Universidad de Oriente.

Nucleó Anzoátegui.

Escuela de Ingeniería y Ciencias Aplicadas.

Departamento de Ingeniería.

Arquitectura Avanzada del Computador.



Desarrollo de una Aplicación para el Control de Temperatura en Hornos de una Empresa ubicada en la zona norte del Estado Anzoátegui.

Tutor Profesor Ing. Alfonso Alfonsí

Grupo N° 1. Integrantes:

María Cardoza C.I: 6.142.718

Luis Correa C.I: 19.840.230

Copyright © 2015 por Luis Correa & María Cardoza. Todos los derechos reservados.

Barcelona, 06 de Julio de 2015

**Tabla de Contenidos**

**I.** **Introducción** 4

**II.** **Objetivos** 5

A. Objetivo general 5

B. Objetivos específicos 5

**III.** **Subsistema Hardware** 5

**IV.** **Subsistema de Software** 7

A. Metodologías y Herramientas 7

B. Código del programa cargado en el Arduino Uno 8

C. Código del programa hecho en java 9

**V.** **Desarrollo del Sistema de Control** 12

A. Requisitos Funcionales 12

B. Requisitos No Funcionales 13

**VI.** **Diagramas a emplearse** 14

A. Diagrama de Casos de Uso 14

B. Diagrama de Clases 15

C. Diagrama de Componentes 16

D. Diagrama de Secuencia 17

E. Diagrama de Estados de Clase DetectorTemperatura 18

**VII.** **Guía de configuración** 18

A. Para plataformas Windows de 32 bits 18

B. Para plataformas Windows de 64 bits 19

**VIII.** **Conclusiones** 19

**IX.** **Lista de referencias** 20

# **Introducción**

En el desarrollo de los sistemas electrónicos, es evidente el impacto que representan los microcontroladores en áreas como la instrumentación, control, automatización industrial, robótica, domótica e incluso en dispositivos de la vida diaria (sensores que detectan movimientos y prenden bombillos). Además se afirma que en estos tiempos existe una orientación estratégica hacia el desarrollo de software para la codificación y control de los microcontroladores. Un ejemplo de ello es el Arduino IDE, que es un software con aplicaciones en las áreas antes mencionadas.

Un sistema de control basado en microcontroladores puede atender uno o varios dispositivos externos y sus respectivas funcionalidades. Ante esta situación una de las vías para el desarrollo del control es utilizar lazos independientes. Se utilizará los Diagramas de Casos de Uso para mostrar el empleo de estos en el Sistema propuesto (ver Figura N°1) y el Diagrama de Clases se muestra en la Figura N°2. Las Figuras N°3, 4 y 5 muestran los diferentes diagramas del sistema.

Este trabajo está estructurado de la siguiente manera:

* Objetivos
* Diagramas a emplearse

# **Objetivos**

### Objetivo general

Utilizar los Diagramas de Casos de Uso, Diagramas de Clase, Diagramas de Componentes, Diagramas de Secuencia, Diagramas de Maquina de Estados, en el estudio del Desarrollo de una Aplicación para el Control de Temperatura en Hornos de una Empresa ubicada en la zona norte del Estado Anzoátegui.

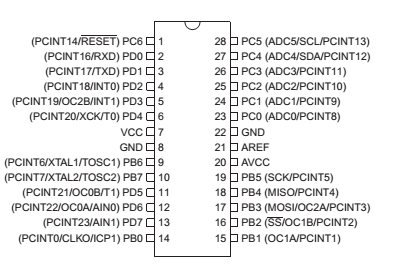
### Objetivos específicos

* Describir la situación actual del seguimiento de las actividades con el control de temperatura de la empresa.
* Identificar los requerimientos de información del Sistema Empotrado (SE).
* Modelar la estructura del software, y de las interfaces asociadas con el sistema.
* Diseñar las interfaces de la aplicación utilizadas para el control de la temperatura.
* Codificar los módulos de la nueva aplicación.

# **Subsistema Hardware**

Como plataforma de hardware para el desarrollo de la investigación se seleccionó Arduino Uno, tomando como referencia los intereses actuales en cuanto al Desarrollo e Implementación de Hardware Libre en Venezuela. El cómo software para el control del Arduino se usó el Arduino IDE.

Arduino Uno es la placa Arduino más popular. Su cerebro puede estar conformado por el microcontrolador Atmel AVR ATmega328P. En la Figura N° 1 se muestra la disposición de los pines del microcontrolador. Entre otras características el Arduino Uno posee una conexión USB, un conector para la alimentación, una cabecera ICSP (In-Circuit Serial Programming), y un botón de reset. Es un sistema de desarrollo sencillo y de bajo costo que permite la realización de múltiples diseños, en la tabla 1 se muestran las características técnicas del ATmega328P.

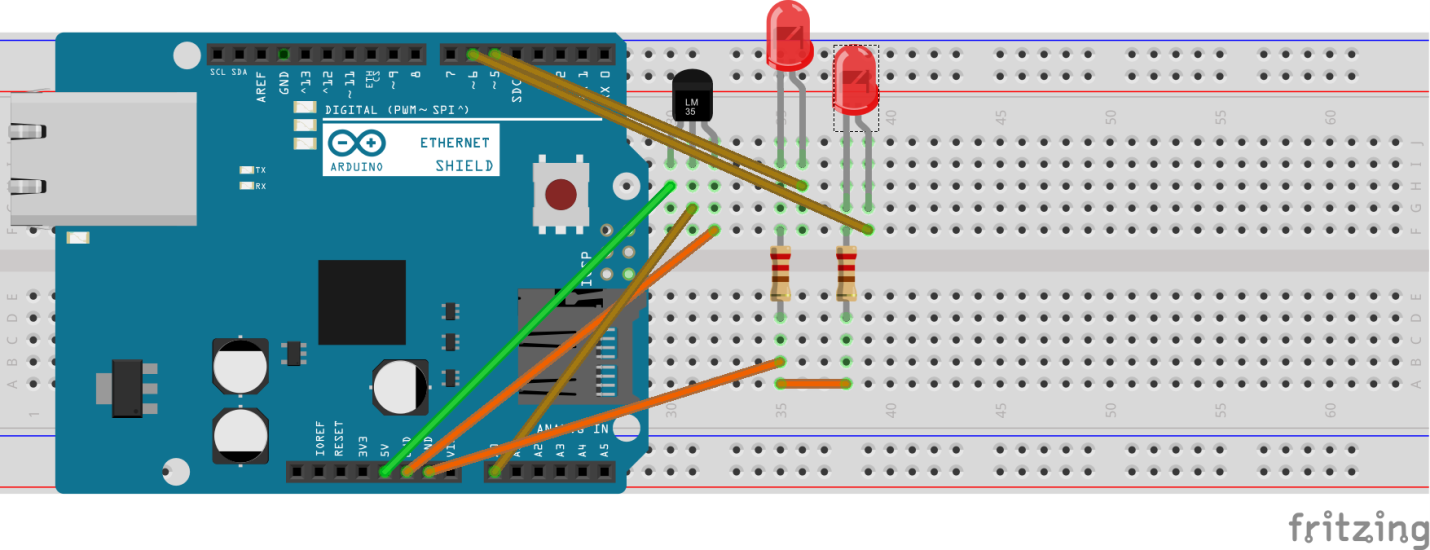


**Figura N° 1. Arduino Uno ATmega328P**

**Tabla N° 1. Características técnicas**

**de los microcontroladores ATmega328P**

|  |  |
| --- | --- |
| PARÁMETROS | VALORES |
| Flash | 32 Kbytes |
| RAM | 2 Kbytes |
| Cantidad Pines | 28 |
| Frecuencia máxima de operación | 20 MHz |
| CPU | 8-bit AVR |
| Número de Canales variables | 16 |
| Pines máximos de E/S | 26 |
| Interrupciones externas | 24 |

****

**Figura N° 2.Diagrama Esquemático del Sistema de Control de Temperatura.**

# **Subsistema de Software**

### Metodologías y Herramientas

Desarrollo de una Aplicación para el Control de Temperatura en Hornos de una empresa ubicada en la zona norte del Estado Anzoátegui.

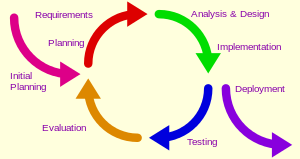
Según la aplicación de metodologías para el desarrollo de un Sistema para el Control de Temperatura en Hornos, dependerá del tipo de sistema y de los objetivos que se persiguen. En un sistema embebido estos procedimientos deben cumplir con las condicionantes en la implementación de este tipo de sistemas.

La estructura del sistema se divide en dos partes fundamentales:

* Los requerimientos lógicos, los cuales están caracterizados por el Modelado del Comportamiento de Datos, Funciones y Control entre el Sistema y el ambiente (descomposición funcional del sistema en la forma de Diagramas de Flujo de Datos).
* Los requerimientos temporales, que permiten predecir el comportamiento y garantizar la ejecución del sistema en el peor caso cuando se aplica un algoritmo de planificación.

Los procedimientos que se deben cumplir con las condicionantes en la implementación (ver figura 1) son:

* Planificación inicial.
* Planificación y Requerimiento.
* Análisis y Diseño, Implementación 🡪 Despliegue.
* Prueba.
* Evaluación 🡪 Iteración.



**Figura N° 3. Proceso de Desarrollo Iterativo.**

Entre las herramientas presentadas para cumplir con los requerimientos temporales del sistema se encuentran las pruebas y los temporizadores e interrupciones propias del microcontrolador, siendo estas dos últimas muy importantes, ya que como es bien sabido, la forma de trabajo de los microcontroladores es secuencial y la única manera que el planificador pueda en todo momento estar monitorizando y controlando la ejecución de la tareas, es a través de estas herramientas.

Las herramientas computacionales utilizadas fueron el Arduino IDE, donde se realizó el código que fue almacenado en el microcontrolador, para la simulación de cada una de las etapas del sistema se utilizó ISIS Proteus, en su versión 7.2.

### Código del programa cargado en el Arduino Uno

const int sensor = 0;

long miliVolts;

long temperatura;

int brillo;

void setup(){

Serial.begin(9600);

}void loop(){

miliVolts = analogRead(sensor)\*5000L/1023;

temperatura = miliVolts/10;

brillo=map(temperatura,10,40,0,255);

brillo = constrain(brillo,0,255);

Serial.println(temperatura);

delay(1000);

}

### Código del programa hecho en java

**Código de Aplicación en java que se conecta con el puerto serial, clases usadas ControlDeTemperatura.java, Sirena.java, DetectorDeTemperatura.java**

**Clase principal.**

public class ControlDeTemperatura{

public static void main(String[] args) throws Exception {

DetectorDeTemperatura main = new DetectorDeTemperatura();

main.initialize();

Thread t=new Thread() {

@Override

public void run() {

System.out.println(main.serialPort);

try {

Thread.sleep(1000000);

} catch (InterruptedException ie) {

}

}

};

t.start();

System.out.println("Started");

}

}

**Clase de detection de temperatura**

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.OutputStream;

import gnu.io.CommPortIdentifier;

import gnu.io.SerialPort;

import gnu.io.SerialPortEvent;

import gnu.io.SerialPortEventListener;

import java.util.Enumeration;

public class DetectorDeTemperatura implements SerialPortEventListener {

SerialPort serialPort; /\*\* The port we're normally going to use. \*/

private static final String PORT\_NAMES[] = {

// "/dev/tty.usbserial-A9007UX1", // Mac OS X

// "/dev/ttyACM0", // Raspberry Pi

// "/dev/ttyUSB0", // Linux

"COM3", // Windows

};

// public String get() throws IOException{return input.readLine();}

/\*\*

\* A BufferedReader which will be fed by a InputStreamReader

\* converting the bytes into characters

\* making the displayed results codepage independent

\*/

private BufferedReader input;

/\*\* The output stream to the port \*/

private OutputStream output;

/\*\* Milliseconds to block while waiting for port open \*/

private static final int TIME\_OUT = 2000;

/\*\* Default bits per second for COM port. \*/

private static final int DATA\_RATE = 9600;

public void initialize() {

// the next line is for Raspberry Pi and

// gets us into the while loop and was suggested here was suggested http://www.raspberrypi.org/phpBB3/viewtopic.php?f=81&t=32186

System.setProperty("gnu.io.rxtx.SerialPorts","COM3"); //"/dev/ttyACM0");

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();

for (String portName : PORT\_NAMES) {

if (currPortId.getName().equals(portName)) {

portId = currPortId;

break;

}

}

}

if (portId == null) {

System.out.println("Could not find COM port.");

return;

}

try {// open serial port, and use class name for the appName.

serialPort = (SerialPort) portId.open(this.getClass().getName(),TIME\_OUT);

// set port parameters

serialPort.setSerialPortParams(DATA\_RATE,SerialPort.DATABITS\_8,

SerialPort.STOPBITS\_1,SerialPort.PARITY\_NONE);

// open the streams

input = new BufferedReader(new InputStreamReader(serialPort.getInputStream()));

output = serialPort.getOutputStream();

// add event listeners

serialPort.addEventListener(this);

serialPort.notifyOnDataAvailable(true);

} catch (Exception e) {

System.err.println(e.toString());

}

}

/\*\*

\* This should be called when you stop using the port.

\* This will prevent port locking on platforms like Linux.

\*/

public synchronized void close() {

if (serialPort != null) {

serialPort.removeEventListener();

serialPort.close();

}

}

/\*\*

\* Handle an event on the serial port. Read the data and print it.

\* @param oEvent

\*/

@Override

public synchronized void serialEvent(SerialPortEvent oEvent) {

if (oEvent.getEventType() == SerialPortEvent.DATA\_AVAILABLE) {

try {

String inputLine=input.readLine();

System.out.println(inputLine);

if(Integer.parseInt(inputLine)>30){

hilo=new Thread(alarma);

alarma.start(true);

hilo.start();

}else{

if(hilo!=null){

alarma.start(false);

hilo.interrupt();

hilo=null;

}

}

} catch (Exception e) {

System.err.println(e.toString());

}

}

}

private Thread hilo=null;

private Sirena alarma=new Sirena();

}

**Clase de Sonido cuando aumenta la temperatura mas de lo normal**

public class Sirena implements Runnable{

private javax.sound.sampled.Clip clip;

private java.io.InputStream path;

public Sirena(){

}

public Sirena(String a){

this.path = getClass().getResourceAsStream(a);

try {

this.clip=javax.sound.sampled.AudioSystem.getClip();

this.clip.open(javax.sound.sampled.AudioSystem.getAudioInputStream(path));

clip.setFramePosition(0);

}

catch (Exception e) {

javax.swing.JOptionPane.showMessageDialog(null,e.getMessage());

}

}

public void play(){

if(clip.isRunning()){

clip.start();

}

else{

clip.setFramePosition(0);

clip.start();

}

}

@Override

public void run() {

Sirena sonido=new Sirena("/res/stall\_warning.wav");//C:/Users/Yolanda Yancel/Documents/MEGAsync/proyectos netbeans/electivas tecnicas/10\_arquitectura\_avanzada/EjemploPuertoSerial/src

while(start){

try{ Thread.sleep(25); }

catch(Exception exc){}

finally{

sonido.play();

}

}

}

public void start(boolean s){start=s;}

private boolean start=true;

}

# **Desarrollo del Sistema de Control**

### Requisitos Funcionales

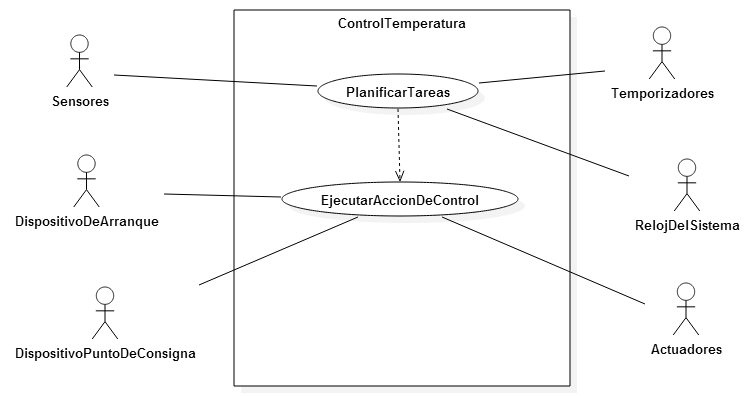
* El sistema debe informar al usuario en todo momento el valor de la temperatura a la cual se encuentra cada horno.
* Para el Sistema de Control se utilizó cinco lazos independientes (cuatro de control y uno de monitoreo), representando cada uno de ellos un conjunto finito de tareas críticas, periódicas, independientes y apropiables; donde el mismo fue validado mediante una serie de simulaciones.
* Para la planta se utiliza como base un sistema de control con el sensor LM35, de amplia utilización en esta área del conocimiento.
* Cada tarea del sistema estará formada por un lazo independiente, siendo cada una de ellas responsable de manejar un proceso en particular del sistema.

### Requisitos No Funcionales

* Seguridad: El administrador tendrá una ventana diferente a la del empleado con lo cual el empleado no tendrá acceso a funciones que si podrá acceder el administrador.
* Accesibilidad: Este es un programa, que solo estará diseñado para escritorio y tendrá dos tipos de usuarios: empleados y administrador. Administrador tiene habilitada todas las funciones, envió y lectura de datos, etc, mientras que la sesión empleado solo podrá acceder a la lectura de datos.
* Usabilidad: Se implementaron un esquema para optimizar el rápido la lectura e ingreso de datos.
* portabilidad: Portable debido a que se utiliza la Orientación a Objetos en el código Java.
* Bajo costo: La aplicación se realizó con herramientas de software libre y no tendremos que pagar ninguna licencia.
* Escalabilidad: Esta aplicación se puede seguir realizando en un futuro más modificaciones, dependiendo de las necesidades que tenga el cliente o desarrollar una versión más actualizada.
* Mantenibilidad: El mantenimiento del software se realizará cuando el cliente requiera q tenga una función adicional al programa.
* Interfaz Gráfica: Garantiza la fácil utilización y alta velocidad de procesamiento de datos.

# **Diagramas a emplearse**

### Diagrama de Casos de Uso



**Figura N° 4. Diagrama de Casos de Uso.**

* **Planificar Tareas:**

Descripción:

Realiza la tarea de verificar la temperatura de los hornos a través de los sensores, por medio de los temporizadores y el reloj del sistema se puede controlar los tiempos de respuesta,.

Actores:

Sensores, Temporizadores, Reloj del Sistema.

Precondición:

Los Sensores deben estar conectados al Arduino Uno y tocar al Horno para poder revisar la temperatura y enviarla al Arduno Uno.

Flujo normal:

Los Sensores envían valores continuamente sobre las variaciones de temperatura.

Flujo alternativo:

Si el Sensor no está bien conectado o no toca al Horno no se obtendrán resultados correctos.

Poscondiciones:

Los sensores envían los valores obtenidos al Arduino Uno.

* **Ejecutar Acción de Control:**

Descripción:

Realiza la tarea de activar los actuadores, y de arrancar el Sistema.

Actores:

Dispositivo de Arranque, Actuadores, Dispositivo Punto de Consigna.

Precondición:

Los Actuadores, El dispositivo que inicia el Arranque del Sistema deben estar conectados al Arduino Uno.

Flujo normal:

El Arduino activara los actuadores cuando se detecte alguna temperatura no deseada.

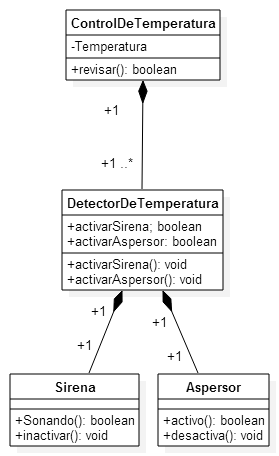
Flujo alternativo:

Si el Actuador no está bien conectado no se podrá bajar la temperatura cuando este muy alta, si el Arranque del Sistema no está bien conectado el Sistema jamás iniciara.

Poscondiciones:

Los Actuadores son activados por el Arduino Uno mientras baja la temperatura.

### Diagrama de Clases

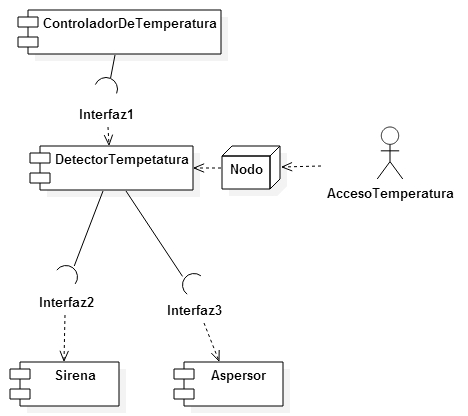


**Figura N°5. Diagrama de Clases**

La clase ControlDeTemperatura tiene que siempre tener una lista con al menos una clase DetectorDeTemperatura, también puede tener muchas.

La clase DetectorDeTemperatura tiene siempre que tener una Sirena y un Aspersor.

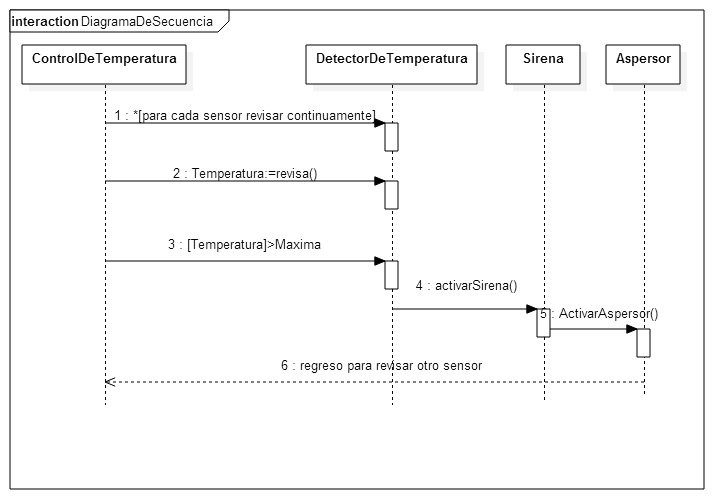
### Diagrama de Componentes



**Figura N°6. Diagrama de Componentes del Sistema**

En el Diagrama de visualiza que se tiene un acceso para la temperatura y las interfaces de conexión de los componentes DetectorTemperatura, Sirena, Aspersor, ControlDeTemperatura.

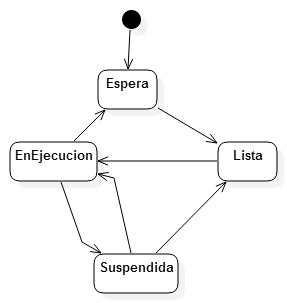
### Diagrama de Secuencia



**Figura N°7. Diagrama de Secuencia**

Diagrama de Secuencia para el control de Temperatura en hornos en el cual se hacen continuas revisiones por cada sensor de temperatura para detectar si se superó la temperatura máxima para activar la alarma y los aspersores, si no se supera la temperatura máxima en un sensor se revisa el siguiente y así sucesivamente.

### Diagrama de Estados de Clase DetectorTemperatura



**Figura N°8. Diagrama de Estado**

* Espera: estado inicial de la tarea. La tarea se encuentra a la espera hasta su activación, código 00.
* Lista: estado indica que Ti terminado que cada vez que se produce una nueva activación, código 10. Es a partir de este momento, cuando el programa toma en cuenta a la tarea en cuestión.
* En ejecución: indica que Ti está haciendo uso del procesador en ese momento, código 01.
* Suspendida: se utiliza para indicar que la tarea fue expulsada por el programa. Tiene  
  dos posibles transiciones, código 01 o 10.

# **Guía de configuración**

## Para plataformas Windows de 32 bits

* Descargar rxtx 2.1-7r2 (stable) de la página http://rxtx.qbang.org/wiki/index.php/Download
* Agregar a las variables de entorno RXTXcomm.jar.
* Copie el win32 / rxtxSerial.dll en C: \ Windows \ System32 Anexar el directorio que contiene rxtxSerial.dll en su variable de entorno PATH.

## Para plataformas Windows de 64 bits

* Descargar rxtx 2.1-7r2 (stable) de la página http://rxtx.qbang.org/wiki/index.php/Download
* Agregar a las variables de entorno RXTXcomm.jar.
* Copie el win32 / rxtxSerial.dll en C: \ Windows \ SysWOW64 Anexar el directorio que contiene rxtxSerial.dll en su variable de entorno PATH.

# **Conclusiones**

* El Diagrama de Casos de Uso, en el presente trabajo es un tipo de Diagrama para modelar la interacción de los Actores (entidades externas u otros sistemas) con el sistema.
* El Diagrama de Clases, en el presente trabajo es un tipo de Diagrama para modelar la estructura de las Clases y su relación con otras Clases en nuestro Sistema.
* El Diagrama de Componentes, en el presente trabajo es un tipo de Diagrama para modelar como nuestro sistema de [software](https://es.wikipedia.org/wiki/Software) es dividido en [componentes](https://es.wikipedia.org/wiki/Componente_de_software) y muestra las [dependencias](https://es.wikipedia.org/w/index.php?title=Dependencia_%28UML%29&action=edit&redlink=1) entre estos componentes. Los componentes físicos incluyen [archivos](https://es.wikipedia.org/wiki/Archivo_%28computaci%C3%B3n%29), cabeceras, [bibliotecas compartidas](https://es.wikipedia.org/wiki/Biblioteca_%28programaci%C3%B3n%29), [módulos](https://es.wikipedia.org/wiki/M%C3%B3dulo_%28programaci%C3%B3n%29), [ejecutables](https://es.wikipedia.org/wiki/Ejecutable), o [paquetes](https://es.wikipedia.org/wiki/Paquete_de_software).
* El Diagrama de Secuencia, en el presente trabajo es un tipo de Diagrama para modelar la interacción entre objetos, entre ellos el controlador de temperatura, el detector de temperatura, la sirena y el aspersor.
* Diagrama de Estado, en el presente trabajo nos permite mostrar los diferentes estados que puede adquirir una clase en nuestro Sistema, los cuales son espera, lista o terminada, en ejecución y suspendida para el objeto detector de temperatura.

# **Lista de referencias**

* Alfonsí, A. (2013). Técnica Dinámica para Ajustar las Necesidades Energéticas de los Sistemas Empotrados de Control de Tiempo Real Autónomos. Trabajo de Ascenso Profesor Titular. Departamento de Computación y Sistemas. Universidad de Oriente, Núcleo Anzoátegui, Barcelona, Venezuela.
* James Rumbaugh, Ivar Jacobson, Grady Booch 2007. El Lenguaje de Modelado Unificado Ed 2. Madrid. Addison Wesley.
* Código fuente de ejemplo y guía de configuración java http://playground.arduino.cc/Interfacing/Java
* Vídeo explicativo de configuración https://www.youtube.com/watch?v=43Vdpz1YmdU