputStream();
    OutputStream out = p.getOutputStream();
}
```
InputSream err = p.getErrorStram() ;
//do whatever you want
//some more code
p.destroy() ;
catch(Exception exc){/*handle exception*/}

This is what we have to do. Using the input stream correctly we can output our parameters at
the right places. Let’s jump into the implementation and testing phase.

Section 4. Implementation and Testing

Section 4.1 Implementation

The full implementation was a little difficult to complete because of many reasons. These
problems will be discussed later in the conclusion. My part of this system was to make a functional GUI
for the Logging program they currently have installed.

The Logging program is called DLOG6.exe. This executable file needs to be running to retrieve
the data from the reader. The pseudo code used before is basically what we need for the program to
work, without forgetting the required GUI parts. This includes frames, buttons, text fields, etc. Here is
the code for this GUI.

package my.contacteditor;

import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class LoggerGUI{
    private JFrame MainFrame = new JFrame("Logger");
    private JTextArea Header_Info;
    private JLabel Header_Info_Label;
    private JTextField File_Name;
    private JLabel File_Name_Label;
    private JTextField Site_Code;
    private JLabel Site_Code_Label;
    private JTextField Time_Delay;
    private JLabel Time_Delay_Label;
    private PrintWriter writer;
    private Container Running;
    private Container container;
    private JTextArea NewTags = new JTextArea(20,30);
Runtime rt = Runtime.getRuntime();
private Process p;

public LoggerGUI()
{
    try{
        p = rt.exec("cmd");
    }catch(IOException e){}
    init();
    initProcess();
}

public void init()
{
    Header_Info_Label = new JLabel("Tagfile Header Information:");
    Header_Info = new JTextArea(10, 40);
    Header_Info.setEditable(true);
    File_Name_Label = new JLabel("Enter filename (mmddyy.txt):" purchasers.
    File_Name = new JTextField(30);
    File_Name.addKeyListener(new KeyAdapter()
    {
        public void keyPressed(KeyEvent event) {
            if (event.getKeyCode() == KeyEvent.VK_ENTER)
            {
                if (writer != null)
                {
                    Header_Info.setText("\n");
                    writer.print(File_Name.getText() + "\n");
                    writer.flush();
                    File_Name.setText("\n");
                }
            }
        }
    });
    Site_Code_Label = new JLabel("Enter Site Code:" purchasers.
    Site_Code = new JTextField(10);
    Time_Delay_Label = new JLabel("Enter same tag information time delay (in seconds):" purchasers.
    Time_Delay = new JTextField(4);
    // The canvas purchasers.
    container = MainFrame.getContentPane();
    container.setLayout(new BoxLayout(container,BoxLayout.PAGE_AXIS));
    container.add(File_Name_Label);
    container.add(File_Name);
    container.add(Site_Code_Label);
    container.add(Site_Code);
    container.add(Time_Delay_Label);
    container.add(Time_Delay);
container.add(Header_Info_Label);
container.add(Header_Info);
// Button operations and code
B button = new B();
JButton Start = new JButton("Start");
Start.addActionListener(button);
container.add(Start);

// Setting up the MainFrame
MainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
MainFrame.pack();
File_Name.grabFocus();
MainFrame.setVisible(true);
}

private class B implements ActionListener{// The event listener for the button
public void actionPerformed(ActionEvent e){
    System.out.println("Button Pressed");
    MainFrame.setVisible(false); // First remove the original window.
    // Send the parameters
    writer.print(File_Name.getText() + \"\r\n\n\n");
    writer.flush();
    writer.print(Site_Code.getText() + \"\r\n\n\n");
    writer.flush();
    writer.print(Header_Info.getText() + \"\r\n\n\n");
    writer.flush();
    writer.print(Time_Delay.getText() + \"\r\n\n\n"); // This one is a problem. It can only take numbers.
    writer.flush();
    // The Second window's attributes
    JFrame SecondFrame = new JFrame("Data Logging...");
    Running = SecondFrame.getContentPane();// The canvas here is called Running
    Running.setLayout(new BoxLayout(Running,BoxLayout.PAGE_AXIS));
    JLabel info = new JLabel("New Tags:");
    NewTags.setEditable(false);
    JScrollPane scrolling = new JScrollPane(NewTags);// This is the area where the registered tags will appear.
    Running.add(info);
    Running.add(scrolling);
    SecondFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    SecondFrame.pack();
    SecondFrame.setVisible(true);
}

public void initProcess()
{
try
{
    p = rt.exec("cmd");
    writer = new PrintWriter(p.getOutputStream());
    // The idea of using threads was from a similar program from a forum at java.sun.com
    Thread thread1 = new Thread(new StreamReader(p.getErrorStream()));
    Thread thread2 = new Thread(new StreamReader(p.getInputStream()));
    thread1.start();
    thread2.start();
    System.out.println("Exit Value = " + p.waitFor());
}
catch (Exception ex)
{
    NewTags.append(ex.getMessage()); // In case there is a problem, append the message to the
    NewTags text area
}
}

public class StreamReader implements Runnable
{
    InputStream is;
    public StreamReader(InputStream is)
    {
        this.is = is;
    }

    public void run()
    {
        try
        {
            writer.print("DLOG6.exe" + "+\n"); // This is the first thing that the program needs to do, is send
            the text name of the Logger program, DLOG6.exe
            writer.flush();
            BufferedReader reader = new BufferedReader(new InputStreamReader(is));
            // For error checking, i leave this here, but commented out. This displays everything that the
            cmd displays.
            /*String data;
             while ((data = reader.readLine()) != null)
             {
                 NewTags.append(data + "+\n");
             }*/

            reader.close();
        }
        catch (IOException ioEx)
        {
            ioEx.printStackTrace();
        }
    }
}
In order to run the application on the PDA, you first need to move the DLOG6.exe and GUI.class files to the PDA using Microsoft ActiveSync USB connection. Then run the GUI.class file in the Mysaifu JVM as instructed above. The GUI should start. Enter your parameters and click “Start” (Figure 4.1.1). Normally, we would see a window like the one in Figure 4.1.2, but with a few lines of tag readings. The reason we don’t see it is because we are not connected to the reader.

![Logger](image)

*Figure 4.1.1*
Section 4.2 Testing

Like mentioned before, this is one of those sections that I cannot explain too much about. The reasons are many, which include our connection and our power source. The main reason is our connection.

The current system has an HP Palm Computer that connects through an adapter to the RS-232 9 pin plug of the RFID reader. This is an older plug that newer computers do not have. My laptop on which I designed and coded the GUI does not have an RS-232 plug. The Dell Axim PDA does not either; its only connections are USB and wireless. This could have been fixed by buying an adapter that could connect our PDA or our computer. But time and money is against us. I only had one day with the equipment because Mr. Wolfe needed it back. Although it was only one day, using the equipment for the RFID system was useful because at least I was able to see how the whole system should work. From here I had enough information to design a GUI for the DLOG6.exe application.

Power was another issue. If the PDA was installed and used for the system, we need a connection that would work and power for the PDA. The PDA should and will use more power than the current HP computer installed. And since the cables that connect the system to the battery are preconfigured for the HP power plug, we cannot test the PDA using the battery. This is a problem that will have to be tested by other students in other semesters.
Since I could not connect to the reader from the computer or the PDA, I only have a partially tested application. In theory, the application should be fully functional, but since I cannot test it thoroughly, it stays as only a theory.

Section 5. Conclusion and References

Section 5.1 Conclusion

There have been many problems along the way, but in the end, our system works wonderfully. I have a few ideas about how to solve some of these problems, especially our connections problem. How do we connect our PDA to the RFID reader? There are many solutions, but the most reasonable would be using the Bluetooth capabilities of the PDA. There is a Bluetooth plug for the RS-232 9 pin plug; the only downside to this is the price of the small item: anywhere between $120 and $130. This plug does not require any external power, so it makes our life easier. Now for the power for the PDA, we would probably reconfigure and rewire the power cable to the battery. We also need to verify the amount of power that the PDA uses in comparison to the HP Computer. Hopefully, this difference is not too much. After a bit of coding and lots of testing, we can conclude that this project can be a useful one. Although incomplete in its design look and feel, our program works efficiently to read and write to the original DLOG6.exe file from Mote Marine Systems. I leave the rest of this project in the hands of the other students after me.

Section 5.2 References


