UNIVERSITY OF HOUSTON
DEPARTMENT OF COMPUTER SCIENCE
MAY 2011

Part I

Hardware and Data Logger UI for the Arduino Project

Sujeet Kuchibhotla

Nripun Sredar

Shaunak Vairagare

A Project for Cyber Physical Systems

Under

Dr Nikolaos Tsekos

Overview:

Arduino Fio is a microcontroller board intended for wireless applications. It has 14 digital I/O pins and 8 analog inputs. The main aim of this project is to attach sensors like temperature sensors, soil moisture sensors and other such analog or digital sensors to the Arduino Fio boards and be able to get values from these sensors to a control station wirelessly. The programming of the Arduino Fio boards is also done wirelessly. Many such Arduino Fio boards can thus be deployed to create a network of sensor stations which communicate wirelessly with the control station.

Programming Arduino:

Arduino can be programmed in two ways as shown below:

1. Preburned with bootloader
    - Two ways to upload sketches(s/w written using Arduino)
        1. FTDI – SUB cable
        2. Wirelessly over pair of XBee radios
2. Bypass the bootloader and program the ATmega328 with an external programmer

We have programmed the Arduino Fio wirelessly in this project. The configuration steps needed to start wirelessly programming the Arduino Fio using XBee's are listed in the next section.

Configuring XBee:

The typical setup for this project is a control station with an XBee unit and a remote Arduino Fio with a XBee unit. The control station XBee unit is used to wirelessly program the remote Arduino board. These XBee's are the only communication channel. Both the XBee units viz. the control station XBee and the remote XBee need to be configured differently so that they can communicate with each other. This configuration can be done using the FioXBeeConfigTool. This tool can be obtained at the following URL:

http://funnel.googlecode.com/files/XBeeConfigTool.zip

The configuration steps are listed below:

1. Connect the Programming Radio to the PC using the XBee adapter
2. Install the driver
3. Go to Device Manager. Right Click USB COM Port 9 and select properties. Select Port Settings Tab. Click Advanced. Check mark Set RTS On Close.
4. Open FioXBeeConfigTool
5. Select COM9

6. Select Programming radio
7. Select 57600 Baud rate
8. Set the PAN ID to 1234
9. MY ID – 0000
10. DL ID – FFFF

Click Configure/ **Message : Configured Successfully**

1. Connect the Arduino Fio Radio to the PC using the XBee adapter
2. Open FioXBeeConfigTool
3. Select COM9
4. Select Arduino Fio radio
5. Select 57600 Baud rate
6. Set the PAN ID to 1234
7. MY ID – 0001
8. DL ID – 0000
9. Click Configure/ **Message : Configured Successfully**

Sensors:

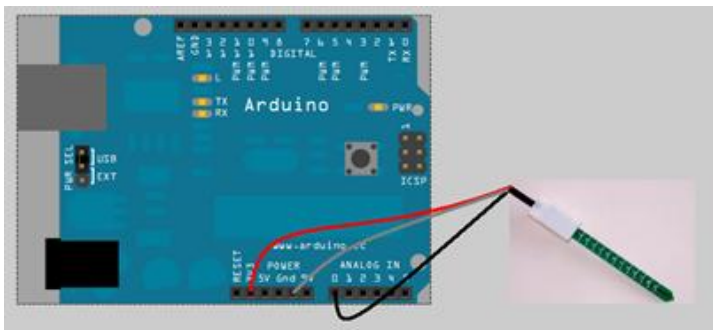
Arduino Fio supports both digital inputs and analog inputs. We have used two analog sensors for our purpose. We have used an analog temperature sensor and a moisture sensor to connect to the Arduino Fio board.

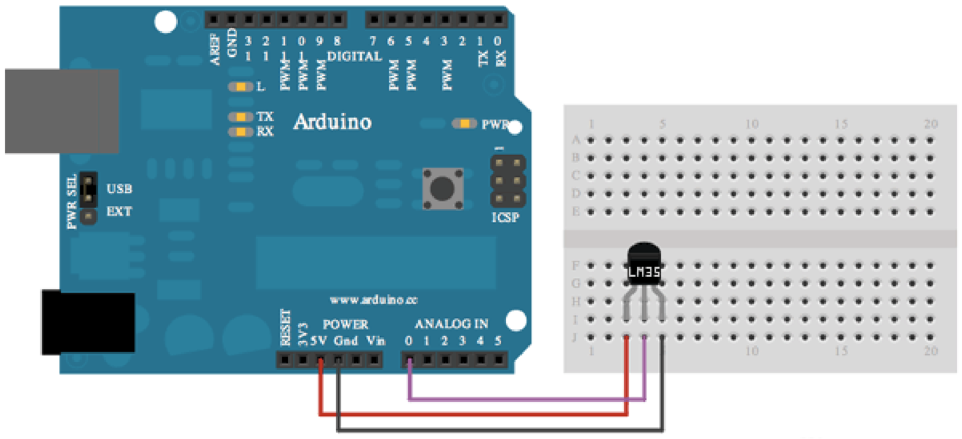
Temperature Sensor: LM35Z (http://www.national.com/mpf/LM/LM35.html#Overview)

Moisture Sensor: VG400 (http://www.vegetronix.com)

Circuit Diagrams:

The sensors are connected to the Arduino Fio board as shown in the figures below:





Arduino Programming:

Arduino can be programmed wirelessly by the following steps:

1. Open Arduino IDE
2. Select Tools/Board/Arduino Fio
3. Select Tools/Serial Port/ COM9 (Should be the same port used while configuring XBee)
4. Go to File/Examples/Basics->Blink (Or any other custom sketch)
5. Change the delay of the blink duration
6. Click Save. Compile. Upload. If upload is successful, the message "Done uploading" is displayed.

Program:

There are three software modules in this project.

1. Sketches flashed onto the Arduino Fio board (Arduino program)
2. Script which communicates with Arduino Fio board (Ruby script)
3. Web-app that displays associated data content (Ruby on Rails app)

Arduino Program:

This program is flashed on to each Arduino Fio Board. The program reads input on the serial port and upon receiving appropriate query input, it sends out requested data over the serial port. The program expects input in a pre-defined format as shown below:

Command Format used to query Arduino:

Command : $#1#temp#moist@ --> Arduino with id #1 will return temperature and moisture in format $#1#23#513@

Command : $#1#temp@ --> Arduino with id #1 will return temperature in format $#1#23@

Command : $#2#moist@ --> Arduino with id #2 will return moisture in format $#1#513@

Temperature is returned in Celsius and moisture is returned in units ranging from 0 to 1024 depending on degree of saturation.

Ruby Script:

The Ruby script is used to send queries to the Fio boards to obtain sensor data from that board. The command format for the queries is as shown in the previous section.

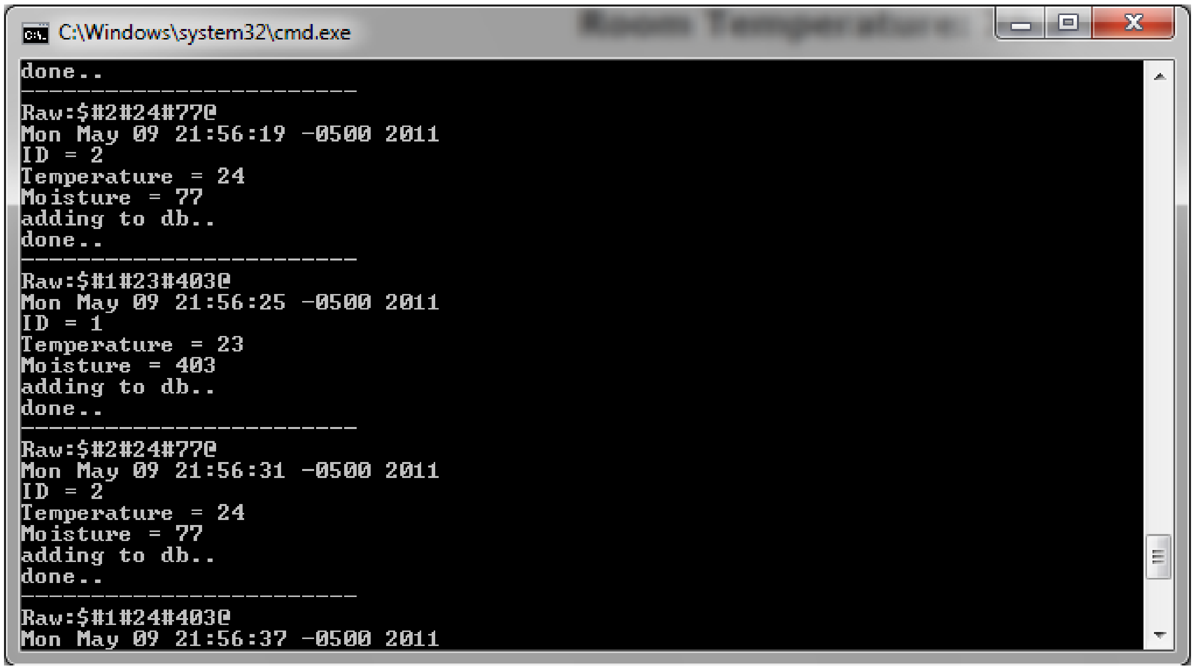
Ruby on Rails App:

The Web-app enables one to add a sensor station (Fio Board). It also provides a dashboard for each sensor station which shows current sensor readings along with the graphs for each sensor.

Program Output Snapshots:

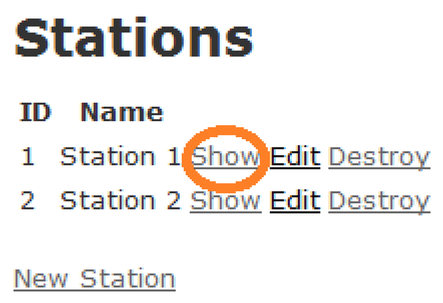
The Arduino program that is uploaded on each Fio board enables one to query individual Fio boards from a controller station using the ruby script. The web-app displays data obtained from the sensors associated with each Arduino board within a browser. The web-app displays historical data which it retrieves from a database and has graphical output too along with sensor data display.

Script output:



```
done..
------------------------
Raw:$#2#24#77@
Mon May 09 21:56:19 -0500 2011
ID = 2
Temperature = 24
Moisture = 77
adding to db..
done..
------------------------
Raw:$#1#23#403@
Mon May 09 21:56:25 -0500 2011
ID = 1
Temperature = 23
Moisture = 403
adding to db..
done..
------------------------
Raw:$#2#24#77@
Mon May 09 21:56:31 -0500 2011
ID = 2
Temperature = 24
Moisture = 77
adding to db..
done..
------------------------
Raw:$#1#24#403@
Mon May 09 21:56:37 -0500 2011
```

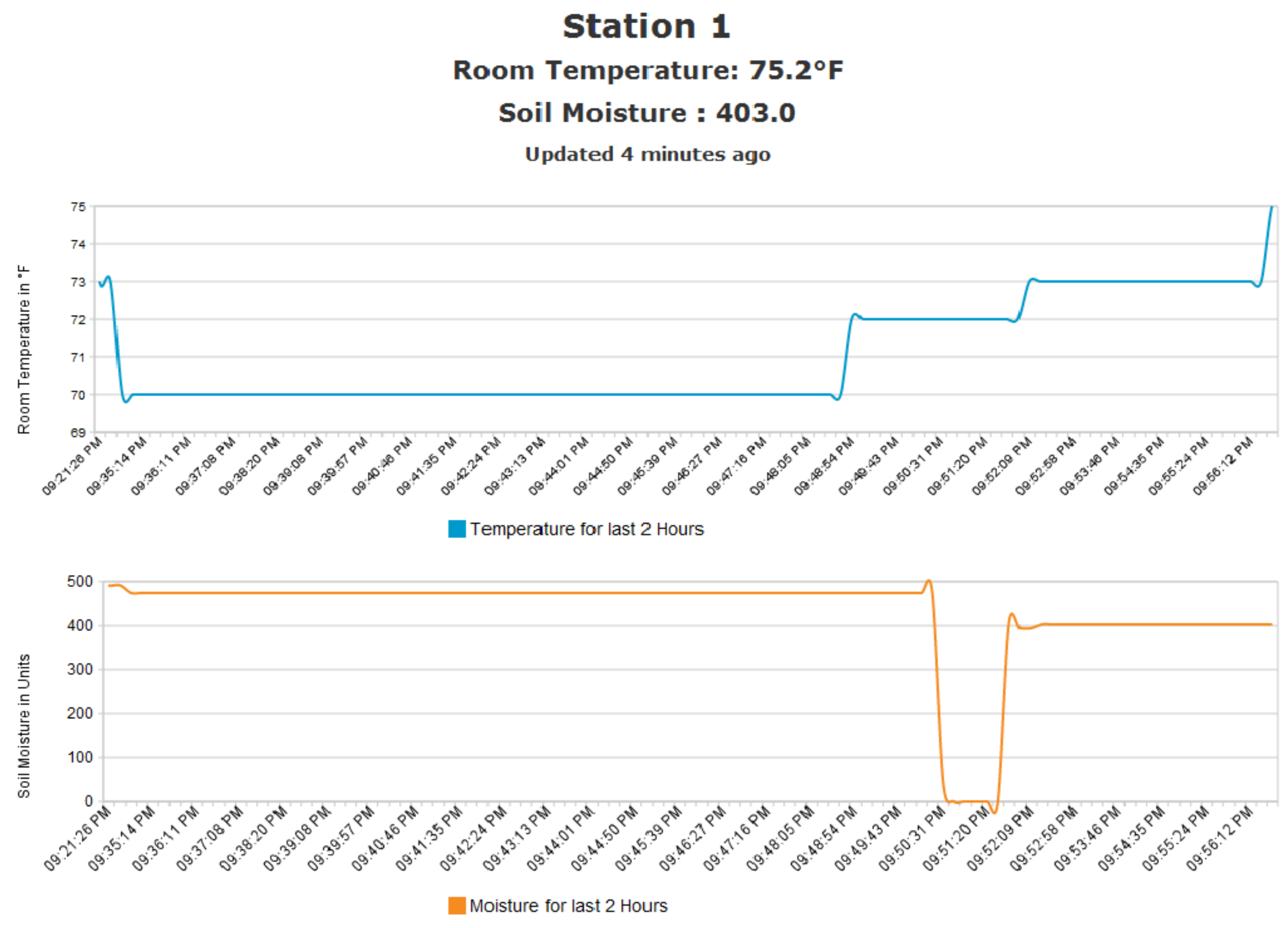
Web-app output:



Stations

ID	Name			
1	Station 1	Show	Edit	Destroy
2	Station 2	Show	Edit	Destroy

New Station



Station 1

Room Temperature: 75.2°F

Soil Moisture : 403.0

Updated 4 minutes ago

Software Installations required on Windows:

1. Download Arduino IDE (http://www.arduino.cc/en/Main/Software)

2. Download and Install Ruby 1.8.7 (http://rubyinstaller.org/downloads/)

3. Install Rails (on command prompt type: 'gem install rails -v 2.3.11')

4. On Command Prompt type : 'gem install seer'

5. On Command Prompt type : 'gem install serialport'

Instructions for the Program:

1. Burn the sketch(in Sketches folder) into Arduino Fio using the Arduino IDE

2. Extract ruby script into 'arduino' folder.

3. To run the data acquisation script : run "arduino/app/jobs> ruby collect_data.rb

4. To run the Web GUI Application : run "arduino> ruby script/server "

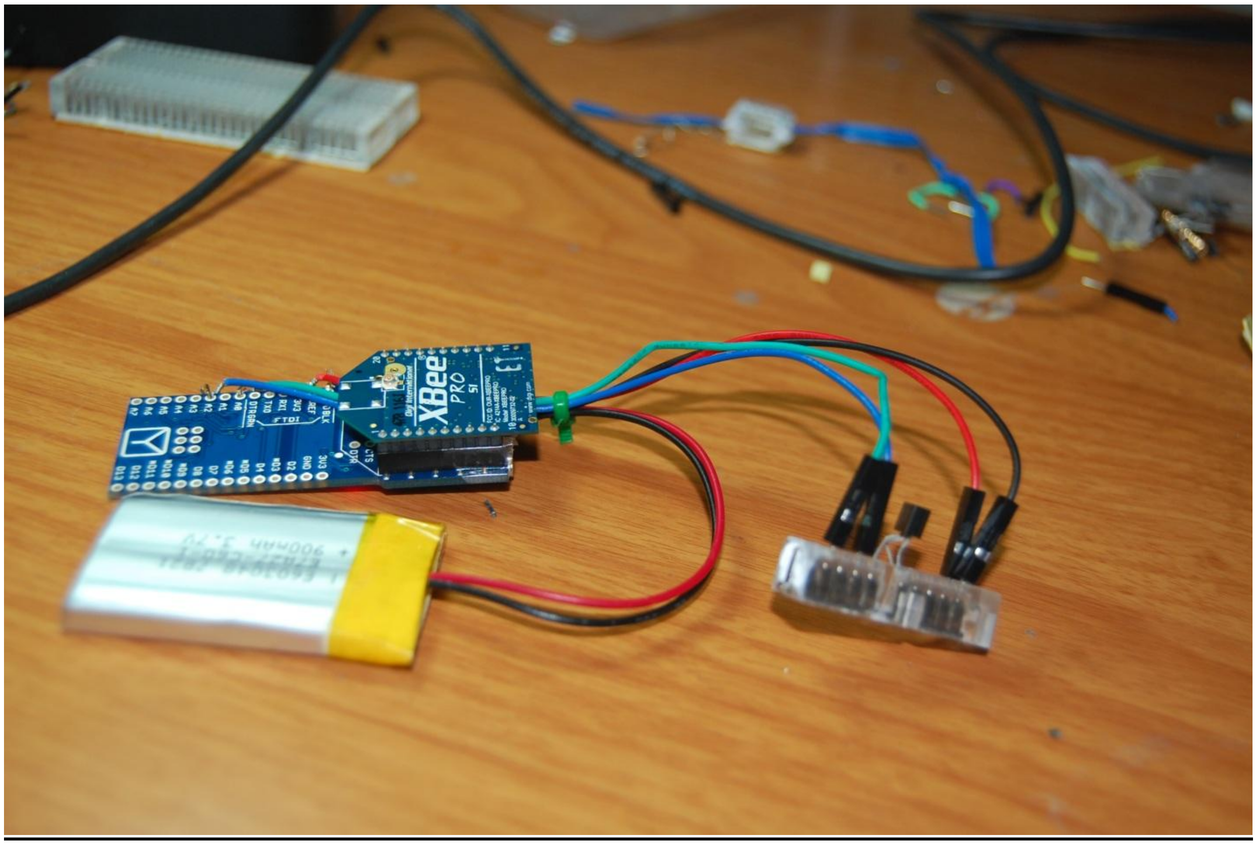
5. goto http://localhost:3000/stations

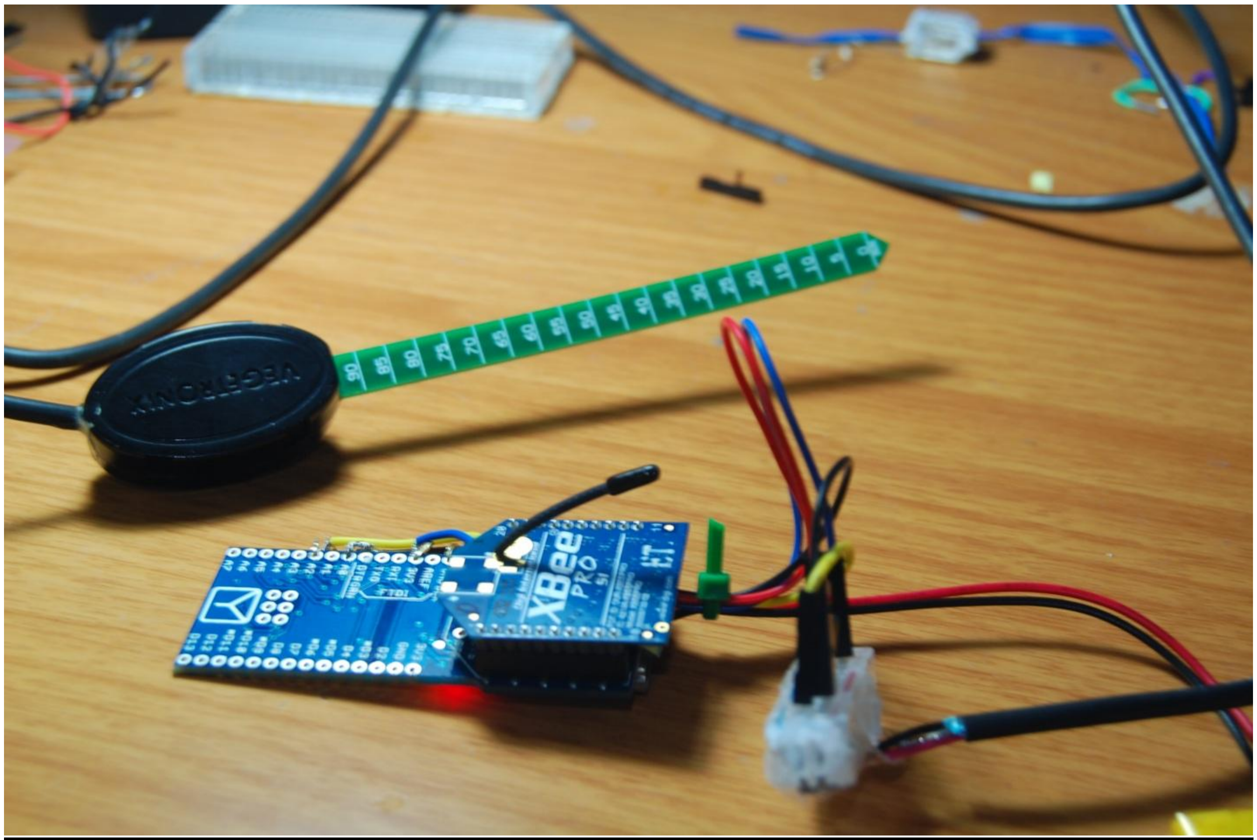
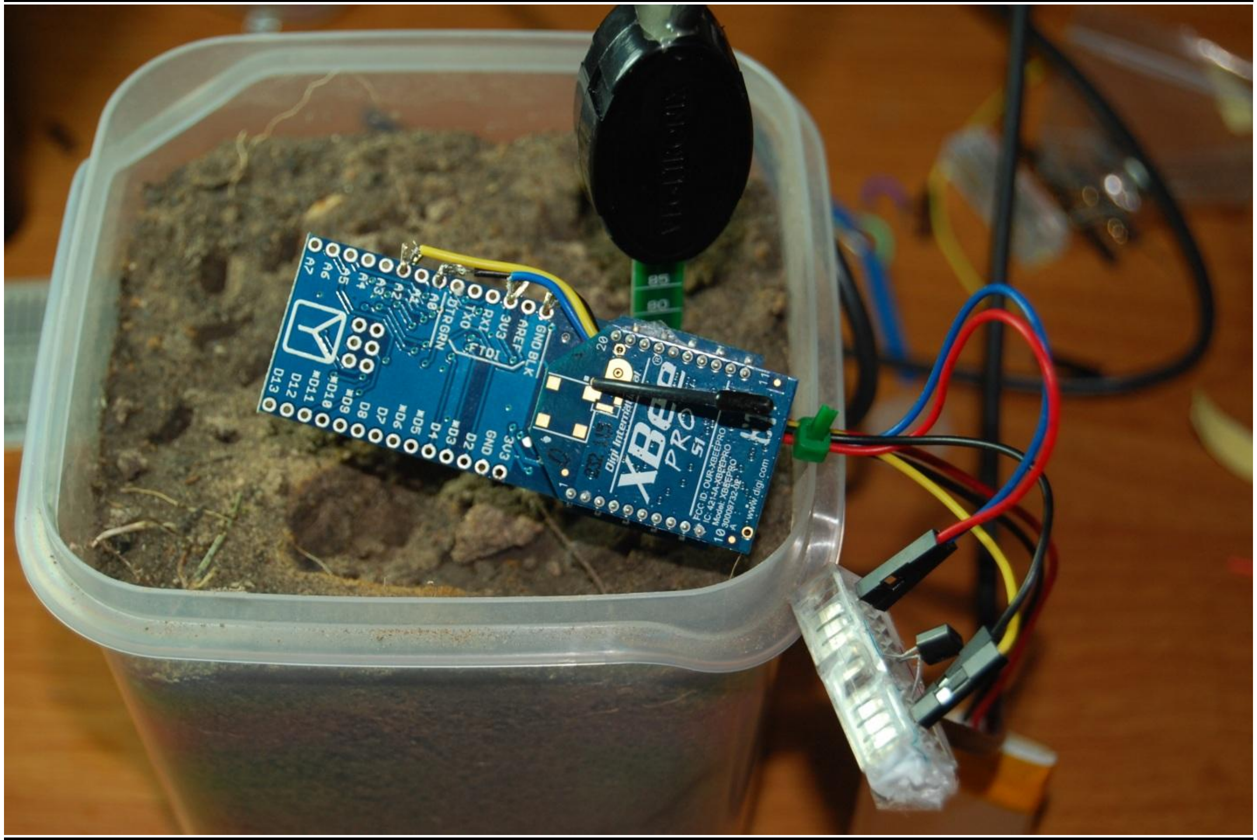
Extensions:

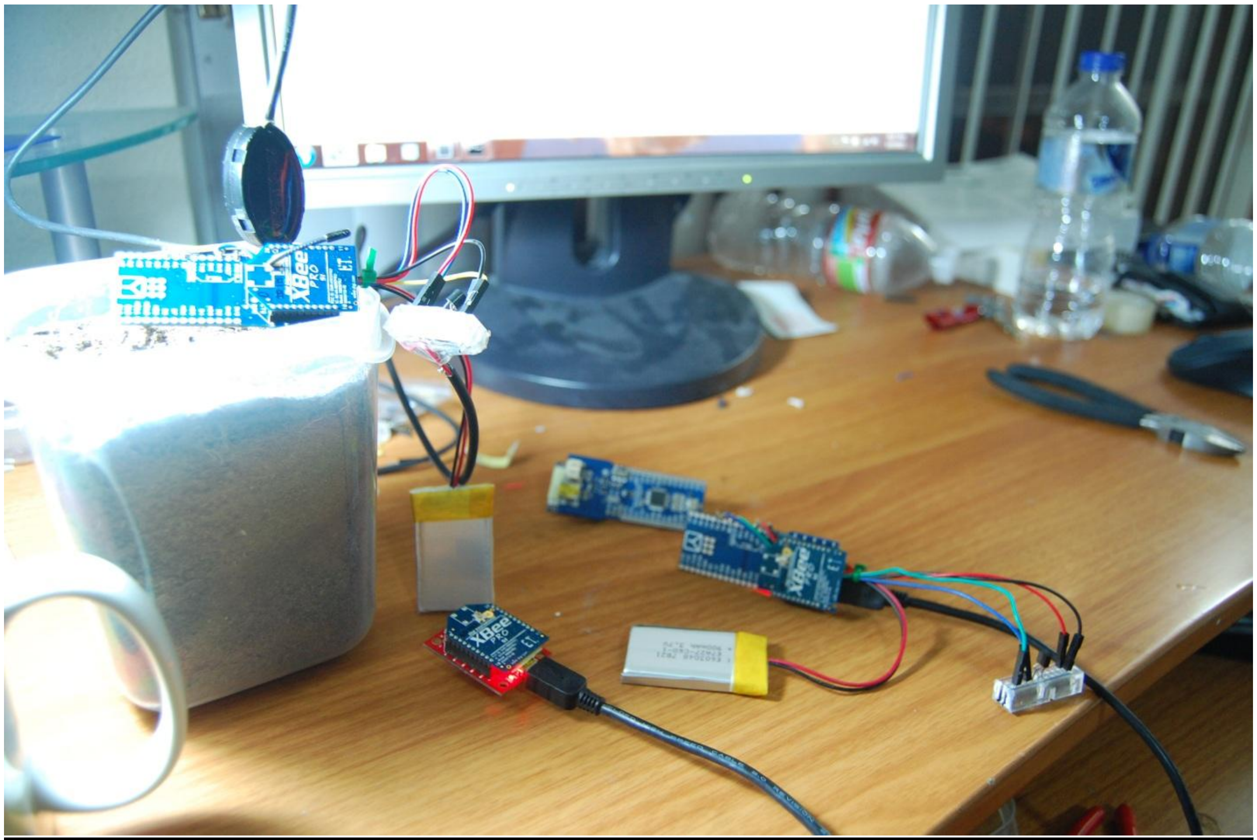
This project uses multiple XBee's communicating in transparent mode. This can be further extended by making use of the API mode for Xbee. This mode enables packet based delivery and acknowledgment protocols ensuring robust data communication. Analog inputs can be directly wired to the Xbee station thus leaving the pins on the Arduino free for more inputs.

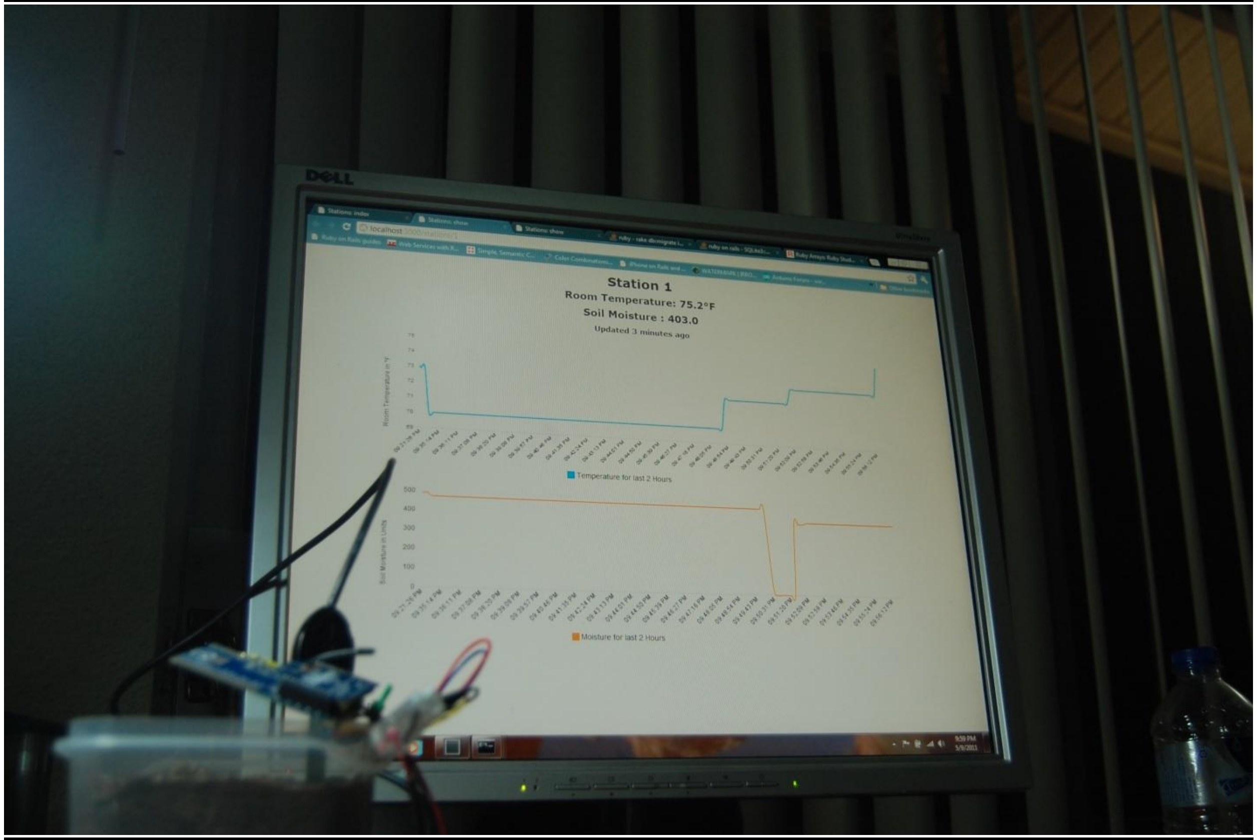
Random Project

Snaps:




XBee PRO
XBee PRO
XBee PRO


DELL
Stations: index
Stations: show
Stations: show
localhost
Station 1
Room Temperature: 75.2°F
Soil Moisture : 403.0
Updated 3 minutes ago
Room Temperature in °F
Temperature for last 2 Hours
Soil Moisture in Units
Moisture for last 2 Hours

Part II

UI and Database Software for the Arduino Project

Arun Chacko
Hsu Wan Kao (Peter)
Yogesh Kalaskar
UL Deepak
Gowthami Muddana
Bekir Sahin
Ali Sura Ozdemir

Code Installation

Serial Client:

1. Select the CPS.jar file.
2. Right click and select "Open with" option and select "Java Platform SE Binary"

Or

1. Go to location of jar file from command prompt.
2. Run the command as "java -jar CPS.jar"

GUI

1. Tomcat web server has to be installed first as it's a web project.
2. Place the Arduino.war file in webapps folder of Tomcat.
3. Start the tomcat server.
4. Access the project in web browser by typing in *http://localhost:80/Arduino*

Serial Client for Xbee:

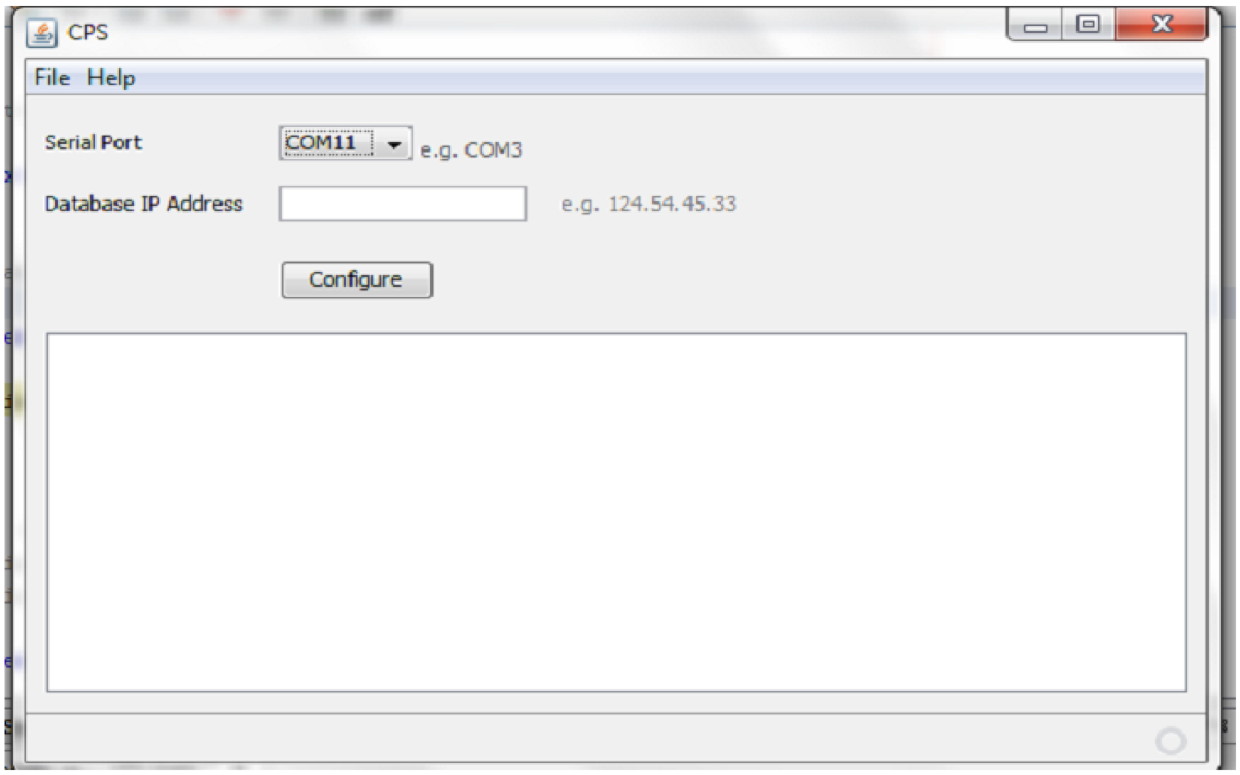
Challenges

In this project Xbee is connected to the arduino which is attached to the temperature sensor and humidity sensor. The Arduino reads the sensor values and transfers it to the client,i.e. in this case a Xbee connected to the PC via serial port. The aim of this project is to store the sensor values and then to draw the linear graph from those values. As the Xbee transmits the data to the serial port, it was necessary create a program to read values directly from Xbee.

Serial Client Development

This part is developed in java language with Swing UI framework. There is not any direct way to access the serial port. The external dll's are called from java program for that purpose. The dll's are platform dependent and they are not backward compatible. So 32 bit dll's would not work on 64 bit machine. Currently all development is done on 64 bit machine.

The data from Xbee is received in byte chunks instead of one line at a time. So there was a need to carefully identify a received message and prepare the meaningful data for the end user.



Serial Port: COM11 e.g. COM3
Database IP Address: e.g. 124.54.45.33

The program identifies the available ports from PC . The user can select the port to which Xbee's are sending the data. The text area in the GUI shows the data read from Xbee radios. The program also gives option to configure the database address. Once the address is provided the values read from sensor can be stored in database. Once the values are available in database, those can be used to render the linear graph.

Graph GUI Part:

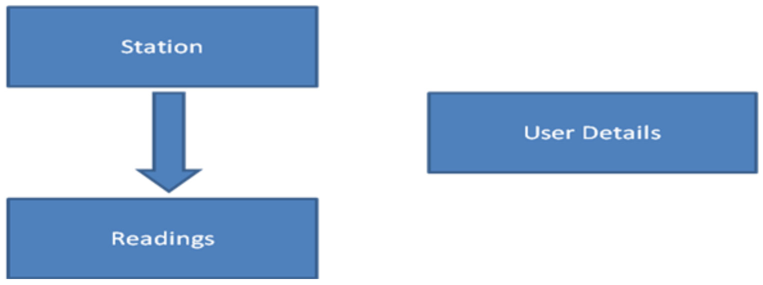
The graph UI is developed in javascript with vector graph graphics. The values are read from database to plot the graph. The vector graphics image is created from the available data. The image is shown in the browser. The data can not be selected from graph shown in the browser because it is rendered as image. Currently the graph can build from integer values only. The sensor values are floats , they are converted to integers to plot the graph.
The temperature data and moisture data can be seen on graph. There are option provided to choose from Complete Data and Latest Data. The complete Data renders the whole data for a particular sensor from database while the Latest data plots the data from last few minutes from current given time.

Database:

The database we used was MySql. In total we used 3 tables:

- Station:
  The Stations table contains specific information regarding the different stations.
- Readings:
- The Readings table contains all of the readings taken directly from the sensors on the different stations.
- UserDetails:
  The UserDetails table contains information for each user on the system.

Database Schema:



Database Interaction Class:

The main purpose of the database interaction class (Database.java) is to provide the GUI a way to interface with the database. The class was written in the JAVA language, and contains ten functions.

Functions:

+DatabaseManager: constructor
-getDateTime: get current date and time
+openDatabase: connects to database
+closeDatabase: disconnects database
+insertData: stores panID, myID, temperature, and moisture into database
+fetchData: fetches a result set of a sensor's periodical values
+fData: returns a string array. Timestamps and values are alternately stored in the array
+addStation: adds a new station into database
+removeStation: removes a station from database
+dummyData: generates dummy data with given sampling period and lasting time

Database Script Code:

This code can just be pasted into the mysql console.

Code:

```
drop database if exists cps;
create database cps;
use cps;

--
-- Table structure for table `station`
--

DROP TABLE IF EXISTS `station`;
CREATE TABLE `station` (
  `panid` varchar(100) NOT NULL,
  `myid` varchar(200) NOT NULL,
  `userid` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`myid`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Table structure for table `readings`
--

DROP TABLE IF EXISTS `readings`;
CREATE TABLE `readings` (
  `auto` int(11) NOT NULL AUTO_INCREMENT,
  `timestamp` datetime NOT NULL,
  `panid` varchar(100) NOT NULL,
  `myid` varchar(200) NOT NULL,
  `temp` double NOT NULL,
  `moist` double NOT NULL,
  PRIMARY KEY (`auto`),
  KEY `myid` (`myid`),
  FOREIGN KEY (myid) REFERENCES station(myid)
) ENGINE=MyISAM AUTO_INCREMENT=10 DEFAULT CHARSET=latin1;

--
-- Table structure for table `userdetails`
--

DROP TABLE IF EXISTS `userdetails`;
CREATE TABLE `userdetails` (
  `userid` varchar(50) NOT NULL,
  `password` varchar(20) DEFAULT NULL,
  `firstname` varchar(20) DEFAULT NULL,
  `middlename` varchar(20) DEFAULT NULL,
  `lastname` varchar(20) DEFAULT NULL,
  `contact` varchar(15) DEFAULT NULL,
  `address` varchar(50) DEFAULT NULL,
  `city` varchar(20) DEFAULT NULL,
  `state` varchar(20) DEFAULT NULL,
  `email` varchar(100) DEFAULT NULL,
  `securityquestion` varchar(100) DEFAULT NULL,
  `securityanswer` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Database Interaction Class:

The DatabaseManager.java file can be imported to the project, and the system can use its functions to interact with the database.

```
package uh.cs.arduino;


import com.sun.xml.internal.ws.message.stream.StreamAttachment;
import java.sql.*;
import java.util.Calendar;
import java.util.Date;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;

/**************************************************************************
*  Summary: This class provide our cps project the
*                              database functionalties.
*  Functions list:
*          -getDateTime: get current date and time
*          parameters:         none
*          variables:          none
*          return string in "yyyy-MM-dd HH:mm:ss" format
*
*          +DatabaseManager: constructor
*          parameters:         none
*          variables:          String url = "jdbc:mysql://localhost/cps"
*                    String username = "cps"
*                              String password = "cps123"
*
*          +openDatabase: connects to database
*          parameters:         none
*          variables:          none
*
*          +closeDatabase: disconnects database
*          parameters:         none
*          variables:          none
*
*          +insertData: stores panID, myID, sensor type, and value into database
*          parameters:         String panid
*                   String myid
*                              String sensortype
*                              Double value
*          variables:          the same to parameters
*
*      +fetchData: fetches a result set of a sensor's periodical values including timestamps and values
*          parameters: String panid
*                   String myid
*                   String sensortype
*                   String starttime
*                       String endtime
*           variables:            the same to parameters
*
*
*       +fData: returns a string array. Timestamps and values are alternately stored in the array
*           parameters: String panid
*                       String myid
*                       String sensortype
*                       String starttime
*                       String endtime
*           variables:            the same to parameters
*
*       +addStation: adds a new station into database
*           parameters:       String panid
*                             String myid
*           variables:        the same to parameters
*
*           +removeStation: removes a station from database
*           parameters:       String panid
*                             String myid
*           variables:        the same to parameters
*/

public class DatabaseManager
{
    String url, username, password;
    Connection connection;

    private String getDateTime() {
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date dateandtime = new Date();
        return dateFormat.format(dateandtime);
    }


    public DatabaseManager()
    {
        url = "jdbc:mysql://localhost/cps";
        username = "cps";
        password = "cps123";
    }

    public void openDatabase() throws ClassNotFoundException, InstantiationException,
            SQLException, IllegalAccessException
    {
        Class.forName ("com.mysql.jdbc.Driver").newInstance ();
        connection = DriverManager.getConnection (url, username, password);
         System.out.println("Connected..");
    }

    public void closeDatabase()
    {
        if (connection != null)
        {
            try
            {
                connection.close ();
                System.out.println("closed...");
            }
            catch (Exception e)
            {
                System.err.println("Error Closing Database!");
            }
        }
    }
```

```
public void insertData(String panid, String myid, String sensortype, Double value)
{
    try
    {
        Statement s = connection.createStatement();
        s.executeUpdate ("INSERT INTO readings (auto,timestamp,panid,myid,sensortype,value)"
                + " VALUES (NULL , '"
                + getDateTime() + "', '"
                + panid + "', '"
                + myid + "' , '"
                + sensortype + "' , '"
                + value + "' )");
        s.close();
    }
    catch (Exception e)
    {
        System.err.println("Error Inserting Data!");
    }
}

public List<Integer> fetchData(boolean isFullData,String columnName, int minutes)
{
        List<Integer> dataList = new ArrayList<Integer>();
    Statement s = null;
    ResultSet rs = null;
    String query ="";
    String endTime="";
    String startTime="";


    if(isFullData)
        query = "SELECT "+columnName+" FROM readings";
    else
    {
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Calendar calendar = Calendar.getInstance();
        endTime = getDateTime();
        calendar.set(Calendar.MINUTE, calendar.get(Calendar.MINUTE)-minutes);
        startTime = dateFormat.format(calendar.getTime());
        query = "SELECT "+columnName+" FROM readings WHERE "
                + "timestamp BETWEEN'" + startTime + "' AND '" + endTime + "'";
    }
    try
    {
        s = connection.createStatement();
        rs = s.executeQuery(query);
        while (rs.next())
        {
            dataList.add((int)(double)rs.getDouble(columnName));
        }
        s.close();
        System.out.println("fetchData success between "+startTime+" "+endTime +" "+dataList.size());
    }
    catch (Exception e)
    {
        System.err.println("Error fetching Data!"+e);
    }
    return dataList;
}
public String[] fData(String panid, String myid, String sensortype, String starttime, String endtime)
{
    Statement s = null;
    ResultSet rs = null;
    String[] result = new String[0];
    String query = "SELECT timestamp, value FROM readings "
                + "WHERE panid = '" + panid
                + "' AND myid = '" + myid
                + "' AND sensortype = '" + sensortype
                + "' AND timestamp BETWEEN'" + starttime + "' AND '" + endtime + "'";
    try
    {
        s = connection.createStatement();
        rs = s.executeQuery(query);
        rs.last();
        result = new String[rs.getRow() * 2];
        rs.beforeFirst();
        int i = 0;
        while (rs.next()) {
            result[i] = rs.getString("timestamp");
            Double tmp = rs.getDouble("value");
            result[i+1] = tmp.toString();
            i += 2;
        }
        s.close();
        System.out.println("fData success.");
    }
    catch (Exception e)
    {
        System.err.println("Error fetching Data!");
    }
    return result;
}

public void addStation(String panid, String myid)
{
    try
    {
        Statement s = connection.createStatement();
        s.executeUpdate ("INSERT INTO station (panid,myid) VALUES ('"
                + panid + "','"
                + myid +"')");
        s.close();
        System.out.println("Adding station success!");
    }
    catch (Exception e)
    {
        System.err.println("Error Adding Station!");
    }
}

public void removeStation(String panid, String myid)
{
    try
    {
        Statement s = connection.createStatement();
        s.executeUpdate ("DELETE FROM station where panid='" + panid + "' AND myid='" + myid + "'");
        s.close();
        System.out.println("Removing station success!");
    }
    catch (Exception e)
    {
        System.err.println("Error Removing Station!");
    }
}

public static void main(String [] args) throws Exception
{
         DatabaseManager base = new DatabaseManager();
         base.openDatabase();
         //base.fetchData(false,"moist",45);
         base.bulkInsert();
         base.closeDatabase();
    }


            private void bulkInsert() throws SQLException, InterruptedException, ParseException
            {
                        // TODO Auto-generated method stub
                        Statement s = null;
        ResultSet rs = null;
        //SimpleDateFormat formatter = new SimpleDateFormat("yyyy-dd-mm hh:mm:ss");
        //java.sql.Timestamp  sqlDate = new java.sql.Timestamp(new java.util.Date().getTime());

        PreparedStatement stmt                    = connection.prepareStatement("INSERT INTO
readings(timestamp,panid,myid,temp,moist) VALUES (?,?,?,?,?)");
        for(int i=0;i<300;i++)
        {
            //stmt.setInt(1, i);
            //Date date = formatter.parse(""+Calendar.getInstance().getTime());
            stmt.setTimestamp(1, new java.sql.Timestamp(Calendar.getInstance().getTimeInMillis()));
//date.getTime()
            stmt.setString(2, "1234");
            stmt.setString(3, "0001");
            stmt.setDouble(4, new Double(Math.random()*100));
            stmt.setDouble(5, new Double(Math.random()*500));
            stmt.execute();
            Thread.sleep(1000);
        }
        stmt.close();
            }

    public List<String> getData() throws SQLException
    {
        Statement s = null;
        ResultSet rs = null;
        String query = "SELECT value FROM readings";
        List<String> valueList = new ArrayList<String>();
        try
        {
            s = connection.createStatement();
            rs = s.executeQuery(query);
        }
        catch (Exception e)
        {
            System.err.println("Error fetching Data!");
        }
        int i=0;
        while (rs.next())
        {
            valueList.add(""+rs.getDouble(1));
           System.out.println(++i+" values..."+ rs.getDouble(1));
         }
        s.close();
        return valueList;
    }
}
```

GUI (Graphical User Interface):

The technologies we have used for building the GUI are

Front End: JSP, HTML/CSS

Back End: MYSQL 5.1

The functionalities provided by the graphical user interface are:

Register: This page is designed to register the user with the website so as to let him use the functionalities provided.

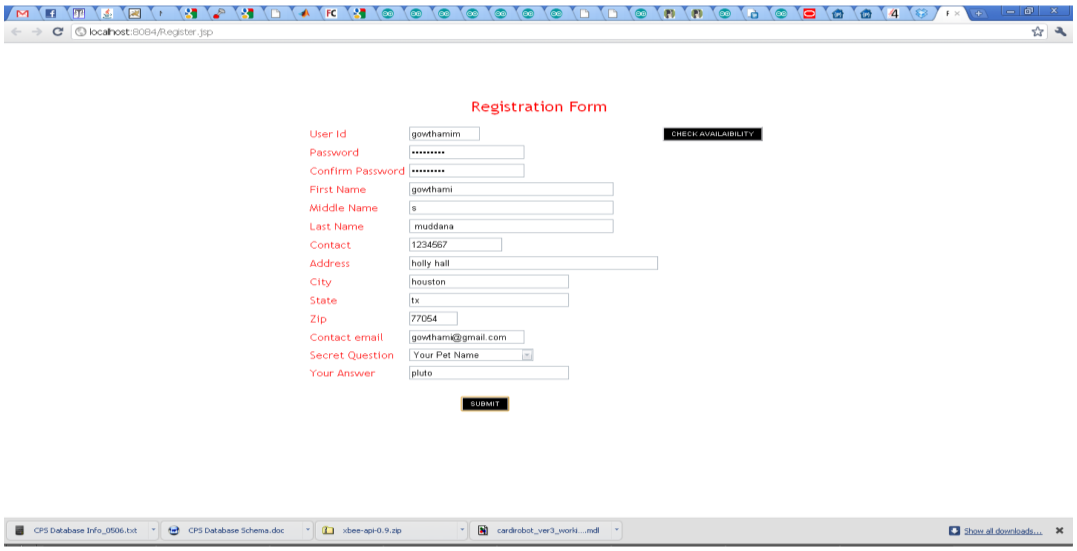
Login: Once the registration is done, the user can login with his userid and password to access the functionalities.

Add Station: This module allows the user to add stations (Xbee) into the database and can access the data coming from the aurdino through the serial port using monitor graph functionality. The user needs to enter the Panid and Myid of the Xbee for the station to be added successfully.

Remove Station: This module allows the user to delete a station from the database if he no longer wants to use that aurdino. The user gets a list of all the stations that the user has added and he can choose from them which station is to be removed or deleted.

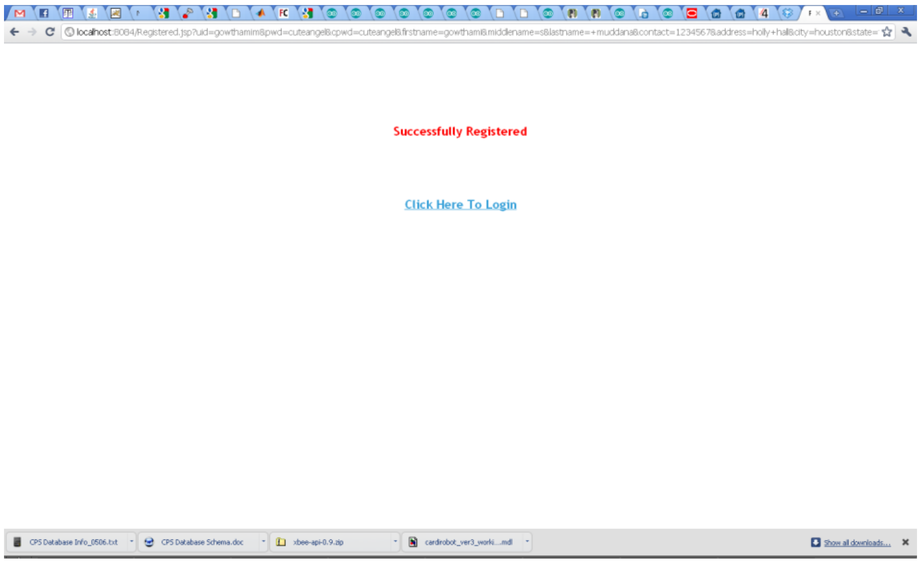
ScreenShots :

Register:



Registration Form

User Id	gowthamim
Password	••••••••••
Confirm Password	••••••••••
First Name	gowthami
Middle Name	s
Last Name	muddana
Contact	1234567
Address	holly hall
City	houston
State	tx
Zip	77054
Contact email	gowthami@gmail.com
Secret Question	Your Pet Name
Your Answer	pluto

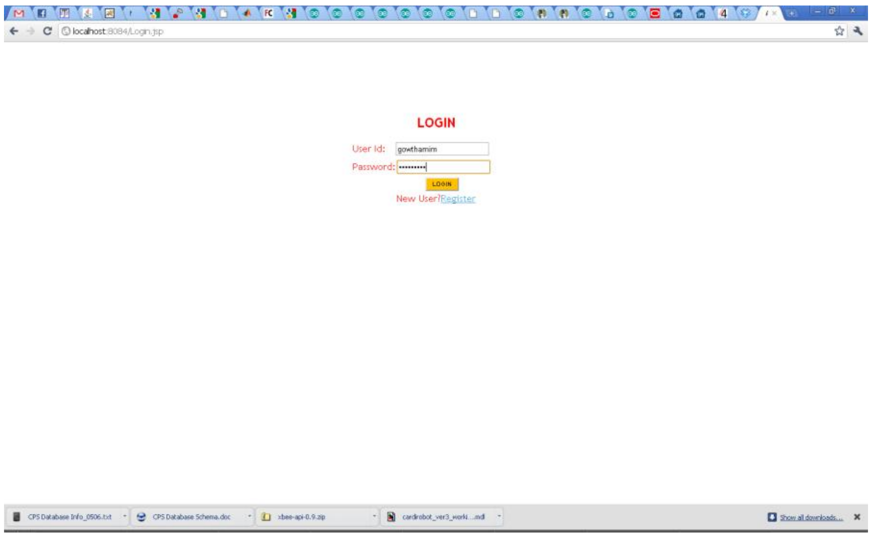


localhost:8084/Registered.jsp?uid=gowthamim&pwd=cuteangel&cpwd=cuteangel&firstname=gowthami&middlename=s&lastname=+muddana&contact=12345678&address=holly+hall&city=houston&state=

Successfully Registered

Click Here To Login

Login:



localhost:8084/Login.jsp

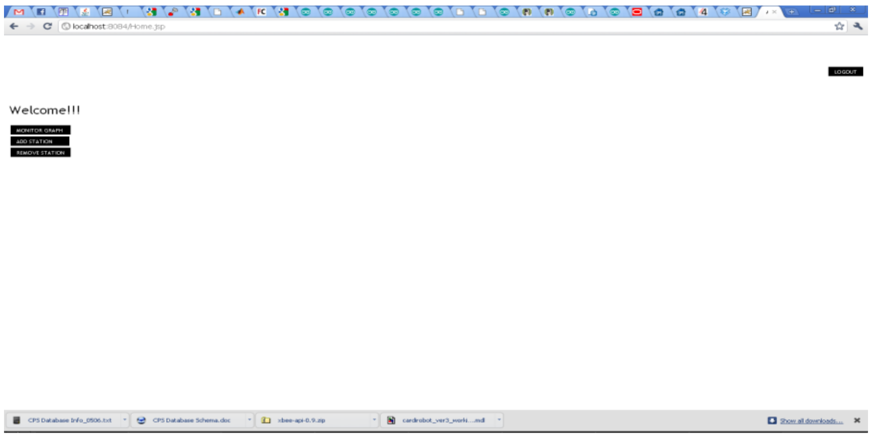
LOGIN

User Id: gowthamim

Password: ••••••••••

LOGIN

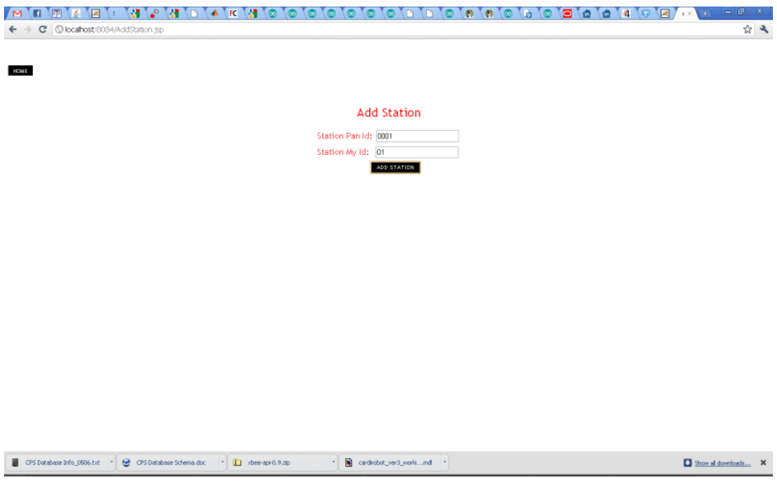
New User?Register



localhost:8084/Home.jsp

Welcome!!!

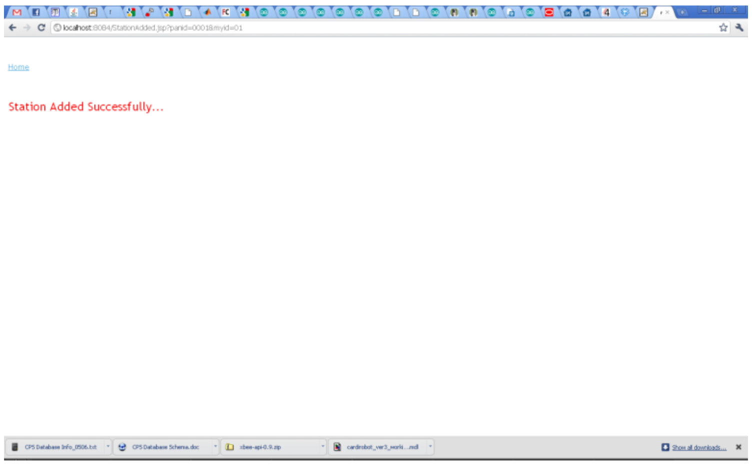
Add Station:


localhost:8084/AddStation.jsp
HOME
ADD STATION
CPS Database Info_0506.txt
CPS Database Schema.doc
xbee-api-0.9.zip
cardrobot_ver3_worki....mdl
Show all downloads...

Add Station

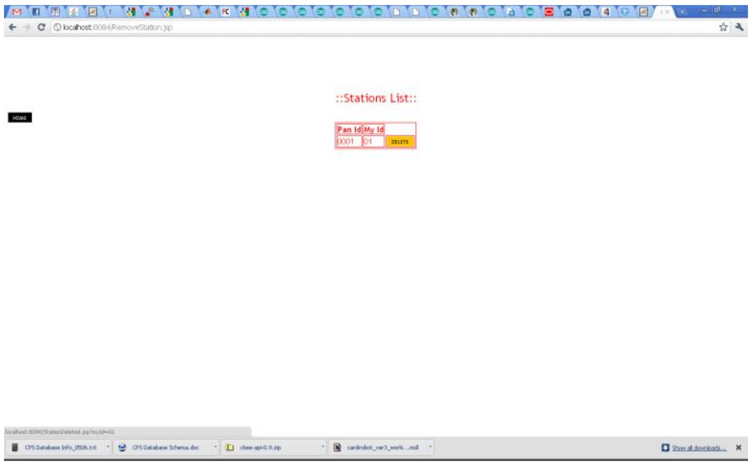
Station Pan Id: 0001

Station My Id: 01


localhost:8084/StationAdded.jsp?panid=0001&myid=01
Home
CPS Database Info_0506.txt
CPS Database Schema.doc
xbee-api-0.9.zip
cardrobot_ver3_worki....mdl
Show all downloads...

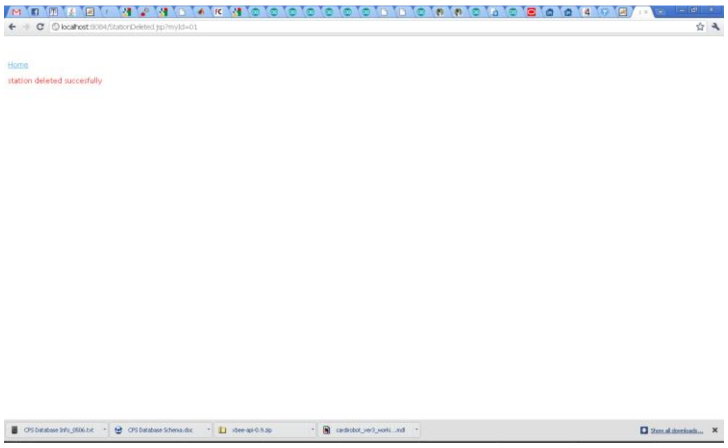
Station Added Successfully...

Remove Station:


localhost:8084/RemoveStation.jsp
HOME
localhost:8084/StationDeleted.jsp?myId=01
CPS Database Info_0506.txt
CPS Database Schema.doc
xbee-api-0.9.zip
cardrobot_ver3_worki....mdl
Show all downloads...

::Stations List::

Pan Id	My Id	
0001	01	DELETE


localhost:8084/StationDeleted.jsp?myId=01
Home
CPS Database Info_0506.txt
CPS Database Schema.doc
xbee-api-0.9.zip
cardirobot_ver3_worki....mdl
Show all downloads...

station deleted succesfully