Communicating with Arduino

Antonio Natali

Alma Mater Studiorum – University of Bologna
viale Risorgimento 2, 40136 Bologna, Italy
antonio.natali@unibo.it

Abstract. *Overview of the content:*

1. Supports for the interaction with Arduino via serial port (Section 2)
2. An Arduino sketch that uses basic communication patterns (Subsection 2.2)
3. Exchanging messages between a pc and Arduino (Section 3)
4. An application observer on the serial port (Subsection 3.2)
5. From messages to events (Section 4)
6. An application with messages and interrupts (Section 5)
7. An application with messages and events (Section 6)
8. DevButtonArduino (Subsection 6.1)
9. A remote button-event receiver (Subsection 6.2)

Pre-reading: [1].

1 Introduction

Serial communications provide the way for an Arduino board to interact with a computer and other devices. In order to lead again arduino communication to our general communication framework, we introduce first of all the interface *ISerialPortInteraction* as a specialization of our generic communication interface *IConnInteraction*:

```
1 package it.unibo.arduino.serial;
2 import gnu.io.SerialPort;
3 import it.unibo.is.interfaces.protocols.IConnInteraction;
4 public interface ISerialPortInteraction extends IConnInteraction{
5     public SerialPort getPort();
6 }
```

Listing 1.1. `ISerialPortInteraction.java`

The required `gnu.io.SerialPort` communication device will be provided by objects that implement the *ISerialPortConnection* interface:

```
1 package it.unibo.arduino.serial;
2 import it.unibo.is.interfaces.IObservable;
3 import it.unibo.is.interfaces.IObserver;
4 public interface ISerialPortConnection extends IObserver, IObservable{
5     public gnu.io.SerialPort connect( String portName, Class userClass) throws Exception;
6     public void closeConnection( String portName ) throws Exception;
7 }
```

Listing 1.2. `ISerialPortConnection.java`

2 Arduino serial interaction supports

An object that implements the interface *ISerialPortConnection* can be defined as follows:

```java
package it.unibo.arduino.serial;
import java.util.Enumeration;
import java.util.Observable;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;
import it.unibo.is.interfaces.IOutputView;
import it.unibo.system.SituatedPlainObject;

public class SerialPortSupport extends SituatedPlainObject implements ISerialPortConnection {
    private static final int TIME_OUT = 2000;
    private static final int DATA_RATE = 9600;
    protected SerialPort serialPort;
    /*
     *  "/dev/tty.usbserial-A9007UX1", // Mac OS X
     *  "/dev/ttyUSB0", // Linux
     *  "COM31", // Windows
     */
    public SerialPortSupport( IOutputView outView) {
        super(outView);
        this.name = "portPojo";
    }
    @Override
    public SerialPort connect(String portName, Class userClass ) throws Exception {
        CommPortIdentifier portId = null;
        Enumeration<?> portEnum = CommPortIdentifier.getPortIdentifiers();
        // First, Find an instance of serial port as set in PORT_NAMES.
        while (portEnum.hasMoreElements()) {
            CommPortIdentifier currPortId = (CommPortIdentifier) portEnum
                    .nextElement();
            if (currPortId.getName().equals(portName)) {
                portId = currPortId;
                break;
            }
        }
        if (portId == null) {
            throw new Exception("Could not find COM port");
        }
        // System.out.println("*** SerialPort connect "+userClass.getName());
        // open serial port, and use class name for the appName.
        serialPort = (SerialPort) portId.open( userClass.getName(), TIME_OUT);
        // set port parameters
        serialPort.setSerialPortParams(DATA_RATE, SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
        return serialPort;
    }
    @Override
    public void closeConnection(String portName) throws Exception {
    }
    @Override
    public void update(Observable arg0, Object arg1) {
    }
}
```

Listing 1.3. `SerialPortSupport.java`

Note that this object can work on different platforms but a proper, platform-specific port name must be given; for example:

Port names

```
Mac OS X  :  "/dev/tty.usbserial-A9007UX1"
Linux     :  "/dev/ttyUSB0"
Windows   :  "COM31"
```

2.1 An Arduino support for basic communication

An object that implements the basic operations `sendALine` and `receiveALine` of the interface *ISerialPortConnection* can be defined as follows:

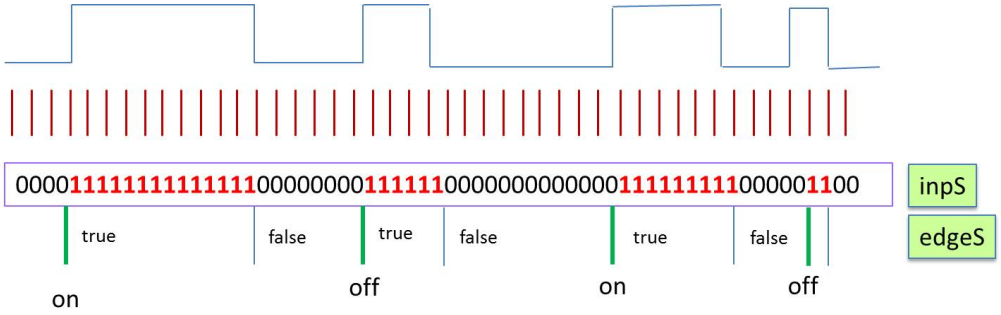
```java
package it.unibo.arduino.serial;
```

```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import gnu.io.SerialPort;
import it.unibo.is.interfaces.IOutputView;
import it.unibo.system.SituatedPlainObject;

public class SerialPortConnSupport extends SituatedPlainObject implements ISerialPortInteraction{
final static int SPACE_ASCII = 32;
final static int DASH_ASCII = 45;
final static int NEW_LINE_ASCII = 10;
final static int CR_ASCII = 13;

private InputStream inpS;
private OutputStream output;
private SerialPort serialPort;

    public SerialPortConnSupport( SerialPort serialPort, IOutputView outView ) {
        super(outView);
        this.serialPort = serialPort;
        init();
    }
    protected void init( ){
        try {
            inpS = serialPort.getInputStream();
            output = serialPort.getOutputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void sendALine(String msg) throws Exception {
        output.write(msg.getBytes());
        output.write('\n');
        //println(" SerialPortConnSupport has sent " + msg);
    }
    @Override
    public void sendALine(String msg, boolean isAnswer) throws Exception {
        sendALine( msg );
    }
    @Override
    public String receiveALine() throws Exception {
        boolean readSerialLineDone = false;
        String s = "";
        do{
            int availableBytes = inpS.available();
            for( int i=1; i<=availableBytes; i++ ){
                int singleData = inpS.read();
                if(singleData == CR_ASCII || singleData == NEW_LINE_ASCII ){
                    if( singleData == NEW_LINE_ASCII ){
                        readSerialLineDone = true;
                    }
                }else s = s + (char)singleData;
            }
        }while( !readSerialLineDone );
        //println("          receiveALine " +  s  + " " + inpS.available());
        return s;
    }
    @Override
    public void closeConnection() throws Exception {
        if (serialPort != null) {
            serialPort.removeEventListener();
            serialPort.close();
        }
    }
    @Override
    public SerialPort getPort() {
        return serialPort;
    }
}
```

Listing 1.4. SerialPortConnSupport.java

2.2 An Arduino sketch that uses basic communication patterns

A physical button can be viewed as a source of information that emits a *wave* like that shown in the following picture:



The samples form a sequence of values in which each value can be modelled as a *boolean*, where `true` means "high" and `false` means "low". From this sequence of values ('*input sequence*' or `inpS`) we must find the edges that in their turn form a sequence of values called here *edge sequence* or `edgeS`. Each value of the `edgeS` sequence can be also modelled as a *boolean*, where `true` means "low to high" and `false` means "high to low". Since the button is initially supposed to be *unpressed* (the voltage level is low), the sequence `edgeS` is either empty or takes always the following form:

A typical (non empty) edgeS sequence

```
true false true false ...
```

The functional requirements of the system can now be expressed as follows: the `Led` is turned on `N` times, where `N` is the number of `true` in odd position in the `edgeS` sequence.

Since there is an abstraction gap between the physical and the logical model of a button, a proper software layer is needed in order to overcome that gap.

2.3 Arduino button (polling)

2.4 Arduino button (interrupt)

The sketch `SerialComm.ino` implements the following features:

- waits for a message (***request***) from the pc and then sends the same message (echo) as ***response***;
- handles interrupts associated with a pin connected to a button interrupt by sending to the pc the message (***event***) `arduinoInterrupt(button)`.

```
1  /*
2   SerialComm.ino
3  */
4  int NEW_LINE_ASCII = 10;
5  int CR_ASCII = 13;
6  int led = 13;
7  int buttonSw = 3; //7          //ARDUINO UNO: pin 3 -> MAPS TO interrupt 1
8  String inputString = "";       // a string to hold incoming data
9  boolean stringComplete = false; // whether the string is complete
10 int n = 1;
11
12
13 /*
14 * Interrupt handling with debouncing
15 */
16 boolean debouncing(){
```

```
static unsigned long lastInterruptTime = 0;
static int lastSw = 0;
unsigned long interruptTime = millis();
  if( (interruptTime - lastInterruptTime) > 100 ){
    lastInterruptTime = interruptTime;
    return true;
  }
  return false;
}

void buttonInterruptHandler(){
  boolean ok = debouncing();
  if( ok ){
     int sw = digitalRead(buttonSw);
     if( sw == 1 )
     //Serial.println("arduinoInterrupt(button,value("+String(sw)+"))");
     Serial.println("arduinoInterrupt(button)");
   }
}
/*
*  Read a line from the serial port
*/
void serialInput() { //ex serialEvent ...
   while (Serial.available()) {
     int inChar = Serial.read();
     if ( inChar == NEW_LINE_ASCII ) {
       stringComplete = true;
       //digitalWrite(led, HIGH);
    } else inputString = inputString + (char)inChar;
  }//while
}

/*
* setup
*/
void setup() {
  pinMode(led, OUTPUT);
  pinMode(buttonSw, INPUT);
  n = 1;
  Serial.begin(9600);
  inputString.reserve(256);
  digitalWrite(buttonSw, HIGH); //attach pull up => unpressed = 1
  attachInterrupt(1, buttonInterruptHandler, CHANGE); // RISING CHANGE FOLLING LOW
}

/*
* LOOP
*/
void loop() {
     serialInput() ;
     if (stringComplete) {
        Serial.println( inputString + " n=" + String(n) );
        delay(10);
        n = n + 1;
        //digitalWrite(led, LOW);
        inputString = "";
        stringComplete = false;
     }
}
```

Listing 1.5. `SerialComm.ino`

3 Exchanging messages with Arduino

The following `Java` applications sends a message to Arduino by waiting for the response before sending another message:

```
package it.unibo.noawtsupports.arduino.intro;
import gnu.io.SerialPort;
import it.unibo.arduino.serial.ISerialPortInteraction;
```

```java
import it.unibo.arduino.serial.SerialPortConnSupport;
import it.unibo.arduino.serial.SerialPortSupport;
import it.unibo.is.interfaces.IOutputView;
import it.unibo.system.SituatedPlainObject;

public class SerialPortRequestResponseMain extends SituatedPlainObject{
    protected String PORT_NAME = "COM23"; // Windows
    protected ISerialPortInteraction portConn;
    protected SerialPort serialPort;

    public SerialPortRequestResponseMain(IOutputView outView){
        super(outView);
    }
    public void doJob(){
        try {
            init();

            requestResponse("First msg");
            requestResponse("Hello from pc");
            requestResponse("Final msg from pc");

            portConn.closeConnection();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    protected void init() throws Exception{
        println(" ==== STARTS. WARNING: rxtxSerial.dll must be in the classpath " + getName() );
        setConnection();
        waitForReady();
    }
    protected void setConnection() throws Exception{
        SerialPortSupport serialPortSupport = new SerialPortSupport( outView );
        serialPort = serialPortSupport.connect(PORT_NAME, this.getClass() );
        println("serialPort="+serialPort);
        portConn = new SerialPortConnSupport(serialPort, outView);
    }
    protected void waitForReady() throws Exception{
        System.out.println("Waiting for arduino STARTUP... ");
        Thread.sleep(2000);
    }
    protected void requestResponse(String msg) throws Exception{
        System.out.println("Sending to arduino ... " + msg);
        portConn.sendALine(msg);
        getTheAnswer();
    }
    protected void getTheAnswer() throws Exception{
        String answer = portConn.receiveALine();
        println("Answer="+ answer);
    }
/*
 * MAIN
 */
    public static void main(String[] args) throws Exception {
        new SerialPortRequestResponseMain(null).doJob();
    }
}
```

Listing 1.6. `SerialPortRequestResponseMain.java`

.

Requirements. To run the application, the `RXTXcomm.jar` library is required. Moreover, the `rxtxSerial.dll` component must be in the classpath.

3.1 An application that handles button-related messages from Arduino

The following `Java` applications is a specialized version of *SerialPortRequestResponseMain* that adds an *observer* to the serial port, so to manage the messages (***events***) sent by Arduino when a button is pressed:

```java
package it.unibo.noawtsupports.arduino.intro;
import it.unibo.is.interfaces.IOutputView;

public class SerialPortButtonMain extends SerialPortRequestResponseMain{

    public SerialPortButtonMain(IOutputView outView){
        super(outView);
    }
    public void doJob(){
        try {
            init();
            System.out.println("APPL END. Now I'll show the answer to interrupts");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    protected void init() throws Exception{
        println(" ==== STARTS. WARNING: rxtxSerial.dll must be in the classpath " + getName() );
        setConnection();
        addSerialPortListener();
    }
    protected void addSerialPortListener() throws Exception{
        serialPort.addEventListener( new ButtonObserver( portConn,outView ) );
        serialPort.notifyOnDataAvailable(true);
    }
/*
 * MAIN
 */
    public static void main(String[] args) throws Exception {
        new SerialPortButtonMain(null).doJob();
    }

}
```

Listing 1.7. `SerialPortButtonMain.java`

3.2 An application observer on the serial port

The *ButtonObserver* class implements an observer of messages sent from Arduino that simply shows the message:

```java
package it.unibo.noawtsupports.arduino.intro;
import java.util.Observable;
import it.unibo.arduino.serial.ISerialPortInteraction;
import it.unibo.is.interfaces.IObserver;
import it.unibo.is.interfaces.IOutputView;
import it.unibo.system.SituatedPlainObject;
import gnu.io.SerialPortEvent;
import gnu.io.SerialPortEventListener;

public class ButtonObserver extends SituatedPlainObject implements IObserver, SerialPortEventListener{
protected ISerialPortInteraction portConn;
protected String curInput = "";
protected int n = 0;

    public ButtonObserver(ISerialPortInteraction portConn,IOutputView outView) {
        super(outView);
        this.portConn = portConn;
    }
    @Override
    public void update(Observable arg0, Object inputLine) {
        try {
            n++;
            String input = (String)inputLine;
            String content = "value(" + n + "," + input + ")";
            System.out.println( "ButtonObserver event= " + content);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
30      @Override
31      public synchronized void serialEvent(SerialPortEvent oEvent) {
32          if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
33            //println("ApplObserver oEvent type=" + oEvent.getEventType() );
34            try {
35               //no read => event not consumed
36               String input = portConn.receiveALine();
37               if( input.length() > 0 ){
38                   update( this, input);
39               }
40            } catch (Exception e) {
41               System.out.println("ArduinoObserver ERROR:"+e.getMessage());
42            }
43          }
44      }
45  }
```

Listing 1.8. `ButtonObserver.java`

4 From messages to events

The *ButtonObserverToEvent* class implements an observer of messages sent from Arduino that trasnforms each message into a local event:

```
1   package it.unibo.noawtsupports.arduino.intro.events;
2   import java.util.Observable;
3   import it.unibo.arduino.serial.ISerialPortInteraction;
4   import it.unibo.event.interfaces.INodejsLike;
5   import it.unibo.inforeply.infrastructure.SysKbInfoReplInfr;
6   import it.unibo.is.interfaces.IOutputView;
7   import it.unibo.noawtsupports.arduino.intro.appl.ApplObserver;
8
9
10  public class ButtonObserverToEvent extends ApplObserver {
11  protected INodejsLike njs;
12
13      public ButtonObserverToEvent(ISerialPortInteraction portConn,IOutputView outView ) throws Exception {
14          super(portConn,outView,null);
15          this.portConn = portConn;
16          njs = SysKbInfoReplInfr.getNodejsLike();
17      }
18      @Override
19      public void update(Observable arg0, Object inputLine) {
20          try {
21              n++;
22              String content = (String)inputLine;
23              //System.out.println( "             update " + content );
24              if( njs != null ) njs.raiseEvent(null, SysKb.eventName, content);
25          } catch (Exception e) {
26              e.printStackTrace();
27          }
28      }
29  }
```

Listing 1.9. `ButtonObserverToEvent.java`

4.1 An event-based application

The *ArduinoEventsHandler* class defines an application event handler for the Arduino events:

```
package it.unibo.noawtsupports.arduino.intro.events;
import it.unibo.event.interfaces.IEventItem;
import it.unibo.is.interfaces.IOutputView;
import it.unibo.nodelike.platform.EventHandler;

public class ArduinoEventsHandler extends EventHandler {
    public ArduinoEventsHandler( IOutputView outView ) throws Exception {
        super( "ardEvH", outView );
    }
    @Override
    public void doJob() throws Exception {
        IEventItem ev = this.getEventItem();
        this.showMsg("ardEvH -> "+ev.getEventId()+ " " + ev.getMsg() );
    }
 }
```

Listing 1.10. `ArduinoEventsHandler.java`

The class *SerialPortButtonEventMain* defines an event-based application:

```
package it.unibo.noawtsupports.arduino.intro.events;
import it.unibo.event.interfaces.INodejsLike;
import it.unibo.is.interfaces.IOutputView;
import it.unibo.noawtsupports.arduino.intro.SerialPortButtonMain;
import it.unibo.nodelike.platform.NodejsLike;

public class SerialPortButtonEventMain extends SerialPortButtonMain{
protected INodejsLike njs;

    public SerialPortButtonEventMain(IOutputView outView){
        super(outView);
    }
    public void doJob(){
        try {
            println(" ==== STARTS. WARNING: rxtxSerial.dll must be in the classpath " + getName() );
            setConnection();
            addSerialPortListener();
            createEventPlatform();
            addApplicationEventHandler();
            println("*** APPL END. Now I'll handle events");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    protected void addSerialPortListener() throws Exception{
        serialPort.addEventListener( new ButtonObserverToEvent( portConn,outView ) );
        serialPort.notifyOnDataAvailable(true);
    }
    protected void addApplicationEventHandler() throws Exception{
        ArduinoEventsHandler evh = new ArduinoEventsHandler(outView);
        njs.insertInEventHandlerWaitQueue(evh, SysKb.eventName);
    }
    protected void createEventPlatform(){
        njs = NodejsLike.createBasic(outView); //SysKbInfoReplInfr.getNodejsLike();
        njs.setWithExternalEvents();
        new Thread(){
            public void run(){
                try {
                    njs.mainLoop();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }.start();
    }
/*
 * MAIN
 */
    public static void main(String[] args) throws Exception {
        new SerialPortButtonEventMain(null).doJob();
    }
}
```

Listing 1.11. SerialPortButtonEventMain.java

5 An application with messages and interrupts

```java
package it.unibo.noawtsupports.arduino.intro.appl;
import it.unibo.is.interfaces.IOutputView;
import it.unibo.noawtsupports.arduino.intro.SerialPortRequestResponseMain;


public class SerialPortRequestResponseButtonMain extends SerialPortRequestResponseMain{
    protected boolean goon = false;
    protected boolean applObserverSet = false;

    public SerialPortRequestResponseButtonMain(IOutputView outView){
        super(outView);
    }
    public void doJob(){
        try {
            init();

            sendMessage("First msg");
            getTheAnswer();
            sendMessage("Hello from pc");
            getTheAnswer();
            sendMessage("Final msg from pc");
            getTheAnswer();

            //portConn.closeConnection();
            System.out.println("APPL END. Now I'll show the answer to interrupts");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    protected void init() throws Exception{
        println(" ==== STARTS. WARNING: rxtxSerial.dll must be in the classpath " + getName() );
        setConnection();
        //addSerialPortListener(); //without a listener the system does not handle interrupts
        waitForReady();
    }
    protected void sendMessage(String msg) throws Exception{
        System.out.println("Sending to arduino ... " + msg);
        portConn.sendALine(msg);
    }
    protected void getTheAnswer() throws Exception{
        if( applObserverSet ) waitContinue();
        else println("Answer="+ portConn.receiveALine());
    }
    public synchronized void waitContinue() throws Exception{
        while( !goon ) wait();
        goon = false;
    }
    //setInput is called by ApplObserver
    public synchronized void setInput(String msg) throws Exception {
        goon = true;
        println( msg );
        notifyAll();
    }
    protected void addSerialPortListener() throws Exception{
        serialPort.addEventListener( new ApplObserver(portConn,outView,this) );
        serialPort.notifyOnDataAvailable(true);
        applObserverSet = true;
    }
/*
 * MAIN
 */
    public static void main(String[] args) throws Exception {
        new SerialPortRequestResponseButtonMain(null).doJob();
    }
```

```
65
66 }
```

Listing 1.12. `SerialPortRequestResponseButtonMain.java`

6 An application with messages and events

The application defined in *SerialPortMessageAndEventMain* exchanges some message with Arduino while using physical button managed by Arduino that emits an event named "`arduinoEvent`" defined in *SysKb.eventName*:

```
1  package it.unibo.noawtsupports.arduino.intro.appl;
2  import java.awt.Color;
3  import java.io.FileInputStream;
4
5  import gnu.io.SerialPort;
6  import it.unibo.arduino.serial.ISerialPortInteraction;
7  import it.unibo.arduino.serial.SerialPortConnSupport;
8  import it.unibo.arduino.serial.SerialPortSupport;
9  import it.unibo.baseEnv.basicFrame.EnvFrame;
10 import it.unibo.button.bridge.DevButton;
11 import it.unibo.domain.interfaces.IDevButton;
12 import it.unibo.domain.interfaces.IDevButtonArduino;
13 import it.unibo.event.interfaces.INodejsLike;
14 import it.unibo.inforeply.infrastructure.SysKbInfoReplInfr;
15 import it.unibo.is.interfaces.IBasicEnvAwt;
16 import it.unibo.is.interfaces.IOutputView;
17 import it.unibo.noawtsupports.arduino.intro.events.ArduinoEventsHandler;
18 import it.unibo.noawtsupports.arduino.intro.events.DevButtonArduino;
19 import it.unibo.noawtsupports.arduino.intro.events.SysKb;
20 import it.unibo.system.SituatedPlainObject;
21
22 public class SerialPortMessageAndEventMain extends SituatedPlainObject{
23     public static final String[] myEvents= new String[] { SysKb.eventName };
24     public static final String[] otherEvents= new String[] { SysKb.eventName };
25     protected String PORT_NAME = SysKb.serialPortWindows;
26     protected ISerialPortInteraction portConn;
27     protected SerialPort serialPort;
28     protected INodejsLike njs;
29
30     public SerialPortMessageAndEventMain(IOutputView outView){
31         super(outView);
32     }
33     public void doJob(){
34         try {
35             init();
36
37             sendMessage("First msg");
38             getTheAnswer() ;
39             sendMessage("Hello from pc");
40             getTheAnswer() ;
41             sendMessage("Final msg from pc");
42             getTheAnswer() ;
43
44             System.out.println("*** APPL END. Now I'll show the answer to interrupts");
45         } catch (Exception e) {
46             e.printStackTrace();
47         }
48     }
49     protected void init() throws Exception{
50         println(" ==== STARTS. WARNING: rxtxSerial.dll must be in the classpath " + getName() );
51         setConnection();
52         createEventInfrastructure( );
53         createArduinoButton();
54         createArduinoButtonEventHandler();
55         waitForReady();
56     }
57     protected void sendMessage(String msg) throws Exception{
58         println("Sending to arduino ... " + msg);
```

```
59          portConn.sendALine(msg);
60      }
61      protected void getTheAnswer() throws Exception{
62          println("     Answer="+ portConn.receiveALine());
63      }
64      protected void setConnection() throws Exception{
65          SerialPortSupport serialPortSupport = new SerialPortSupport( outView );
66          serialPort = serialPortSupport.connect(PORT_NAME, this.getClass() );
67          println("serialPort="+serialPort);
68          portConn = new SerialPortConnSupport(serialPort, outView);
69      }
70      protected void waitForReady() throws Exception{
71          println("*** Waiting for arduino STARTUP... ");
72          Thread.sleep(2000);
73      }
74      protected void createArduinoButton() throws Exception{
75          IDevButtonArduino devImpl = new DevButtonArduino("arduinoButton", null, portConn);
76          IDevButton dev = new DevButton("button", null);
77          dev.setDevImpl(devImpl);
78      }
79      protected void createArduinoButtonEventHandler() throws Exception{
80          ArduinoEventsHandler evh = new ArduinoEventsHandler(outView);
81          njs.insertInEventHandlerWaitQueue(evh, SysKb.eventName);
82      }
83      public void createEventInfrastructure( ) throws Exception{
84      try {
85          SysKbInfoReplInfr sysInfr = SysKbInfoReplInfr.getInstance( outView ,
86                                              new FileInputStream("sysconfig.pl") );
87          println( "createInfrastructure " + sysInfr);
88          sysInfr.createInfoReplInfrastructure( "1", otherEvents ); //SEE sysconfig.pl
89          njs = SysKbInfoReplInfr.getNodejsLike();
90          njs.setWithExternalEvents();
91          sysInfr.createMainLoop();
92          println( "createInfrastructure DONE " );
93      } catch (Exception e) {
94          println( "createEventInfrastructure ERROR " + e.getMessage());
95      }
96      }
97
98  /*
99   * MAIN
100  */
101     public static void main(String[] args) throws Exception {
102         IBasicEnvAwt env = // null;
103                 new EnvFrame( "SerialPortMessageAndEventMain", Color.cyan, Color.black );
104         env.init();
105         env.writeOnStatusBar("SerialPortMessageAndEventMain" + " | working ... ",14);
106         new SerialPortMessageAndEventMain(env.getOutputView()).doJob();
107     }
108
109 }
```

Listing 1.13. `SerialPortMessageAndEventMain.java`

The application defined in *SerialPortMessageAndEventMain* creates a logical button (line 68) that implements the high level `IDevButton` interface and injects in it an implementation of a physical button managed by Arduino:

6.1 DevButtonArduino

```
1  package it.unibo.noawtsupports.arduino.intro.events;
2  import it.unibo.arduino.serial.ISerialPortInteraction;
3  import it.unibo.is.interfaces.IOutputEnvView;
4  import it.unibo.button.bridge.impl.DevInputImpl;
5  import it.unibo.domain.interfaces.IDevButtonArduino;
6
7  public class DevButtonArduino extends DevInputImpl implements IDevButtonArduino {
8      protected ISerialPortInteraction portConn;
9
10     public DevButtonArduino(String name, IOutputEnvView outView, ISerialPortInteraction portConn) {
```

```
        super(name, outView);
        this.portConn = portConn;
        configure();
    }
    protected void configure() {
        try {
            portConn.getPort().addEventListener( new ButtonObserverToEvent(portConn,outView) );
            portConn.getPort().notifyOnDataAvailable(true);
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

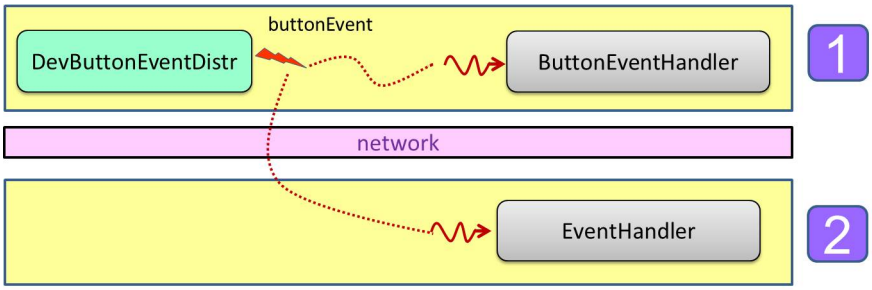
Listing 1.14. `DevButtonArduino.java`

Let us report here the definition of the class `DevInputImpl` introduced in the *it.unibo.buttonLedSystem* project:

```
package it.unibo.button.bridge.impl;
import java.util.Observable;
import it.unibo.button.bridge.SysKb;
import it.unibo.domain.interfaces.IDevInputImpl;
import it.unibo.is.interfaces.IActivityBase;
import it.unibo.is.interfaces.IOutputEnvView;
import it.unibo.system.SituatedPlainObject;

public class DevInputImpl extends SituatedPlainObject implements IDevInputImpl, IActivityBase {
protected boolean isPressed = false;
protected String name ;
    public DevInputImpl(String name, IOutputEnvView outView) {
        super(outView);
        this.name = name;
    }
    @Override
    public int getInput() throws Exception { return isPressed ? 1 : 0; }
    @Override
    public void execAction(String cmd) {
        isPressed = cmd.equals( SysKb.repHigh );
        println("DevButton isPressed=" + isPressed + " since upadatd by " + cmd);
        this.setChanged(); //!!!!
        this.notifyObservers(cmd);
    }
    public void update( boolean v){
        String cmd = v ? SysKb.repHigh : SysKb.repLow ;
        execAction( cmd );
    }
    @Override
    public void update(Observable arg0, Object arg1) {
        String vs = ""+arg1;
        isPressed = vs.equals( SysKb.repHigh );
        update( isPressed );
    }
    @Override
    public String getDefaultRep() {
        try {
            return "sensor("+this.name+","+this.getInput()+")";
        } catch (Exception e) { return "sensor("+this.name+",null)"; }
    }
    @Override
    public String getName() {
        return name;
    }
 }
```

Listing 1.15. `DevInputImpl.java`

6.2 A remote event receiver




```
1  package it.unibo.noawtsupports.arduino.intro.appl;
2  import java.awt.Color;
3  import java.io.FileInputStream;
4  import gnu.io.SerialPort;
5  import it.unibo.arduino.serial.ISerialPortInteraction;
6  import it.unibo.baseEnv.basicFrame.EnvFrame;
7  import it.unibo.event.interfaces.INodejsLike;
8  import it.unibo.inforeply.infrastructure.SysKbInfoReplInfr;
9  import it.unibo.is.interfaces.IBasicEnvAwt;
10 import it.unibo.is.interfaces.IOutputView;
11 import it.unibo.noawtsupports.arduino.intro.events.ArduinoEventsHandler;
12 import it.unibo.noawtsupports.arduino.intro.events.SysKb;
13 import it.unibo.system.SituatedPlainObject;
14
15 public class ButtonEventReceiverMain extends SituatedPlainObject{
16     public static final String[] myEvents= new String[] { SysKb.eventName };
17     public static final String[] otherEvents= new String[] { SysKb.eventName };
18     protected String PORT_NAME = SysKb.serialPortWindows;
19     protected ISerialPortInteraction portConn;
20     protected SerialPort serialPort;
21     protected INodejsLike njs;
22
23     public ButtonEventReceiverMain(IOutputView outView){
24         super(outView);
25     }
26     public void doJob(){
27         try {
28             init();
29             println("*** APPL END. Now I'll show the answer to events");
30         } catch (Exception e) {
31             e.printStackTrace();
32         }
33     }
34     protected void init() throws Exception{
35         createEventInfrastructure( );
36         createArduinoButtonEventHandler();
37     }
38     public void createEventInfrastructure( ) throws Exception{
39     try {
40         SysKbInfoReplInfr sysInfr = SysKbInfoReplInfr.getInstance( outView ,
41                                             new FileInputStream("sysconfig.pl") );
42         println( "createInfrastructure " + sysInfr);
43         sysInfr.createInfoReplInfrastructure( "2", otherEvents ); //SEE sysconfig.pl
44         njs = SysKbInfoReplInfr.getNodejsLike();
45         njs.setWithExternalEvents();
46         sysInfr.createMainLoop();
47         println( "createInfrastructure DONE " );
48     } catch (Exception e) {
49         println( "createEventInfrastructure ERROR " + e.getMessage());
50     }
51     }
52     protected void createArduinoButtonEventHandler() throws Exception{
53         ArduinoEventsHandler evh = new ArduinoEventsHandler(outView);
54         njs.insertInEventHandlerWaitQueue(evh, SysKb.eventName);
55     }
56
57 /*
```

```
 * MAIN
 */
   public static void main(String[] args) throws Exception {
       IBasicEnvAwt env = // null;
               new EnvFrame( "ButtonEventReceiverMain", Color.lightGray, Color.black );
       env.init();
       env.writeOnStatusBar("ButtonEventReceiverMain" + " | working ... ",14);
       new ButtonEventReceiverMain(env.getOutputView()).doJob();
   }

}
```

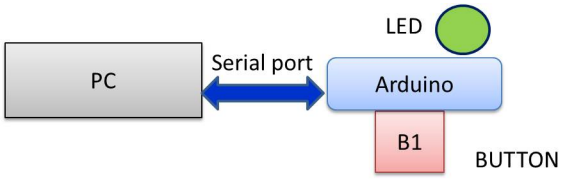
Listing 1.16. `ButtonEventReceiverMain.java`

6.3 The system configuration file

```
%sysconfig.pl
%=========================
%%startUpMode(synch).

%====================================
node( 1, pc, "localhost", 8025 ).
node( 2, receiver, "localhost", 8035 ). %% 192.168.43.229
```

Listing 1.17. `sysconfig.pl`

7 The ButtonLed system



7.1 ButtonLed.ino (on Arduino UNO)

```
/*
 ButtonLed.ino
*/
int NEW_LINE_ASCII = 10;
int CR_ASCII = 13;
int led = A0;
int buttonSw = 3; //7          //ARDUINO UNO: pin 3 -> MAPS TO interrupt 1
String inputString = "";        // a string to hold incoming data
boolean stringComplete = false; // whether the string is complete
/*
* Interrupt handling with debouncing
*/
boolean debouncing(){
static unsigned long lastInterruptTime = 0;
static int lastSw = 0;
unsigned long interruptTime = millis();
  if( (interruptTime - lastInterruptTime) > 100 ){
    lastInterruptTime = interruptTime;
    return true;
  }
  return false;
}
void buttonInterruptHandler(){
  boolean ok = debouncing();
```

```
  if( ok ){
     int sw = digitalRead(buttonSw);
     if( sw == 1 )
     //Serial.println("arduinoInterrupt(button,value("+String(sw)+"))");
     Serial.println("arduinoInterrupt(button)");
   }
}
/*
*  Read a line from the serial port
*/
void serialInput() { //ex serialEvent ...
   while (Serial.available()) {
     int inChar = Serial.read();
     if ( inChar == NEW_LINE_ASCII ) {
       stringComplete = true;
    } else inputString = inputString + (char)inChar;
  }//while
}
/*
* setup
*/
void setup() {
  pinMode(led, OUTPUT);
  pinMode(buttonSw, INPUT);
  Serial.begin(9600);
  inputString.reserve(256);
  digitalWrite(buttonSw, HIGH); //attach pull up => unpressed = 1
  attachInterrupt(1, buttonInterruptHandler, CHANGE); // RISING CHANGE FOLLING LOW
}
/*
* LOOP
*/
void loop() {
     serialInput() ;
     if (stringComplete) {
        //Serial.println( inputString + " n=" + String(n) );
        if( inputString == "HIGH" )
          digitalWrite(led, HIGH);
        else
          digitalWrite(led, LOW);
        inputString = "";
        stringComplete = false;
     }
}
```

Listing 1.18. `ButtonLed.ino`

7.2 ButtonLedArduinoEventMain (on pc)

```
package it.unibo.noawtsupports.arduino.intro.buttonled;
import java.awt.Color;
import java.io.FileInputStream;
import gnu.io.SerialPort;
import it.unibo.arduino.serial.ISerialPortInteraction;
import it.unibo.arduino.serial.SerialPortConnSupport;
import it.unibo.arduino.serial.SerialPortSupport;
import it.unibo.baseEnv.basicFrame.EnvFrame;
import it.unibo.button.bridge.DevButton;
import it.unibo.domain.interfaces.IDevButton;
import it.unibo.domain.interfaces.IDevButtonArduino;
import it.unibo.event.interfaces.INodejsLike;
import it.unibo.inforeply.infrastructure.SysKbInfoReplInfr;
import it.unibo.is.interfaces.IBasicEnvAwt;
import it.unibo.is.interfaces.IOutputView;
import it.unibo.noawtsupports.arduino.intro.events.DevButtonArduino;
import it.unibo.noawtsupports.arduino.intro.events.SysKb;
import it.unibo.system.SituatedPlainObject;

public class ButtonLedArduinoEventMain extends SituatedPlainObject{
    public static final String[] myEvents= new String[] { SysKb.eventName };
```

```
22      public static final String[] otherEvents= new String[] { SysKb.eventName };
23      protected String PORT_NAME = SysKb.serialPortWindows;
24      protected ISerialPortInteraction portConn;
25      protected SerialPort serialPort;
26      protected INodejsLike njs;
27
28      public ButtonLedArduinoEventMain(IOutputView outView){
29          super(outView);
30      }
31      public void doJob(){
32          try {
33              init();
34              println("*** APPL END. Now I'll handle events");
35          } catch (Exception e) {
36              e.printStackTrace();
37          }
38      }
39      protected void init() throws Exception{
40          println(" ==== STARTS. WARNING: rxtxSerial.dll must be in the classpath " + getName() );
41          setConnection();
42          createEventInfrastructure( );
43          createArduinoButton();
44          createArduinoButtonEventHandler();
45          waitForReady();
46      }
47      protected void setConnection() throws Exception{
48          SerialPortSupport serialPortSupport = new SerialPortSupport( outView );
49          serialPort = serialPortSupport.connect(PORT_NAME, this.getClass() );
50          println("serialPort="+serialPort);
51          portConn = new SerialPortConnSupport(serialPort, outView);
52      }
53      protected void waitForReady() throws Exception{
54          println("*** Waiting for arduino STARTUP... ");
55          Thread.sleep(2000);
56      }
57      protected void createArduinoButton() throws Exception{
58          IDevButtonArduino devImpl = new DevButtonArduino("arduinoButton", null, portConn);
59          IDevButton dev = new DevButton("button", null);
60          dev.setDevImpl(devImpl);
61      }
62      protected void createArduinoButtonEventHandler() throws Exception{
63          ArduinoButtonEventHandler evh = new ArduinoButtonEventHandler(outView, portConn);
64          njs.insertInEventHandlerWaitQueue(evh, SysKb.eventName);
65      }
66      public void createEventInfrastructure( ) throws Exception{
67      try {
68          SysKbInfoReplInfr sysInfr = SysKbInfoReplInfr.getInstance( outView ,
69                                              new FileInputStream("sysconfig.pl") );
70          println( "createInfrastructure " + sysInfr);
71          sysInfr.createInfoReplInfrastructure( "1", otherEvents ); //SEE sysconfig.pl
72          njs = SysKbInfoReplInfr.getNodejsLike();
73          njs.setWithExternalEvents();
74          sysInfr.createMainLoop();
75          println( "createInfrastructure DONE " );
76      } catch (Exception e) {
77          println( "createEventInfrastructure ERROR " + e.getMessage());
78      }
79      }
80
81  /*
82   * MAIN
83   */
84      public static void main(String[] args) throws Exception {
85          IBasicEnvAwt env = // null;
86                  new EnvFrame( "SerialPortMessageAndEventMain", Color.cyan, Color.black );
87          env.init();
88          env.writeOnStatusBar("SerialPortMessageAndEventMain" + " | working ... ",14);
89          new ButtonLedArduinoEventMain(env.getOutputView()).doJob();
90      }
91
92  }
```

Listing 1.19. `ButtonLedArduinoEventMain.java`

7.3 ArduinoButtonEventHandler (on pc)

The handler *ArduinoButtonEventHandler* performs a switch of the led each time a button event is perceived. The led switch is done by sendong a message to Arduino via the serial port connection.

```
1  package it.unibo.noawtsupports.arduino.intro.buttonled;
2  import it.unibo.arduino.serial.ISerialPortInteraction;
3  import it.unibo.event.interfaces.IEventItem;
4  import it.unibo.is.interfaces.IOutputView;
5  import it.unibo.nodelike.platform.EventHandler;
6
7  public class ArduinoButtonEventHandler extends EventHandler {
8  protected ISerialPortInteraction portConn;
9  protected boolean on = false;
10     public ArduinoButtonEventHandler( IOutputView outView, ISerialPortInteraction portConn ) throws Exception {
11         super( "butHan", outView );
12         this.portConn = portConn;
13     }
14     @Override
15     public void doJob() throws Exception {
16         IEventItem ev = this.getEventItem();
17         showMsg(getName() + ":" + ev.getEventId()+ " " + ev.getMsg() );
18         if( on ) portConn.sendALine("HIGH");
19         else portConn.sendALine("LOW");
20         on = ! on;
21     }
22  }
```

Listing 1.20. `ArduinoButtonEventHandler.java`

References

1. A. Natali. Handling (sensor) inputs: the ButtonCounterLed system. https://137.204.107.21/Readings/xtext/InputHandling.pdf.