UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Sistema de gestión automatizada de la calefacción de un hogar basada en Arduino y Java

Trabajo Fin de Grado
Grado en Ingeniería Informática

Autor: Juan Carlos González Torrijos
Tutor: Joan Fons i Cors

2014/2015

Primero quiero agradecer a Joan Fons, que ha servido de tutor, por ayudarme y atenderme en todas las dudas y problemas que he tenido en el desarrollo del proyecto.

También quiero agradecer a Lidia por estar ahí cada día apoyandome.

Gracias a Carlos por toda la ayuda.
Y gracias a mi familia por estar conmigo cada día

Resumen

El objetivo de este proyecto es implementar un sistema de gestión automatizada de la calefacción por suelo radiante de un hogar, de forma que el usuario de dicho sistema solo se preocupe de indicar la temperatura deseada en cada habitación del hogar y el sistema se encargará de alcanzar y mantener dicha temperatura dentro de unos rangos de tolerancia adecuados.

Para ésto, el sistema se encargará de monitorizar tanto las temperaturas del agua por todo el circuito de la casa, como de regular el circuito de producción de calor. Se diseñará y prototipará una infraestructura física basada en la plataforma Arduino.

Sobre esta capa física, se diseñará, siguiendo los principios de la Internet de las Cosas, una capa software que permitirá controlar y acceder a este sistema. Esta capa software contendrá la lógica de regulación automatizada del sistema, y proveerá a los usuarios de una interfaz de control (web/móbil) y parametrización del sistema (desarrollado en Java).

En el proyecto, se tomará como referencia una instalación real, y se desarrollará un prototipo teórico (usando simuladores para los componentes físicos), y se plantea como objetivo poder llegar a hacer un deployment sobre una infraestructura Arduino real.

Palabras clave: Aplicaciones de la Internet de las Cosas, Hogares Inteligentes, Arduino

Abstract

The objective of this project is to implement an automatic management system of the radiant ground heating of a home. The user of this system only has to select the temperature in each room of the house. The system will work to achieve and maintain this temperature within an appropriate tolerance range.

The system will also work to check the water's temperature in every part of the house, as well as control the production of heat. It will design and prototype a physical infrastructure based on the platform Arduino.

Over this physic layer, it will design a software layer based on the philosophy of the Internet of Things that will give you control and access to the system. This software layer will regulate the system, and provide a control interface for the users.

In the project, it will take as a reference a real installation, and develop a theoretical prototype (using simulators for the physical components), and raise the objective of how to achieve a deployment in a real Arduino infrastructure.

Keywords: Applications of the Internet of Things, Intelligent Homes, Arduino.

Tabla de contenidos

Tabla de contenidos

Tabla de contenidos: Figuras

1 Introducción

El presente texto es la memoria del proyecto de un sistema automatizado de gestión de la calefacción por suelo radiante de un hogar. Se va a dividir en cuatro grandes bloques:

-El primero de ellos es una explicación de todas las tecnologías que se van a emplear para desarrollar el sistema.

-En el segundo bloque encontraremos un análisis del sistema de calefacción, y un análisis de como debería de ser nuestro sistema.

-El siguiente bloque es el de diseño, se completará la fase de análisis, acercándonos más a la implementación añadiendo más elementos.

-El último bloque es la parte principal de la memoria, encontraremos la implementación de todo el sistema. Se explicará detalladamente los puntos más importantes, así como los problemas que se han encontrado y su solución. Este bloque a su vez se puede dividir en tres grandes apartados, que corresponden a las tres grandes partes en las que se ha dividido el sistema que hemos desarrollado.

Por último se hablará de las conclusiones a las que hemos llegado y el trabajo futuro que queda por realizar. También habrá un apartado de bibliografía, donde se podrá ver los documentos y páginas web que hemos visitado para poder desarrollar el proyecto.

1.1 Motivación

Que la domótica es el futuro de las viviendas y edificios es un hecho. Hoy en día, poco a poco se va introduciendo en los edificios mas modernos, pero le está costando dar pasos adelante. Uno de los problemas a los que se enfrenta actualmente, es que distintas empresas hacen sus propios sistemas para controlar sus elementos que instalan en las casas. Así, por ejemplo, hay un controlador para la alarma de la casa, un controlador distinto para las luces, otro más para la calefacción, y así un largo etc. Pero no están interconectados entre sí, ni hay posibilidad de conectar con ellos ¿Es esto realmente funcional? ¿Es realmente el futuro? Pensamos que no.

Conforme me enseñó mi profesor y tutor del presente proyecto, el futuro es "el Internet de las cosas", todo tiene que estar interconectado. Por lo que todos los sistemas tienen que tener una forma de poder contactar con ellos, para mandarle ordenes, o por lo menos conocer cuál es su estado. Un ejemplo de lo que estoy hablando:
En una ciudad del futuro (y cogiendo el presente proyecto), cada hogar puede tener su propio sistema de calefacción, y cada persona puede controlar el sistema de calefacción de su casa desde su *smartphone*, *tablet* u ordenador. Pero además, en la ciudad del futuro puede haber un sistema que supervise todos los sistemas de calefacción, y si hace falta apagarlos (porque va a hacer calor y no serán necesarios por ejemplo), el sistema que supervisa puede apagarlos.

Aquí es donde surge la idea, (y necesidad) por parte de mi tutor del presente proyecto: hacer un sistema que proporcione una interfaz para interactuar con el sistema de calefacción de su casa, siguiendo la idea de "el Internet de las cosas". Y este es el segundo propósito del proyecto, demostrar que se puede hacer un sistema basado en "el Internet de las cosas", demostrando que no es tan complicado, y que el futuro no está tan lejos.

1.2 Contexto actual - KNX

Actualmente podemos decir que hay dos vertientes en cuanto a la domótica. La primera la hemos nombrado en el apartado anterior, empresas que venden sus productos con sus propios controladores y que no tienen capacidad de interconectarse entre sí o con otros sistemas. En la otra vertiente encontramos es KNX.

KNX es un estándar (ISO/IEC 14543) de protocolo de comunicaciones de red, basado en OSI, para edificios inteligentes. Su principal propósito es establecer un protocolo de comunicación para todos los elementos, independientemente del fabricante. De esta manera se hace posible la comunicación entre todos los elementos, que es la base de la domótica.

"KNX consta de básicamente 4 grupos de elementos:

ACTUADORES: Los actuadores son los elementos del sistema que se conectan físicamente sobre los elementos a controlar en el edificio, por ejemplo las luces, electroválvulas, motores, contactos secos, etc. y hacen la traducción de las instrucciones que viajan del mundo KNX al mundo físico conmutado, regulando o accionando los dispositivos que son controlados.

SENSORES: Los sensores son los elementos del sistema que recogen datos o interpretan órdenes del usuario, por ejemplo pulsador, botonera, detector de movimiento, termostato, anemómetro, sensor crepuscular; muchos sensores incorporan visualizadores o pantallas donde se controla y monitoriza el sistema, como las botoneras o pantallas táctiles.

PASARELAS: Las pasarelas (*gateways* o *routers*) enlazan otros sistemas con otros protocolos de comunicación con KNX, por ejemplo de DALI, BACnet, LONWORKS, RS485, IP, RS232, X10 etc. a KNX. Estos equipos permiten interaccionar con proyectores, otros sistemas inteligentes o incluso comunicarse en remoto con el sistema.

ACOPLADORES: Estos elementos realizan una separación física dentro del bus consiguiendo agrupar los dispositivos en un segmento de características determinadas para la cantidad de equipos, ubicaciones físicas o funciones determinadas y conectarlo con otro segmento para una mayor eficacia en el envío de datagramas a través del bus, alcanzar mayores distancias (repetidores), además de darle un direccionamiento físico muy entendible utilizando la división de Áreas, grupos y líneas.

SOFTWARE Distinguiremos el software en 2 tipos:

a) Software de gestión: Permite configurar los dispositivos y ponerlos en marcha. Esta herramienta es la única forma de configurar los dispositivos KNX y es creada, suministrada y regulada únicamente por la KNX Association.

b) Software de control: Es el programa que da acceso al sistema tanto para visualizar su estado como para controlarlo. Por otra parte también controla el correcto funcionamiento de toda la instalación y sus elementos."

Fuente: https://es.wikipedia.org/wiki/KNX#Presente_y_Futuro

1.3 Objetivos

En la parte final del apartado de motivación ya hemos explicado los principales objetivos del presente proyecto. Pero a continuación vamos a hablar con más detalle de ellos.

El principal objetivo es demostrar que la domótica es posible, y que no está tan lejos como puede parecer. Es decir, demostrar que no es ni muy caro ni muy complicado desarrollar un sistema que controle un elemento físico de un hogar, y que a su vez se pueda acceder a él remotamente desde cualquier dispositivo.

Cogiendo el ejemplo nombrado en el aparto motivación, del sistema de calefacción y un sistema que supervise todos los sistemas del pueblo o ciudad. Esto puede parecer hoy en día para muchas personas algo muy complicado y lejano en el tiempo, es verdad que no es sencillo, pero no está tan alejado en el tiempo como parece.

Las razones que tenemos para defender esta predicción son que con este proyecto estamos desarrollando una de las dos partes del escenario. Falta otro proyecto para esa segunda parte que supervise, pero podemos ver que no es tan complicado diseñar ese escenario. Por supuesto hay más temas a tratar, como pueda ser la seguridad, para evitar que no todos puedan acceder al sistema. Pero todos estos temas hoy en día se pueden solucionar.

El segundo objetivo es, ya que actualmente no disponemos fácilmente de estos sistemas, diseñarnos un sistema para nuestra propia casa con el que podamos controlar el sistema de calefacción remotamente, y incorporarlo a todo el sistema que ya hay montado en la casa.

1.4 Problemática

Una vez tuvimos claro que queríamos hacer este proyecto, el primer paso lógico es empezar a buscar referencias de otros proyectos parecidos, para inspirarse en ellos, coger ideas, cosas que se pueden mejorar, soluciones que han empleado que pueden ser útiles... El problema es que este es un campo bastante moderno, y en el que aún no se han realizado demasiados proyectos. Además, la mayoría de proyectos son de empresas privadas que no publican sus avances. De forma que no pudimos encontrar ayuda en este sentido.

Otro problema al que nos hemos enfrentado es que no teníamos experiencia previa en este campo, ni en el uso de algunas de las tecnologías empleadas. Por tanto hemos empleado bastante tiempo en aprender el funcionamiento, restándole tiempo a la última parte, quedándonos únicamente en la implementación de un prototipo.

1.5 Metodología

El plan de trabajo que seguiremos irá desde una primera fase de análisis, pasando por la fase de diseño, y terminando en la fase de implementación y desarrollo de un prototipo.

En la fase de análisis intentaremos buscar referencias de otros proyectos parecidos. También estudiaremos el funcionamiento de toda la instalación, este paso es muy importante, ya que para desarrollar un sistema que la gobierne, primero tenemos que conocer los elementos a gobernar. Por último analizaremos y diseñaremos los bocetos de la interfaz de usuario.

Una vez terminemos con el análisis, el siguiente paso es empezar con el diseño. Empezaremos con el diseño de lo que va a ser nuestro sistema que vamos a desarrollar, utilizando un diagrama de clases para la parte central del sistema. También diseñaremos una API[1] de acciones que son necesarias en el sistema de calefacción físico.

Por último, una vez hayamos tenido todo diseñado y sabemos como queremos que sea el sistema a desarrollar, empezaremos a implementar todas las partes. Empezando desde abajo, la parte más física, y finalizando en la interfaz de usuario.

1: Interfaz de programación de aplicaciones, abreviada como *API*, es el conjunto de subrutinas, funciones y procedimientos (métodos en programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción.

Vamos a describir todas las tareas que vamos a realizar.

Tarea 1: Caso de Estudio

Tarea 1.1: Estudio del Caso de Estudio
Duración: 3 días
Input: Información por parte del tutor y de páginas web
Output: Documento explicativo del funcionamiento

Tarea 1.2: Análisis de Elementos Físicos
Duración: 1 día
Input: Documento explicativo del funcionamiento (Tarea 1.1)
Output: Especificación de los elementos físicos significativos

Tarea 1.3: Interacción del Usuario
Duración: 1 día
Input: Documento explicativo del funcionamiento (Tarea 1.1)
Output: Especificación de las acciones del usuario

Tarea 2: Análisis del Sistema

Tarea 2.1: Especificación del dominio
Duración: 1 día
Input: Documento explicativo del funcionamiento (Tarea 1.1)
Output: Diagrama de Clases

Tarea 2.2: Interfaz de Usuario
Duración: 1 día
Input: Especificación de las acciones del usuario(Tarea 1.3)
Output: Boceto interfaz de usuario

Tarea 2.3: Interfaz de Comunicación con el Sistema Físico
Duración: 1 día
Input: Documento explicativo del funcionamiento (Tarea 1.1)
Output: Documento explicativo de acciones sobre sistema físico

Tarea 3: Diseño del Sistema

Tarea 3.1: Rest
Duración: 2 días
Input: Diagrama de Clases (Tarea 2.1), Boceto interfaz de usuario (Tarea 2.2)
Output: Documento recursos necesarios para Rest

Tarea 3.2: Api Arduino, Comunicación con Sistema Físico
Duración: 2 días
Input: Documento explicativo de acciones sobre sistema físico (Tarea 2.3)
Output: Documento con Api para Arduino

Tarea 4: Implementación

Tarea 4.1: Conocer Funcionamiento Arduino
Duración: 4 días
Input:-
OutPut:-

Tarea 4.1: Implementación Capa Física
Duración: 5 días
Input: Documento con Api para Arduino (Tarea 3.2)
Output: Capa Física

Tarea 4.2 Implementación Capa Lógica de Negocio
Duración: 20 días
Input: Diagrama de Clases (Tarea 2.1)
Output: Capa Lógica

Tarea 4.3 Implementación Interfaz de Usuario
Duración: 7 días
Input: Documento recursos necesarios para Rest (Tarea 3.1),
Boceto interfaz de usuario (Tarea 2.2)
Output: Capa Interfaz de Usuario

Aquí se presentan las tareas principales que desarrollaremos durante el proyecto. Sin embargo hay tareas más explícitas que en el momento de definir el plan de trabajo no podemos prever, y que nos encontraremos en el momento que empecemos la programación del sistema.

2 Tecnologías

A continuación vamos a nombrar todas las tecnologías que hemos empleado en el desarrollo de nuestro proyecto:

La primera tecnología que hemos utilizado ha sido el simulador Proteus. Es un simulador de electrónica, que descargando unas librerías especializadas, permite simular circuitos con la placa Arduino.

Arduino es una plataforma de hardware libre basada en una placa con un microcontrolador y un entorno de desarrollo. La placa Arduino permite conectar y controlar múltiples sensores y elementos electrónicos. El entorno de desarrollo es un editor de texto que permite programar la placa. Esta tecnología la hemos empleado para desarrollar la parte física del sistema.

Para el sistema central que controlará la lógica de negocio, hemos empleado el lenguaje de programación Java. Es un lenguaje orientado a objetos que permite una fácil modelación de la realidad, ayudando a la programación.

REST es un patrón de diseño destinado a la parte del servidor que indica como se debe acceder a los recursos un cliente que los necesite. El objetivo es crear soluciones sencillas usando como protocolo de transferencia HTTP. Tiene como base la orientación a recursos, a los que se accede mediante *URI's*[1], y sobre los que se realizan operaciones permitidas por HTTP: *GET, POST, DELETE, HEAD, OPTIONS.*

AJAX, acrónimo de *Asynchronous JavaScript And XML*, es una técnica de desarrollo web que permite hacer peticiones a servicios remotos. Normalmente forma parte de la parte cliente, y manda operaciones HTTP a un servidor. Gracias a esta técnica hemos podido actualizar el estado del sistema en la interfaz, o emplear la interfaz para controlar el sistema.

La interfaz de usuario ha sido diseñada usando HTML, el lenguaje que sirve para diseñar páginas web. Y para dotarle de dinamismo a la interfaz hemos usado JavaScript, un lenguaje de programación que permite modificar la interfaz diseñada en HTML conforme el usuario realiza acciones, o conforme recibe datos externos (de un servidor, por ejemplo).

1: Es una cadena de caracteres que identifica los recursos de una red de forma unívoca.

2.1 Proteus

Como no teníamos conocimientos previos sobre la plataforma Arduino, empleamos un simulador para entender como funcionaba antes de empezar con la placa real.

Usamos el simulador Proteus, más concretamente su herramienta Isis. Proteus es un simulador de electrónica principalmente, pero también proporciona librerías para simular placas Arduino.

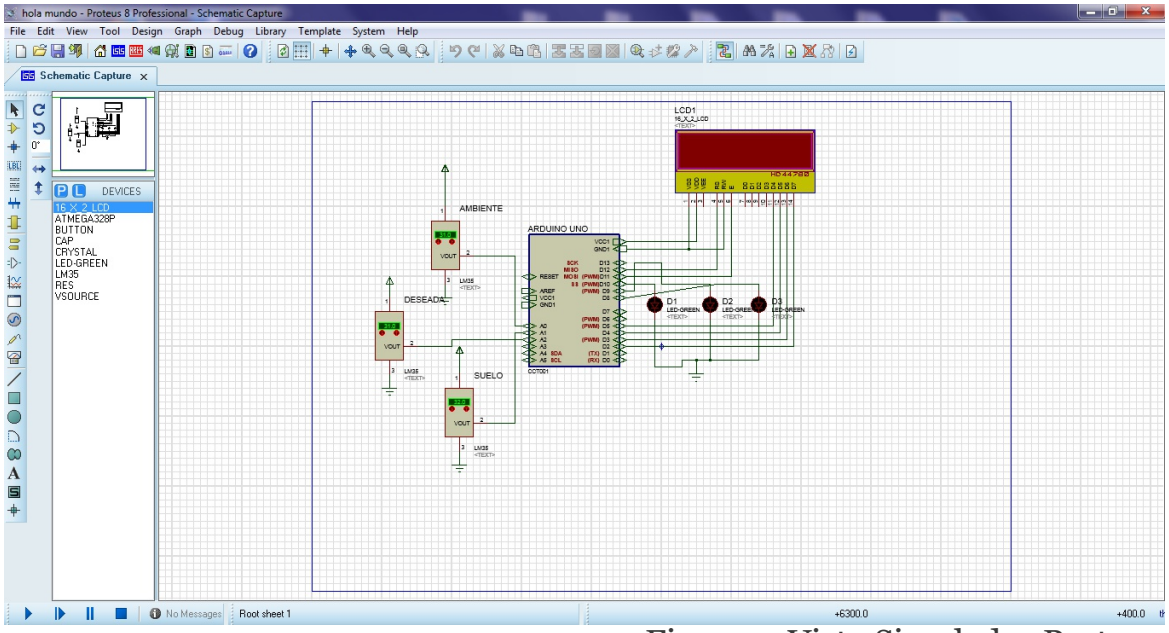

Figura 1: Vista Simulador Proteus

En la imagen podemos ver la primera simulación que hicimos del sistema en el programa Proteus. Representa únicamente una habitación, con dos sensores de temperatura a la izquierda, que muestran la temperatura real de la habitación y la temperatura del agua que pasa por el suelo radiante de la misma. Además pusimos un tercer sensor, para insertar la temperatura que deseamos, (en la realidad no existe tal sensor, se inserta la temperatura deseada mediante la interfaz web).

A la derecha podemos observar tres LED's que representan tres resistencias que calientan el agua, cuando se enciende un LED simula que se enciende una resistencia. Arriba se encuentra una pantalla LCD para hacer comprobaciones de que todo funcionaba bien, mostrando las temperaturas leídas.

2.2 Arduino

Arduino es una plataforma de hardware libre sobre las que múltiples empresas fabrican los mismos tipos de placas. En nuestro caso hemos utilizado la placa Funduino Uno, que es una de las copias de la placa Arduino Uno que existen en el mercado. Consta de 6 entradas análogicas donde normalmente se conectan los sensores (para recibir información), y 15 entradas digitales en las que se conectan los actuadores (para activar o desactivarlos). Por ejemplo, cogiendo la simulación del punto anterior, los sensores de temperatura se insertan en las entradas analógicas. Por otra parte los LED's se conectan a las entradas digitales, para activarlos o no.

También hay clavijas para obtener corriente continua a varios voltajes, entre ellos 5V o 3V, que se utiliza para dar electricidad a algunos tipos de sensores que la necesitan. Por supuesto también hay conectores que ofrecen 0V, o también llamado "masa".

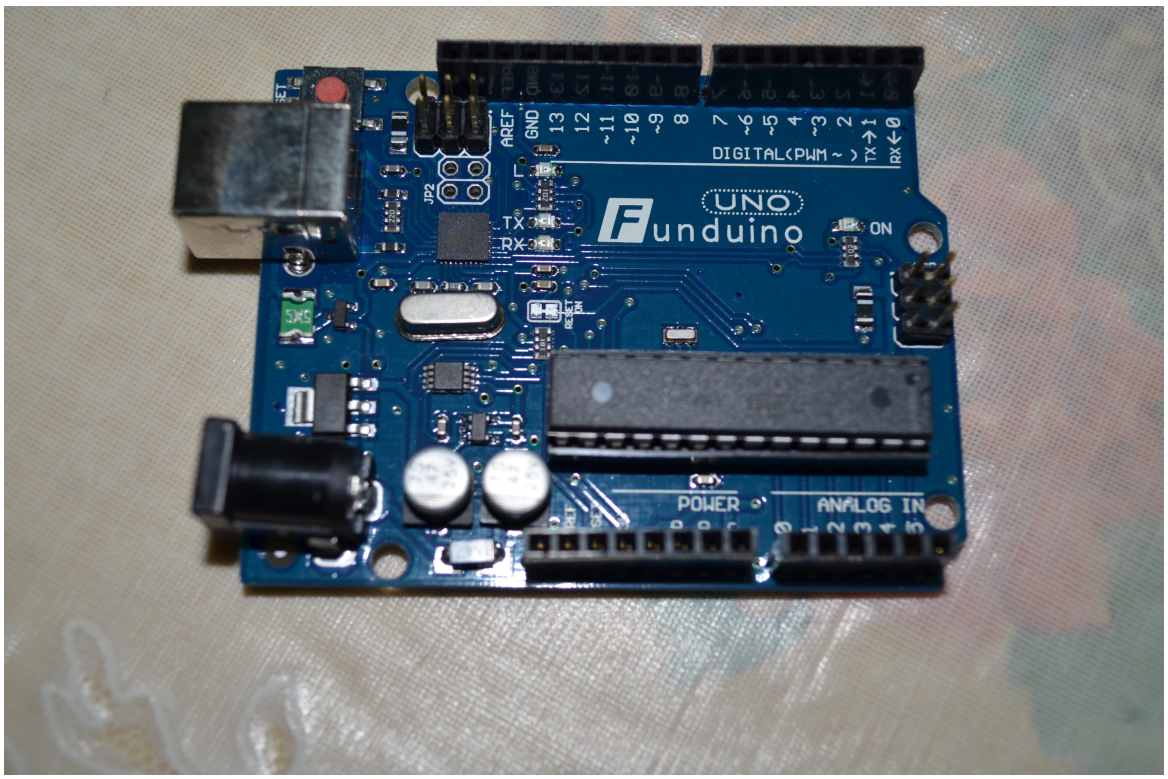

Figura 2: Placa Arduino

2.3 Arduino, Entorno de Desarrollo

La placa Arduino se programa utilizando el lenguaje C adaptado a este especial hardware. Básicamente un programa para Arduino tiene dos partes o métodos principales. Un primer método llamado *"setup"*, donde se preparan todas las variables, entradas y salidas necesarias, etc. Y un segundo método llamado *"loop"*, que es básicamente un bucle infinito donde se ejecuta continuamente una serie de operaciones, llamadas a métodos. Así por ejemplo, podemos tener un método llamado "getTemp()", que lee la temperatura de un sensor "x". En el *"loop"*, llamaremos a la subrutina "getTemp()", por tanto siempre se estará leyendo la temperatura de dicha subrutina.

Para darle descanso a Arduino, y que no esté continuamente realizando mediciones o ejecutando métodos, se inserta un "delay(x)" de "x" mili-segundos dentro del *"loop"*.

Por otra parte, la plataforma Arduino tiene un software propio que permite escribir estos programas. A continuación mostramos un ejemplo de programa realizado en Arduino.

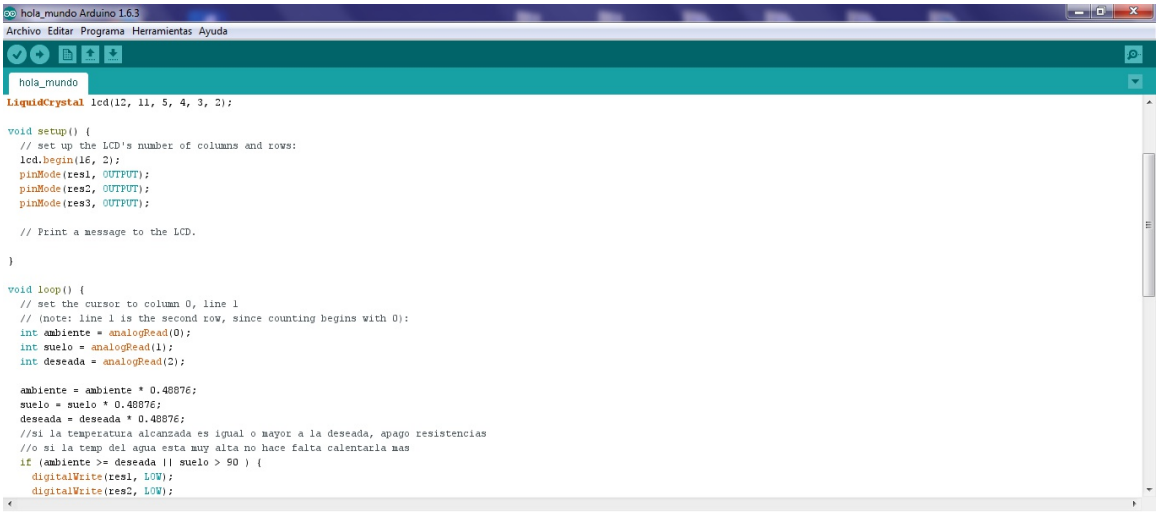

Figura 3: Entorno Desarrollo Arduino

El mismo programa nos permite compilar y cargar el programa a la placa Arduino. Para cargar el programa en el simulador hay que hacer algo distinto:
Se escribe el programa igual, y al compilarlo, en la parte baja del software muestra una serie de archivos temporales que se crean al compilar, nos fijamos en el último archivo, un ".hex", vamos a la carpeta donde se encuentra y cargamos ese archivo en el simulador.

2.4 Java

Como ya hemos dicho Java es un lenguaje de programación que sigue el paradigma de programación orientada a objetos, lo que significa que se programa utilizando objetos y realizando operaciones sobre ellos, incluso entre ellos mismos. Un objeto puede verse como un paquete que contiene el “comportamiento” (el código) y el “estado” (datos). La programación orientada a objetos hace que el software que se desarrolla se parezca más a la realidad.

El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software.

La segunda característica importante es la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "write once, run anywhere". Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como “bytecode” (específicamente Java bytecode)—instrucciones máquina simplificadas específicas de la plataforma Java. Posteriormente este “bytecode” se ejecutará sobre la máquina virtual (JVM), que si que trabajará sobre el hardware específico de cada ordenador, traduciendo las instrucciones del “bytecode” a las instrucciones del hardware.

Fuente https://es.wikipedia.org/wiki/Java_(lenguaje_de_programación)

2.5 *REST*

Un servicio *R*EST se resume en el siguiente esquema:

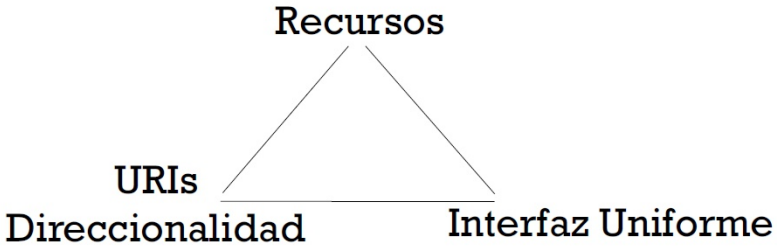

Figura 4: Esquema de Funcionamiento *Rest*

Los recursos son los elementos de un sistema que es interesante ,por alguna razón, hacerlos accesibles desde fuera del sistema. Por cada uno de los elementos se crea un recurso. Cada recurso debe tener un identificador, y unas operaciones que se podrán hacer sobre él.

Los identificadores los llamaremos *Uri*'s. Cada recurso debe tener una *Uri* que emplearemos para acceder a él, y una *Uri* sólo puede apuntar a un recurso. Deben de ser descriptivas, de forma que sean fácilmente recordables y a la vez indiquen claramente a que recurso vamos a acceder. Un ejemplo puede ser "/App/SistemaProduccion/Resistencia".

La interfaz son las operaciones definidas para cada recurso, un recurso puede tener una, varias, o ninguna operación. En cualquier caso, las operaciones permitidas son las operaciones que proporciona el protocolo HTTP:

-Get: con ella se recupera una representación de un recurso.

-Post: sirve para crear un nuevo recurso.

-Put: muy parecida a Post, crea un nuevo recurso, pero en el caso de que ya exista, lo actualiza.

-Delete: se borra un recurso.

-Head: se puede entender como un “resumen” de Get, se recibe la cabecera del recurso.

-Options: devuelve las operaciones permitidas de un recurso.

2.6 Ajax

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página, aunque existe la posibilidad de configurar las peticiones como síncronas de tal forma que la interactividad de la página se detiene hasta la espera de la respuesta por parte del servidor.

JavaScript es el lenguaje interpretado (*scripting language*) en el que normalmente se efectúan las funciones de llamada de Ajax, mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales.

Ajax es una combinación de cuatro tecnologías ya existentes:

-XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño que acompaña a la información.

-*Document Object Model* (DOM) accedido con un lenguaje de *scripting* por parte del usuario, especialmente implementaciones ECMAScript como JavaScript y JScript, para mostrar e interactuar dinámicamente con la información presentada.

-El objeto XMLHttpRequest para intercambiar datos de forma asíncrona con el servidor web. En algunos *frameworks* y en algunas situaciones concretas, se usa un objeto *iframe* en lugar del XMLHttpRequest para realizar dichos intercambios. PHP es un lenguaje de programación de uso general de *script* del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico también utilizado en el método Ajax.

-XML es el formato usado generalmente para la transferencia de datos solicitados al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSON y hasta EBML.

Fuente: https://es.wikipedia.org/wiki/AJAX

2.7 HTML

HTML, siglas de *HyperText Markup Language* («lenguaje de marcas de hipertexto»), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, entre otros.

El HTML se escribe en forma de «etiquetas», rodeadas por corchetes angulares (<,>). El HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir o hacer referencia a un tipo de programa llamado script, el cual puede afectar el comportamiento del documento HTML.

Un elemento generalmente tiene una etiqueta de inicio (por ejemplo, <nombre-de-elemento>) y una etiqueta de cierre (por ejemplo, </nombre-de-elemento>). Los atributos del elemento están contenidos en la etiqueta de inicio y el contenido está ubicado entre las dos etiquetas (por ejemplo, <nombre-de-elemento atributo="valor"> Contenido </nombre-de-elemento>). Algunos elementos, tales como
, no tienen contenido ni llevan una etiqueta de cierre.

Fuente: https://es.wikipedia.org/wiki/HTML

2.8 JavaScript

JavaScript (abreviado comúnmente "JS") es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Al contrario que Java, que lo hemos explicado en el punto 2.4, JavaScript es un lenguaje interpretado. Es decir, no se compila, si no que directamente se interpreta en el momento de la ejecución.

Los usos más significativos son:

-Cargar nuevo contenido para la página o enviar datos al servidor a través de AJAX sin necesidad de recargar la página (por ejemplo, una red social puede permitir al usuario enviar actualizaciones de estado sin salir de la página).

-Animación de los elementos de página, hacerlos desaparecer, cambiar su tamaño, moverlos, etc.

-Contenido interactivo, por ejemplo, juegos y reproducción de audio y vídeo.

-Validación de los valores de entrada de un formulario web para asegurarse de que son aceptables antes de ser enviado al servidor.

-Transmisión de información sobre los hábitos de lectura de los usuarios y las actividades de navegación a varios sitios web. Las páginas Web con frecuencia lo hacen para hacer análisis web, seguimiento de anuncios, la personalización o para otros fines.

Una de las características más importantes en cuanto a seguridad es el hecho de que JavaScript está limitada por la política del mismo origen: los scripts de un sitio web no tienen acceso a la información enviada a otro sitio web (de otro dominio) como pudiera ser nombres de usuario, contraseñas o cookies. Este “problema” nos lo encontraremos más adelante en el proyecto, cuando desarrollemos la interfaz, y explicaremos como solucionarlo.

Fuente: https://es.wikipedia.org/wiki/JavaScript

3 Caso de Estudio: Sistema de Calefacción

El proyecto se ha basado en un sistema real de calefacción por suelo radiante instalado en una casa y que actualmente está funcionando. El sistema contiene dos partes diferenciadas: una de producción y otra de consumo. El objetivo del sistema de producción es calentar agua. El de consumo, repartir ese calor vía suelo radiante. A continuación vamos a explicar detalladamente su funcionamiento.

3.1 Sistema de Calefacción, Producción

El sistema de producción es responsable de calentar el agua para su consumo. Para ello, consta de dos circuitos de agua: el primario (la propia producción) y el secundario (que se irá para su consumo a las diferentes zonas de la vivienda). Los dos circuitos son cerrados e independientes entre sí. Cada uno de ellos tiene una bomba que hace circular el agua por los circuitos (el circuito de consumo tiene dos bombas debido a que es un circuito más extenso y necesita de la ayuda de dos bombas). Junto a las bombas hay un interruptor de flujo (un sensor/interruptor) que se activa cuando hay circulación de agua. Se usa para que el sistema de producción no esté activo en caso que no haya circulación de agua, ya que al estar detenida, podría aumentar mucho la temperatura en una zona concreta y dañar elementos (sondas, resistencias, etc.), además de provocar un consumo energético innecesario.

El elemento central del sistema de producción es el intercambiador. Este es responsable de poner en contacto el agua caliente de producción con el agua fría que llega del circuito secundario (tras el consumo), elevando su temperatura (y igualándose a la de producción, y bajando consecuentemente ésta) para que pueda volver a enviarse para su consumo. El intercambiador es muy eficiente en la transmisión de calor, mucho más que un depósito acumulador. Para conseguir que en el circuito secundario hayan 30ºC (para repartir ese calor por la vivienda y obtener entre 21-23ºC), usando un acumulador hace falta calentar el agua del circuito primario a unos 55-60ºC (desfase térmico de 25-30ºC), mientras que con el intercambiador sólo es necesario calentar el agua a 33ºC (desfase térmico de 3ºC).

Al intercambiador le llega el agua ya calentada (ida) del circuito primario de calefacción. En la instalación hay dos maneras de calentar el agua: uno ‘forzado’, basado en resistencias (consumen energía) y otro ‘natural’ que calienta el agua con placas solares. A cada uno de ellos le llega una ida del primario (con el agua a baja temperatura que sale del intercambiador) y le devuelven el agua caliente después de haber influído sobre ella. Los dos mecanismos de producción se conectan vía una válvula de 3 vías (electro-mecánica) que selecciona uno de los circuitos (no pueden estar los dos conectados en paralelo/a la vez).

3.2 Sistema de Calefacción, Consumo

El sistema de consumo se constituye de un circuito cerrado donde se reparte el agua por medio de colectores (puede haber uno o varios) a las zonas de consumo (habitaciones, estancias, baños, etc.). Para cada zona hay una electroválvula gradual (permite apertura total/parcial), y básicamente un sensor de temperatura ambiente de la zona. Con la electroválvula gradual se puede ajustar la cantidad de agua que se deja pasar a la zona para conseguir más o menos aumento (o disminución de temperatura): cuanta más agua se deje fluir, más temperatura se 'quedará' en esta zona.

Este sistema de reparto de agua por colectores consigue montar un circuito 'en paralelo', de manera que a todas las estancias les llega 'en teoría' el agua a la misma temperatura de entrada. Decimos en teoría porque en la práctica, el agua va perdiendo calor a medida que va avanzando metros, y no todos los colectores y circuitos de agua de las zonas tienen la misma longitud, por lo que el agua no puede llegar 'eficazmente' a todos los sitios con la misma temperatura. De todas maneras, tal y como está diseñado el sistema, esto no es un problema, ya que si se quiere, por ejemplo, aumentar la temperatura de una zona, podemos hacer dos cosas: abrir más la electroválvula de la zona (consiguiendo que entre más temperatura en la zona) sin aumentar la producción de calor, o aumentar la temperatura de envío del agua activando más resistencias (si es que estamos con el circuito 'forzado' actuando).

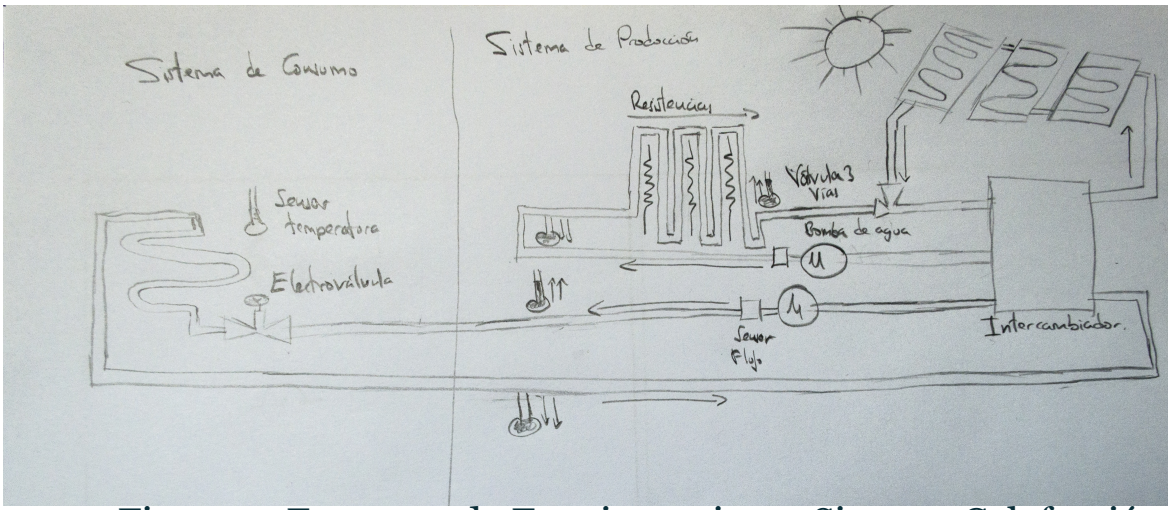

Figura 5: Esquema de Funcionamiento Sistema Calefacción

3.3 Elementos Físicos

Ya hemos explicado el funcionamiento de todos los componentes del sistema de calefacción, pero para nuestro sistema no los vamos a utilizar todos. Solo hemos utilizado en nuestro diseño los que hemos considerado más importantes para controlar el correcto funcionamiento de la calefacción.

Sensor de temperatura: Es una de las partes más importantes ya que nos permite conocer las temperaturas. Hay sensores de temperatura en cada habitación para poder saber la temperatura actual, y así poder saber la diferencia térmica que hay con respecto a la temperatura de confort, y realizar las acciones pertinentes sobre los componentes. La única acción que podemos hacer es la de obtener la temperatura.

Resistencias: Es otra de los principales componentes del sistema de calefacción. Son las encargadas de calentar el agua que circula por la casa. Las acciones que podemos realizar son encender y apagar.

Electroválvulas: Las encontramos en el circuito de agua que circula por la casa, a la entrada de cada estancia. Permite el paso del agua y las podemos cerrar o abrir parcial o totalmente para permitir más paso de agua o menos, y así calentar más o menos la estancia.

Bombas: Se encuentran en el sistema de producción, y se encargan de mover el agua por los circuitos tanto de producción como de consumo. Las podemos encender o apagar, para mover el agua o no respectivamente.

Sensor de flujo: Es el elemento que está junto a las bombas, sirve para controlar que realmente hay flujo si encendemos las bombas, ya que puede haber una obstrucción en el circuito, o las bombas pueden no funcionar correctamente. Es un sensor que nos informa del estado del circuito, por lo que no podemos realizar ninguna acción sobre, únicamente obtener el estado.

Válvula de 3 vías: Es una válvula que tiene dos posibles entradas y una única salida. Cuando una entrada está abierta la otra entrada está cerrada, únicamente puede haber una entrada abierta. La salida está siempre abierta. Se usa para elegir la fuente de producción de calor, resistencias o placas solares.

3.4 Interacción de Usuario

Una vez hemos definido que elementos del sistema de calefacción nos interesa controlar, tenemos que decidir que interacción debemos ofrecer al usuario, es decir, que acciones le permitimos tomar y cuales no. Por supuesto, no vamos a darle todo el control del sistema al usuario, ya que podría tocar alguna cosa que no debiera ya sea por accidente o desconocimiento, provocando un fallo o aún peor, una avería. Pero tampoco vamos a negarle todo el control, ya que habrá órdenes que si podrá dar, así como también se necesita que el usuario inserte información.

Por tanto la funcionalidad que hemos decidido que nuestro sistema le debe ofrecer al usuario es la siguiente:

-El usuario puede encender o apagar el sistema de calefacción.

-El usuario puede encender o apagar la calefacción en cada habitación independientemente del resto de habitaciones.

-El usuario inserta la temperatura de confort para cada habitación independiente del resto, o la misma temperatura de confort para todas las estancias de la casa.

-Se le ofrece información al usuario del estado de las resistencias, para saber cuántas hay encendidas en cada momento, y así conocer el consumo aproximado de luz que se está produciendo en cada instante.

-También se le ofrece información de si hay flujo o no en los circuitos, así podrá saber si hay un problema y algo no funciona correctamente para que lo pueda solucionar.

-En cada estancia se le indica la temperatura actual y la temperatura de confort que se ha seleccionado; también la diferencia térmica entre ambas.

Para poder ofrecer lo descrito, hemos desarrollado una interfaz de usuario. En los siguientes puntos 4, 5 y 6, en los apartados "Interfaz de Usuario" la iremos desarrollando e implementando.

4 Análisis del sistema

Una vez hemos acabado de analizar nuestro caso de estudio y tenemos claro su funcionamiento, tenemos que empezar con el sistema que vamos a diseñar. La primera parte es el análisis de como tiene que ser la parte del sistema que llevará la lógica de negocio. La segunda como tiene que ser la interfaz del sistema, qué tiene que ofrecerle al usuario y cuál es el principal objetivo. Y por último la interfaz de comunicación con el sistema físico, qué acciones debemos poder realizar.

4.1 Diagrama de Clases

La primera parte del análisis es analizar como va a ser la lógica de negocio. Se llama lógica de negocio a la parte del sistema que se encarga de controlar el funcionamiento del resto del sistema. Para analizar todos elementos que necesitamos se utiliza un diagrama de clases. En él se representa todas las partes de la lógica de negocio, las relaciones entre las partes, etc.

En el centro, en la parte de arriba observamos la clase SistemaCalefacción, esta clase no tiene ninguna función, únicamente representa el conjunto del sistema de calefacción. A los lados tenemos el sistema de producción y el sistema de consumo, que a su vez se componen de resistencias el sistema de producción y de habitaciones el de consumo. Por último tenemos los elementos físicos, representan todos los elementos del sistema que tenemos que controlar, las resistencias, electroválvulas, sensores de flujo, de temperatura y por último los motores o bombas de agua.

Para cada clase, se tienen unos atributos que permiten observar su estado o la información de interés que proporcionan. Además, en la parte de abajo, se encuentran los métodos o acciones que podemos realizar sobre cada clase.

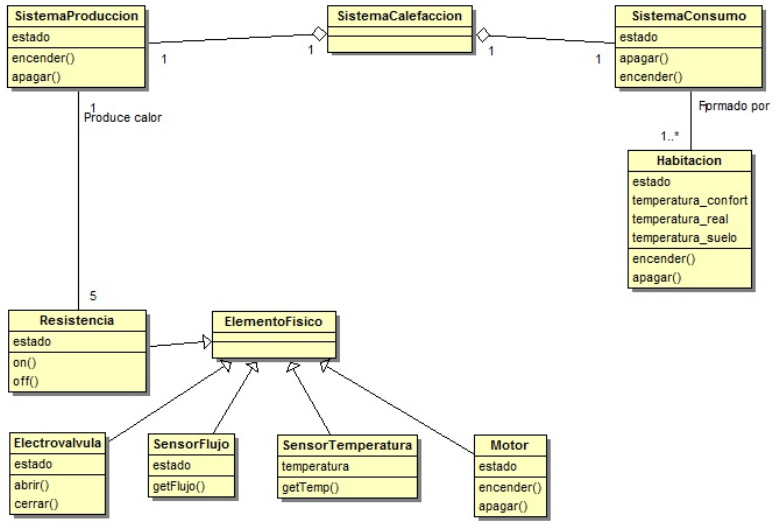

Figura 6: Diagrama de Clases Sistema de Calefacción (Análisis)

4.2 Interfaz de Usuario

La interfaz queríamos que tuviese un diseño muy simple, para que cualquier usuario sean cuales sean sus conocimientos en nuevas tecnologías pueda usar nuestro sistema. Por este motivo hemos seguido un diseño basado en iconos sencillos, los cuales son fácilmente comprensibles. También hemos situado los iconos y elementos que nos permiten realizar las acciones sobre el sistema en lugares fácilmente visibles, de forma que al usuario le resulte simple usar la interfaz.

La primera vista que tenemos es la del sistema de calefacción. En ella se podrá encender o apagar todo el sistema, acceder a la siguiente vista (la de la habitación). Y también se visualizará el estado del sistema de calefacción mediante dos iconos, que se mostrarán encima de una foto de la instalación. Es decir, se utilizará de fondo una imágen real de la instalación, y sobre esta foto se sobrepondrá los iconos.

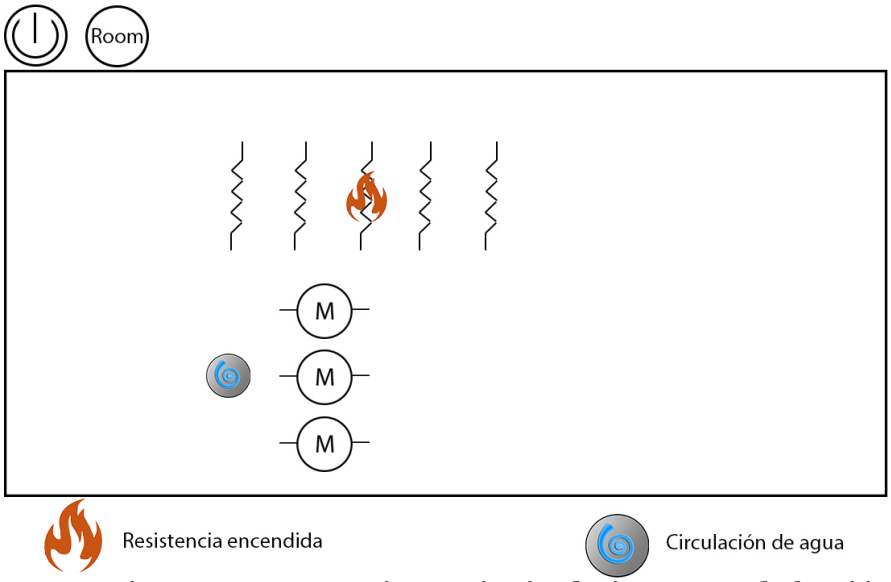

Figura 7: Boceto vista principal Sistema Calefacción

La segunda vista que tenemos es de la habitación. En ella podemos conectar o desconectar la calefacción en esa habitación, seleccionar la temperatura que deseamos que haya, o volver a la vista anterior. En la parte superior derecha se muestra la temperatura de confort de la habitación y la temperatura real que hay.

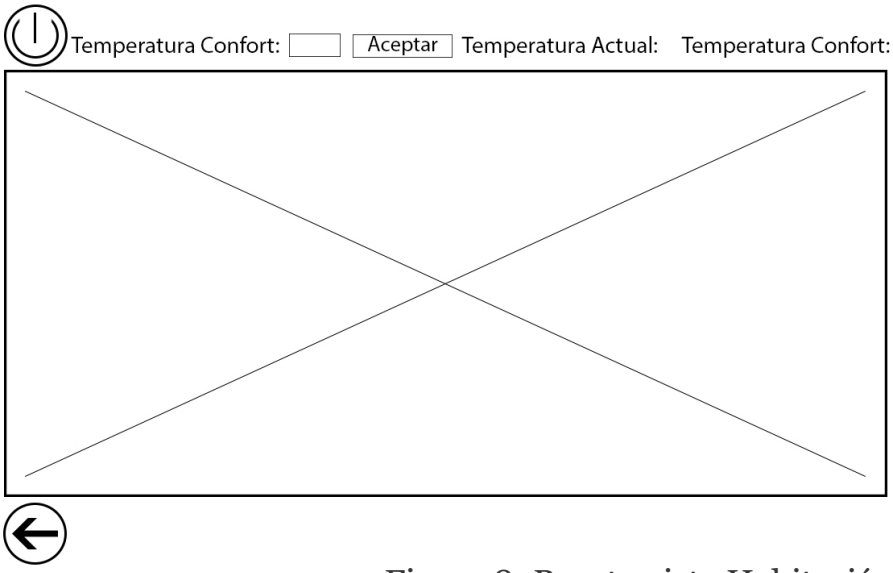

Figura 8: Boceto vista Habitación

Para cada habitación o estancia de la casa tendremos una vista distinta, y para cada vista habrá un plano distinto de la propia habitación. Así, para el comedor encontraremos un plano del comedor, para una habitación el plano de esa habitación, etc.

4.3 Interfaz de Comunicación con Sistema Físico

El sistema se ha dividido en tres capas. En la parte física encontramos la tecnología Arduino para controlar el sistema de calefacción por suelo radiante. Para controlar dicho sistema hay que definir todas las acciones que podamos realizar sobre él. A continuación se muestran todas las acciones que hemos definido:

-Apagar cada una de las resistencias por separado, (apagar resistencia nº1, nº2...).

-Encender cada una de las resistencias por separado, (encender resistencia nº1, nº2...).

-Apagar cada bomba de agua por separado, (apagar bomba nº1, nº2, nº3).

-Encender cada bomba de agua por separado, (encender bomba nº1, nº2, nº3).

-Obtener las temperaturas de cada una de las estancias.

-Obtener la temperatura de salida del sistema de producción.

-Abrir la electroválvula de cada una de las estancias.

-Cerrar la electroválvula de cada una de las estancias.

-Cambiar entrada válvula de 3 vías.

Con estas acciones podemos controlar el sistema de calefacción, en el punto 5 Diseño: Interfaz de comunicaciones se verán los comandos que hemos implementado para cada una de las acciones.

5 Diseño

Ya hemos acabado con el análisis del sistema y hemos analizado los tres pilares fundamentales, por lo que ahora vamos a empezar con el diseño "refinando" todo lo analizado en el punto anterior. Es decir, acercándonos más a la implementación, añadiendo elementos y puntos a tener en cuenta cuando empecemos a desarrollar el sistema.

Primero vamos a completar el diagrama de clases que hicimos en el punto 4, añadiendo atributos y demás elementos. Más tarde hablaremos de la capa de interacción entre la interfaz y la capa de lógica de negocio. Por último definiremos los comandos de las acciones de la interfaz de comunicación con el sistema físico.

5.1 Diagrama de Clases

En el punto 4 hemos hecho el primer diagrama de clases. Un diagrama básico para empezar a analizar la lógica de negocio. Ahora tenemos que añadir unas clases más que necesitaremos para que todo el sistema funcione. Por una parte las reglas de funcionamiento, que se encargarán de controlar el funcionamiento de todo el sistema. Y por otro lado la conexión a la parte física del proyecto.

La conexión a la parte física es necesaria ya que esta parte del proyecto está desarrollada en un lenguaje, y estará situada en un ordenador. La parte física está desarrollada en otro lenguaje y se sitúa en otra plataforma. Para realizar esta conexión empleamos los *drivers*, cada elemento físico tendrá su propio *driver* que se conectará con el túnel serie.

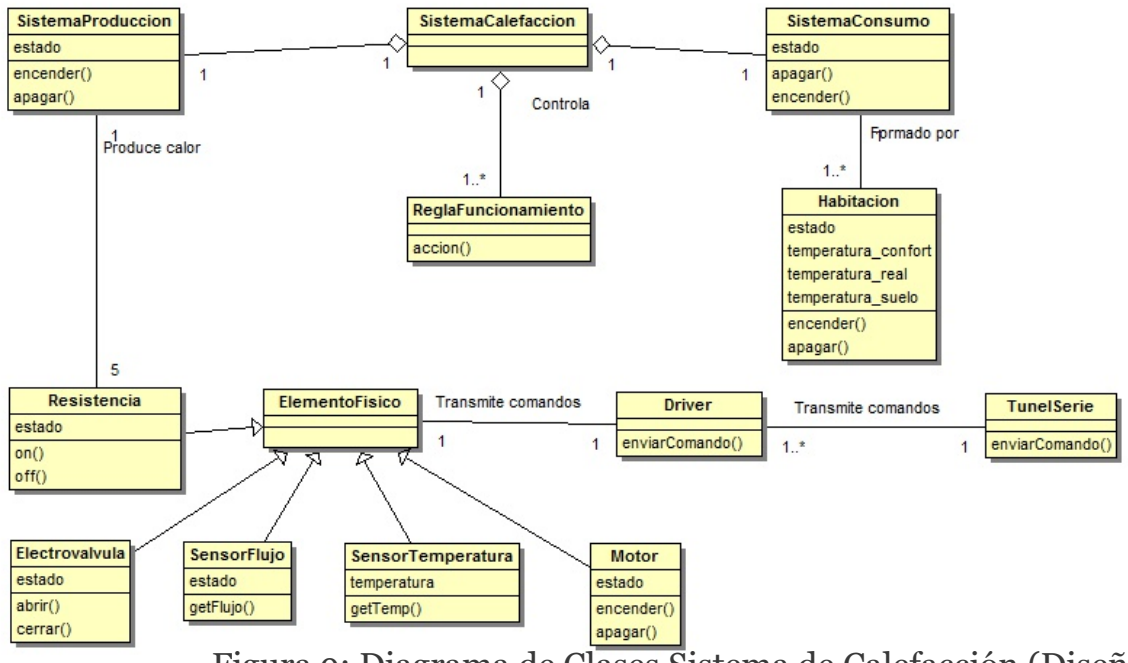

Figura 9: Diagrama de Clases Sistema de Calefacción (Diseño)

5.2 Capa de Interacción, *REST*

En el punto 4 hemos hablado de la interfaz de usuario, que permite al usuario interaccionar con el sistema. Ahora vamos a ver como "conectar" la interfaz con el resto del sistema. El problema principal es que la interfaz y "el sistema central", la parte que lleva toda la lógica de negocio estarán implementados con lenguajes totalmente diferentes, por lo que la conexión no es trivial.

En el punto 2 Tecnologías hemos hablado de *REST*, un patrón de diseño que sirve para indicar como un cliente puede acceder a los recursos de un servidor. Esto es lo que vamos a usar para resolver este problema. Vamos a identificar los recursos que puede necesitar la interfaz para implementarlos y asignarles una *Uri*. De esta forma, cuando la interfaz quiera transmitir alguna acción al sistema o necesite cierta información, accederá a un recurso. A continuación vamos a identificar que recursos necesitamos, que *Uri* le vamos a asignar a cada uno, y que acciones debemos poder realizar sobre ellos.

-Resistencias. Hay que crear un recurso para cada una de las cinco resistencias que tenemos, para que podamos conocer su estado y poder representarlo en la interfaz.
Uri : /App/Resistencias/
Operaciones: Get.

-Bombas. Tenemos tres bombas de agua, por lo que necesitamos un recurso para cada una. Las acción que podemos hacer es conocer su estado.
Uri: /App/Bombas/
Operaciones: Get.

-Flujo de agua. Hay un sensor de flujo detrás de cada bomba de agua, que nos indica si hay flujo de agua o no. Este es otro recurso del que queremos conocer su estado.
Uri: /App/Flujo/
Operaciones: Get.

-Temperaturas. Necesitamos crear un recurso por cada sensor de temperatura que tenemos. La única acción que podemos realizar es leer su valor.
Uri: /App/Temperaturas/
Operaciones: Get.

-"OnOff". Hemos creado este recurso para representar de alguna forma la posibilidad de apagar o encender el sistema de calefacción, pero realmente no corresponde a ningún elemento físico. Le podemos indicar que se encienda o se apague.
Uri: /App/OnOff/
Operaciones: Get, Put.

5.3 Interfaz de Comunicación con Sistema Físico

En el apartado 4.3 hemos identificado que acciones necesitamos hacer sobre el sistema de calefacción. Ahora vamos a definir que comandos usaremos para cada una de las acciones, de forma que mediante esos comandos se puedan comunicar la parte que lleva la lógica de negocio con la parte física de nuestro sistema que se encarga de controlar el sistema de calefacción. Es decir, la parte de lógica de negocio enviará los comandos a la parte física, y la parte física recibirá los comandos y sabrá que acción debe hacer. A esta serie de comandos la llamaremos la *API* de comunicación con la parte física.

-1ResOff, 2ResOff, 3ResOff, 4ResOff, 5ResOff => Apagar cada una de las resistencias por separado, (apagar resistencia nº1, nº2...).

-1ResOn, 2ResOn, 3ResOn, 4ResOn, 5ResOn => Encender cada una de las resistencias por separado, (encender resistencia nº1, nº2...).

El número inicial identifica a cada una de las resistencias, “Res” de resistencia (resistance en inglés) , y “On” o “Off” de encender o apagar respectivamente.

-Eng1off, Eng2off, Eng3off => Apagar cada bomba de agua por separado, (apagar bomba nº1, nº2, nº3).

-Eng1on, Eng2on, Eng3on => Encender cada bomba de agua por separado, (encender bomba nº1, nº2, nº3).

“Eng” de *engine* (motor en inglés), el número de identificación de cada bomba, y “on” o “off” para encender o apagar.

-SrgetTemp => Obtener las temperaturas.

SR es el identificador, “getTemp” significa en inglés obtener temperatura. Para cada sensor de temperatura tendremos un comando distinto cambiando el identificador.

-1ElectTrue => Abrir la electroválvula de cada una de las estancias.

-2ElectFalse => Cerrar la electroválvula de cada una de las estancias.

El número inicial es el identificador de cada electroválvula, “Elect” de electroválvula, “True” sirve para activarla y abrirla; con “False” la desactivamos y cerramos. (Hay que recordar que hemos simplificado en nuestro prototipo, y sólo la abrimos y cerramos. En el sistema de consumo las electroválvulas se pueden abrir parcialmente)

-3Vias1 => Cambiar a la entrada 1 la válvula de 3 vías.

-3Vias2 => Cambiar a la entrada 2 la válvula de 3 vías.

3Vias es el nombre de la válvula, y 1 o 2 representan las dos posiciones posibles que puede tener.

En un primer momento, definimos unos comandos más largos que explicaran mejor lo que representa cada uno. Sin embargo nos dimos cuenta, al empezar a hacer las primeras pruebas, que si los comandos eran demasiado largos y empezaban igual (por ejemplo, “EncenderMotor1”, “EncenderMotor2”), se producían errores.

La placa Arduino no los distinguía, y escogía la acción del primer comando que encontrase empezase igual (en el ejemplo que hemos puesto, aunque mandásemos la acción “EncenderMotor2”, siempre hacía la acción perteneciente al primer comando “EncenderMotor1”, suponiendo que al programar se define primero la acción “EncenderMotor1”). Parece como si solo pudiese distinguir las primeras 5 o 6 primeras letras de los comandos, por lo que tuvimos que volver diseñar los comandos para hacerlos más cortos.

También se puede observar que no siguen un patrón común los comandos, en unos “On” representa la activación mientras que en otros empleamos “True”. Debido a los errores que encontramos, decidimos hacerlos muy diferenciables para evitar cualquier tipo de problema.

Esta es la *API* que hemos definido y que usaremos en el proyecto para mandar acciones a la capa física. Sin embargo no hemos asignado aún ninguna acción a los comandos. En el punto 6, donde empezaremos con la implementación, se explicará como asignar acciones a los comandos.

6 Implementación

Hemos realizado un primer paso de análisis del sistema de calefacción, y de como debería de ser nuestro sistema y que debe ofrecer al usuario. Posteriormente hemos seguido con el diseño de nuestro sistema partiendo del análisis. Ahora que ya hemos terminado con esos dos pasos previos, el siguiente paso es empezar con la implementación, empezando desde ejemplos sencillos para aprender como funciona las tecnologías que vamos a emplear, y terminando con el prototipo que hemos creado.

Tenemos que mencionar, que realmente no hemos terminado el proyecto, (explicaremos los motivos más detalladamente en el punto 7 de conclusiones y trabajo futuro). Lo que hemos realizado ha sido un prototipo del sistema real, donde únicamente tenemos una habitación. Tampoco hemos llegado a utilizar la instalación real de calefacción, si no que la hemos simulado en la placa Arduino de la que disponemos. Sin embargo, el paso para llegar a la instalación real es "simplemente" cambiar los LED's que hemos conectado en las salidas, a los interruptores de los elementos (resistencias, electroválvulas, etc.).

Salvando las distancias, el prototipo que hemos realizado es bastante parecido al proyecto que se tendría que desarrollar para el sistema de calefacción, habría que ampliarlo y añadir más elementos. Sin embargo, sirve como base, ya que la mayoría de los problemas encontrados han sido solucionados.

El apartado se va a dividir en tres grandes bloques, que pertenecen a las tres capas en el que se divide el proyecto. El primer paso es explicar de que tratan estas tres capas, y porque hemos seguido ese patrón de diseño. Posteriormente iremos desde la capa más baja a la capa superior, explicando detalladamente todos los elementos que hemos desarrollado e implementado.

6.1 Desarrollo por Capas

Como ya hemos dicho, el proyecto ha sido desarrollado siguiendo el modelo de capas, dividiendo así el sistema en capas independientes pero conectadas entre sí. Está formado por tres capas: la capa hardware, la capa lógica de negocio, y la capa de interfaz de usuario. Hemos seguido un desarrollo ascendente de la capa mas baja, hardware, a la capa mas alta, la interfaz de usuario.

En la primera capa, la hardware, encontramos el sistema de calefacción y la plataforma Arduino. En esta capa la tarea principal es controlar mediante la plataforma Arduino al sistema de calefacción. Para ello emplearemos los comandos que diseñamos en el apartado de diseño, y que ahora definiremos.

La segunda capa, la lógica de negocio, es la encargada de manejar todo el sistema y que funcione correctamente. Aquí encontramos la conexión entre la interfaz de usuario y el hardware, pero además también estarán las reglas de funcionamiento que dictarán el funcionamiento de todo el sistema. Las reglas se basan en los datos recibidos de la capa hardware provenientes del sistema de calefacción, y de los datos de usuario recibidos de la interfaz de usuario; para decidir que acciones tienen que tomar sobre el sistema de calefacción (encender resistencias, apagarlas, etc). Esta capa ha sido desarrollada en Java.

La tercera capa es la interfaz de usuario. Es la capa que sirve al usuario para acceder al sistema, conocer su estado y controlarlo. Hemos implementado una web, de forma que pueda visualizarse en cualquier plataforma.

El desarrollo por capas permite hacer independientes cada parte del proyecto, aunque puede ser algo mas costoso de desarrollar, permite hacer cambios en una capa sin que este cambio afecte al resto del proyecto.

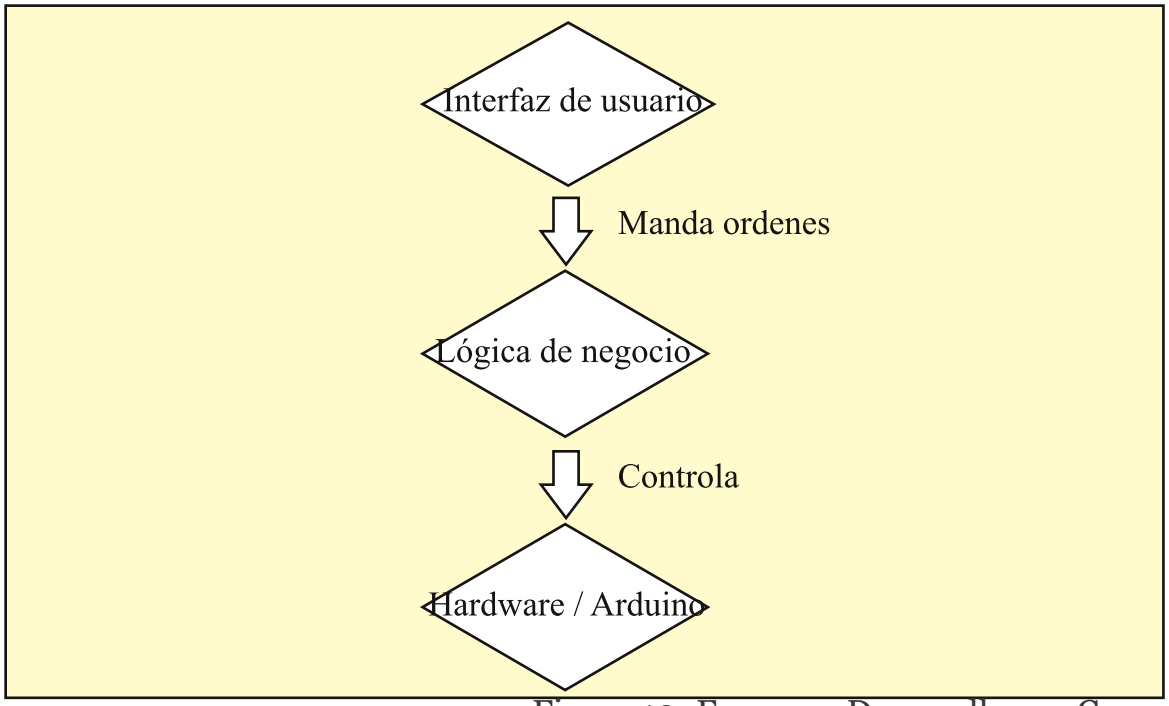

Figura 10: Esquema Desarrollo por Capas

6.2 Capa Hardware

Como hemos dicho en la introducción de este punto, vamos a empezar el desarrollo desde la capa más baja. En esta capa empezamos conociendo como leer temperatura de los sensores de temperatura. Posteriormente estuvimos trabajando en la conexión con la capa lógica, como se realizaba la conexión, enviar y transmitir y como trabajar con los comandos. Finalmente tuvimos que implementar la *API* que hemos diseñado, asignando las acciones a cada comando.

6.2.1 Leer Temperatura

Para leer las temperaturas hemos usado el sensor de Arduino "LM35". Es un sensor que cuenta con tres conectores:
-El de la izquierda y el de la derecha tienen que ir conectados a 0V y 5V respectivamente, para darle electricidad, (depende del lado que mires el sensor, tiene una parte parte plana y otra curva, en la imagen la parte plana esta abajo).
-El conector del centro devuelve un voltaje, este voltaje es el que representa la temperatura leída por el sensor.

A continuación se muestra una fotografía donde se puede ver uno de los sensores de temperatura empleados.

Figura 11: Sensor LM35

Como hemos dicho, el sensor "LM35" devuelve un voltaje dependiendo de la temperatura. Nosotros trabajamos con grados centígrados, no con voltajes, por lo tanto tenemos que transformar ese voltaje en grados. La fórmula es la siguiente ("voltaje" representa el voltaje de entrada que se le proporciona):

```
5* "voltaje" * 100 / 1024
```

O lo que es igual:

```
"voltaje" * 0,48828
```

6.2.2 Serial Command

La parte más importante de la capa hardware es la librería "Serial Command". Es una librería que permite, por una parte leer comandos por la conexión Serial[1] (que utilizaremos más tarde para conectar la placa Arduino con la capa lógica en Java), y por otra parte permite ejecutar métodos o acciones relacionadas con cada comando.

La utilidad de ésta librería es poder mandarle órdenes desde la capa lógica a la capa hardware, y que la capa hardware ejecute acciones dependiendo de las órdenes recibidas. Por ejemplo, la capa lógica decide que no es necesario tener todas las resistencias encendidas y hay que apagar dos. La capa lógica manda la orden de apagar dos resistencias, y Arduino recibe ésta orden y las apaga.

A continuación se muestra la *API* que hemos diseñado, y las acciones asociadas a cada uno de los comandos. La primera parte es el comando que recibiremos de la capa lógica por la conexión Serial, la segunda parte es el nombre de la subrutina que llamaremos, (la implementaremos más abajo, después del "loop").

```
sCmd.addCommand("SRgetTemp",    SRgetTemp );
sCmd.addCommand("SSgetTemp",    SSgetTemp );
sCmd.addCommand("Electtrue",    abrirElectro );
sCmd.addCommand("Electfalse",    cerrarElectro );

sCmd.addCommand("1Res0n",    unoRes0n );
sCmd.addCommand("2Res0n",    dosRes0n );
sCmd.addCommand("3Res0n",    tresRes0n );
sCmd.addCommand("4Res0n",    cuatroRes0n );
sCmd.addCommand("5Res0n",    cincoRes0n );
sCmd.addCommand("1Res0ff",   unoRes0ff );
sCmd.addCommand("2Res0ff",   dosRes0ff );
sCmd.addCommand("3Res0ff",   tresRes0ff );
sCmd.addCommand("4Res0ff",   cuatroRes0ff );
sCmd.addCommand("5Res0ff",   cincoRes0ff );

sCmd.addCommand("Eng1on",   EngineUno0n );
sCmd.addCommand("Eng2on",   EngineDos0n );
sCmd.addCommand("Eng3on",   EngineTres0n );
sCmd.addCommand("Eng1off",   EngineUno0ff );
sCmd.addCommand("Eng2off",   EngineDos0ff );
sCmd.addCommand("Eng3off",   EngineTres0ff );
```

Figura 12: API Arduino

Ahora vamos a explicar rápidamente como se utiliza esta librería. Lo primero es descargarla de Internet, se puede obtener fácilmente, y la añadimos al software donde desarrollamos los programas en Arduino en "Programa -> Include Library -> add .ZIP Library.." . Una vez la hemos añadido en el software, la incluimos en nuestro programa en C de la siguiente manera:

```
#include <SerialCommand.h>
SerialCommand sCmd;
```

1: La conexión Serial es un protocolo de conexión, empleado por ejemplo en las impresoras. Esta conexión funciona a traves de los puertos USB

Los métodos donde se implementan las acciones de cada comando son de la siguiente forma. Tan solo debemos incluirlo en el código fuera del “loop”. Aquí podemos ver tres acciones: encender las resistencias 1, 2, y 3.

```
void unoResOn(){
  digitalWrite(6,HIGH);
  }

void dosResOn(){
  digitalWrite(2,HIGH);
  }

void tresResOn(){
  digitalWrite(3,HIGH);
  }
```

Una vez hemos hecho esto, inicializamos la conexión Serial que permitirá a la placa Arduino recibir información esta conexión. Se realiza con el siguiente comando:

```
Serial.begin(9600);
```

Cuando hemos inicializado la conexión, nos queda leer de la misma para poder recibir los comandos y activar las acciones. La librería que empleamos nos proporciona un sencillo comando, con el que recibiremos toda la información de la conexión Serial. También este comando se encargará de comprobar si la información recibida pertenece a algún comando, y activar la acción del comando recibido.

```
sCmd.readSerial();
```

Por último tenemos que escribir también en la conexión Serial, para enviar información de los sensores a la lógica de negocio, se escribe con el siguiente comando:

```
Serial.println("SS_35");
```

Ahora ya tenemos todo listo en Arduino para leer las acciones que nos indice la capa lógica y realizar las acciones pertinentes. El siguiente paso es empezar con la lógica de negocio.

6.3 Capa Lógica de Negocio

La capa lógica hemos dicho que se encarga de recibir información tanto del usuario como de los sensores físicos de la instalación, y los utiliza para decidir que acciones debe tomar. En esta capa es donde más tiempo hemos empleado, ya que por una parte es la más compleja, pero además es la que lleva todo el funcionamiento del sistema. Precisamente por ser la parte que lleva todo el funcionamiento hemos elegido como lenguaje de desarrollo Java.

La primera parte del desarrollo de esta capa fue la conexión con la placa Arduino. En el apartado anterior vimos como preparar Arduino para recibir los comandos, ahora tenemos que enviarlos desde Java. Lo siguiente es empezar a implementar el diagrama de clases que diseñamos anteriormente. Por último añadimos la parte fundamental de esta capa, las reglas de funcionamiento que regularán y se encargarán del manejo del sistema de calefacción físico.

6.3.1 Librería RxTx

El primer paso que dimos en esta capa fue establecer la conexión con la capa inferior, la capa hardware, con Arduino. Para esta tarea creamos la clase túnel y empleamos la librería de Java "RxTx". Es una librería no muy conocida, implementada por una persona ajena a Oracle (propietaria de Java). Desde el 2009 lleva sin modificarse, pero de momento sigue funcionando. A continuación vamos a explicar como funciona esta librería y como establecer la conexión mediante el puerto Serial con la placa Arduino.

Lo primero que vamos a hacer es establecer la conexión. En el cpi, tenemos que indicar el puerto *Com* al que esté conectado nuestro Arduino:

```
private static SerialPort p = null;

//iniciamos la conexion con arduino.
public void iniciarConexion(){



   try {


      CommPortIdentifier cpi=CommPortIdentifier.getPortIdentifier("COM5");

      p=(SerialPort) cpi.open("", 1000);
      p.setSerialPortParams(9600,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE );
      //Para dar tiempo a que arduino se inicie, y luego empezar a mandarle comandos.
      Thread.sleep(2000);

   } catch (Exception e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
   }

}
```

Figura 13: Iniciar conexión RxTx Java-Arduino

Lo siguiente que hacemos es añadir el *listener* para que así, cuando haya un envío de información por parte de Arduino, saltará el *listener* para poder recibir esa información y tratarla. (Más adelante en el punto 6.3.4 se explicará mas detalladamente el funcionamiento de los *listener's*):

```
//iniciamos el Listener.
public void iniciarListener() {
try
{
   p.addEventListener(this);
   p.notifyOnDataAvailable(true);
}
catch (Exception e) {}

}
```

Figura 14: Iniciar listener RxTx

A continuación lo que tenemos que hacer es implementar el método que será el encargado de recibir lo que envíe Arduino, el método se tiene que llamar "serialEvent":

```
//Recibimos de arduino.
public void serialEvent(SerialPortEvent evt) {
   String in = "";
   System.out.println("leyendo");

   if (evt.getEventType() == SerialPortEvent.DATA_AVAILABLE)   {
      try   {

         while (p.getInputStream().available() > 0) {

            //Para dar tiempo a que arduino vaya escribiendo
            try {              Thread.sleep(50);
            }   catch (InterruptedException e1) {       e1.printStackTrace();  }


            int dato = p.getInputStream().read();
            in=in+(char)dato;

          }//cierre while


      }     catch (Exception e)    {     }
```

Figura 15: Implementación Método serialEvent

Este método proviene de la interfaz "SerialPortEventListener" que debemos implementar en esta clase que se encarga de la conexión Serial.

```
public class TunelSerie implements SerialPortEventListener{
```

Lo siguiente que hay que hacer es tratar la información recibida, en nuestro caso desde Arduino envíamos primero el identificador del sensor correspondiente, y después la información, separado por un "_" :

```
System.out.println(in);

//Dividimos lo que recibimos en identificador y dato.
String recibido[] = in.split("_");
String identificador = recibido[0];
String datos = recibido [1];
//asi quitamos del numero, los \r\n que lleva al final, como final de cadena.
//PERO SI USAMOS(EN ARDUINO) SERIAL.PRINT, Y NO SERIAL.PRINTLN NO HACE FALTA

datos = datos.substring(0, datos.length() - 2);
// System.out.println(temp);
double dato =  Double.parseDouble(datos);

for( int i = 0 ; i <Drivers.size() ; i++){

    Drivers.get(i).devolverDato(identificador, dato);
  }
}
```

Figura 16: Tratar información método serialEvent

Lo que hacemos en la última parte es avisar a todos los drivers, para enviarles la información recibida. En los siguientes apartados se explicará mas detalladamente.

Una vez hemos establecido la conexión y hemos implementado el método que recibe la información de la conexión Serial, nos queda enviar información por la conexión:

```
public void enviarArduino(String comando) {

   try {
      p.getOutputStream().write(comando.getBytes());
      p.getOutputStream().flush();
      System.out.print("enviando:  " + comando);
      Thread.sleep(60);
   } catch (Exception e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
   }
}
```

Figura 17: Método enviar a Arduino

Hacemos un "sleep" después de enviar ya que se dan situaciones en las que se envían varias órdenes a la vez. Y como la placa Arduino tiene mucha menos potencia de cálculo que un ordenador normal, puede que se envíe información mas rápido de lo que Arduino puede procesar, así le damos tiempo a Arduino a procesar toda la información.

6.3.2 Esquema Sistema Java

Una vez establecida la conexión, el siguiente paso era implementar el sistema con todas las clases siguiendo el diagrama de clases que diseñamos en la fase de diseño.

A continuación se muestra un esquema básico y simplificado de como está diseñado el sistema Java. Como se puede ver, se ha empleado una vez mas el desarrollo por capas, yendo de la capa superior "App" a la capa mas inferior "driver" que conecta con el túnel Serial.

Las reglas de funcionamiento se encargan de controlar el funcionamiento del sistema y dar órdenes, y los *drivers* de enviar a Arduino las órdenes de las reglas de funcionamiento. En los siguientes apartados se explicará mas detalladamente el funcionamiento ambos elementos.

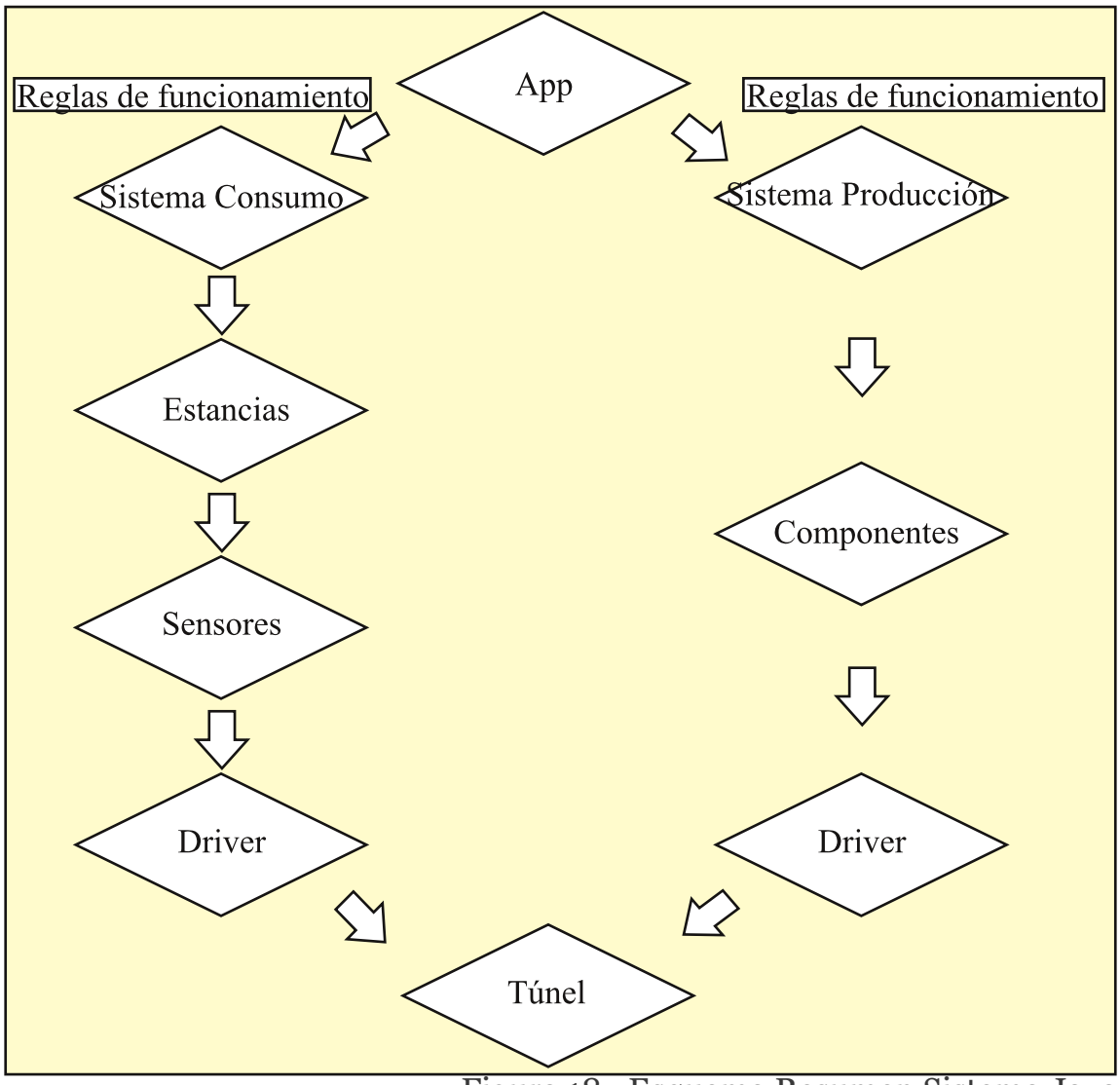

Figura 18: Esquema Resumen Sistema Java

6.3.3 *Driver*

Los *drivers* son unos programas o librerías que sirven para acceder a componentes físicos generalmente. Por ejemplo las impresoras, cada una tiene sus *drivers* que sirven para poder imprimir, escanear... Los programas como por ejemplo el Word, no saben como imprimir, solo saben que mandándole la orden *"print"* a los *drivers*, ellos se encargan de conectar con la impresora y imprimir. Básicamente sirven para separar los programas de los componentes físicos, de forma que los programas solo se preocupen de conectar con los *drivers*. Y estos se encargan de conectar con los componentes físicos. Así si cambiamos de impresora no tiene que cambiar el Word, tan solo tienen que cambiar los *drivers*.

Nosotros hemos usado unas clases *drivers* para conectar las clases que representan los elementos del sistema real (sensores, motores, etc.), con el túnel RxTx. El motivo de hacerlo así separado es el mismo que hemos explicado, separar los elementos del túnel, de forma que si cambiamos el tipo de conexión solo tendremos que cambiar los *drivers* y no todo el sistema.

La implementación que hemos hecho es muy simple. Los elementos tienen métodos para apagarlos, encenderlos o las operaciones propias de cada uno, por ejemplo los métodos de un motor:

```
DiverMotor dm ;

public void encender(){
   dm.encender();
}

public void apagar(){

   dm.apagar();
}
```

El motor tiene las acciones que puede realizar, encenderse y apagarse. Lo que hacemos es mandar la orden a su *driver.*

El *driver* ya envía al túnel lo que éste tiene que enviar por la conexión Serial a Arduino.

```
public void encender (){

   String comando = "Eng"+ id +"on\n";
   tunel.enviarArduino(comando);
}

public void apagar(){

   String comando = "Eng"+ id +"off\n";
   tunel.enviarArduino(comando);
}
```

6.3.4 Listener

Listener es un patrón de programación que facilita el "aviso" a objetos ante un evento que ha ocurrido y ante el cual esos objetos tienen que actuar. Dicho de otra forma mas práctica tomando nuestro proyecto. Arduino envía a Java información. En nuestra clase RxTx salta un evento, que recogemos en el método "serialEvent", y realizamos las operaciones que sean necesarias.

Este patrón lo hemos usado por todo el sistema, hemos basado el funcionamiento del sistema en eventos. Poniendo un ejemplo:
Cuando Arduino manda una temperatura ambiente al sistema Java, en la clase túnel salta un evento de que se ha recibido algo por el puerto Serial. A continuación se avisa mediante otro evento a todos los *drivers*, estos activan su método *listener*, realizan la comprobación de si la información es para ellos, y avisan a su respectivo componente y si hace falta este componente avisa a su superior. Hasta que se activa la regla que esté preparada para recibir ese evento y realizar algún cambio si hace falta. En nuestro ejemplo, si la temperatura ambiente que recibe está lejos de la temperatura de confort que ha pedido el usuario, la regla activará mas resistencias para calentar más el agua.

A continuación vamos a ver un ejemplo de ésta implementación:

Cuando recibimos algo por la conexión Serial avisamos a todos los *drivers*:

```
for( int i = 0 ; i <Drivers.size() ; i++){

   Drivers.get(i).devolverDato(identificador, dato);
}
```

El driver recibe el aviso en el método asignado:

```
public void devolverDato(String id, double temp){
   //Si el id del sensor de arduino que ha mandado la temperatura es este,
   //la cogemos y la mandamos a nuestro sensor para que la actualice.
   if(this.id.equals(id)){
   stl.nuevaTemperatura(temp);
   }
}
```

Para asegurarnos de que la clase implemente el método listener, hemos diseñado una interfaz conla declaración de dicho método, y la implementamos en la clase que tiene que escuchar eventos. Además también necesitamos que implemente dicha interfaz para la declaración de los listeners en la clase que lanza los eventos:

```
import java.util.EventListener;


public interface DriverListenerSensorTemperatura extends EventListener{
   public void devolverDato(String id, double temp);
}
```

```
public class DriverSensorTemperatura extends Driver implements DriverListenerSensorTemperatura{
```

Lo último que nos queda es añadir como *listener*, la clase que escucha un evento, a la clase que los lanza:

```
  public void addDriverListener(Driver d){

     this.Drivers.add(d);
  }
```

```
super.getTunel().addDriverListener(this);
```

Este esquema de funcionamiento por eventos se ha repetido por todo el sistema Java. Al principio puede ser un poco complicado o dificil de entender, pero una vez se comprende el funcionamiento es sencillo de utilizar.

6.3.5 Reglas de Funcionamiento

Una vez tenemos implementadas todas las clases, lo último que nos queda es añadir las reglas de funcionamiento. Son *listeners* que están esperando a que les llegue un evento de un cambio en el sistema real de calefacción, comprueban el evento (actualización de temperatura ambiente por ejemplo), y a partir de la nueva información deciden si deben de hacer un cambio (encender o apagar alguna resistencia).

A continuación se muestra la regla cuando se activa la regla de una habitación y lo que realiza:

```
public void nuevaTemperatura(SensorTemperatura Sensor) {
    //avisar a reglas que tienen que saber que ha cambiado la temp

    //ademas actualizamos nuestras temperaturas.

    System.out.println(Sensor.id +"    " + Sensor.temp);
    if(Sensor.id.equals("SensorReal"))
    this.temperaturaReal = Sensor.temp;
    else this.temperaturaSuelo = Sensor.temp;


    regla.activarRegla(this);

}
```

Figura 19: Activar regla funcionamiento

```
public void activarRegla(Habitacion h) {
   double real = h.temperaturaReal;
   double suelo = h.temperaturaSuelo;
   int confort = h.temperaturaConfort;
   SistemaProduccion sp = dormitorio.getSistemaProduccion();
   SistemaConsumo sc = dormitorio.getSistemaConsumo();
   Resistencia resistencias [] = sp.getResistencias();
   Motor motoresSC [] = sc.getMotores();

   Motor motorSP  = sp.getMotor();
   //HACER COMPROBACIONES DE TEMPERATURAS, Y ABRIR O CERRAR VALVULA DE PASO, O APAGAR O E
   //Y APAGAR O ENCENDER MOTORES DE CONSUMO.
   if(real >= confort) {
      h.cerrarValvula();
      //cerramos electrovalvula y apagamos todas las resistencias.
      for(int i = 0; i<=4;i++){
         resistencias[i].Off();
      }
      //apagamos motores
      motoresSC[0].apagar();
      motoresSC[1].apagar();
      motorSP.apagar();
   }
   else {
      motoresSC[0].encender();
      motoresSC[1].encender();
      motorSP.encender();
```

Figura 20: Implementación Regla funcionamiento

La regla hace muchas comprobaciones y en base a esas comprobaciones manda las órdenes necesarias. Ahí solo se muestra una parte de todo el método.

6.4 Interfaz de Usuario

Una vez realizada la lógica de negocio nos queda la última parte, la interfaz de usuario. La tarea de la interfaz es permitirle al usuario del sistema controlarlo (encenderlo/apagarlo, insertar la temperatura de confort deseada..), y también conocer el estado del sistema (temperatura actual, estado de las resistencias, etc.).

En el punto de tecnologías hemos explicado el patrón de diseño *REST*, ahora lo vamos a aplicar para la interfaz. De esta forma la interfaz podrá acceder a los recursos del sistema que necesite. A su vez, para implementarla vamos a emplear las tecnologías de desarrollo web, con la intención de realizar una interfaz web que sea compatible con todos los sistemas.

El primer paso que vamos a dar es aplicar el patrón *REST*. Posteriormente aplicaremos la tecnología Ajax para realizar las peticiones desde la interfaz al sistema. En este punto encontramos un problema de seguridad que nombramos en el punto 2.8 de tecnologías, hablaremos de él y explicaremos la solución que hemos aplicado. Por último desarrollaremos la parte visible de la interfaz.

En el resumen del proyecto, al principio de la memoria, se habla del Internet de las Cosas, de que todo tiene que estar conectado. En este punto, aplicando el patrón *REST,* se cumple este término, ya que se diseña una interfaz para conectar con el sistema.

6.4.1 *REST*

Para implementar la parte servidora del servicio *REST* utilizamos el sistema java que desarrollamos en la capa anterior para añadirle este servicio. Así nos evitamos tener que desarrollar otro proyecto de Java, y lo ponemos todo junto en el mismo programa. A continuación vamos a ver como se ha desarrollado:

Lo primero que tenemos que hacer es inicializar un "Component" de Java. Este elemento permite iniciar el sistema *REST*. Le indicamos el protocolo y puerto en el que va a funcionar. Lo siguiente es iniciar una "Application" que definiremos en el siguiente paso.

```
Component componente = new Component();
componente.getServers().add(Protocol.HTTP, 8182);
Application app = new RestApplication(sc, sp);
componente.getDefaultHost().attach("/app", app);
```

Figura 21: Implementación *Component* Java

```
public class RestApplication extends Application {

  public SistemaConsumo sc;
  public SistemaProduccion sp;

  public RestApplication(SistemaConsumo sc, SistemaProduccion sp) {
    this.sc = sc;
    this.sp = sp;
  }
  @Override
  public Restlet createInboundRoot( ) {

    Router router = new Router(getContext());
    router.attach("/Resistencias/", ResistenciaRest.class);
    //apaga/enciende el sistema total
    router.attach("/OnOff/", OnOffRest.class);
    router.attach("/Dormitorio/", DormitorioRest.class);
    router.attach("/Flujo/", FlujoRest.class);
    return router;
  }
```

Figura 22: Implementación RestApplication

En esta figura se puede ver como se declara en el método "createInboundRoot" las *Uri's* de cada recurso que hemos considerado necesario. La declaración consiste en dos partes, en la primera se decide la *Uri*, y la segunda se indica la clase java que se encargará de responder de esa *Uri*. Ahora vamos a ver como se implementa esta clase:

En este caso solo hemos implementado el método "Get", ya que desde la interfaz lo único que necesitamos es conocer el estado de las resistencias. Por lo que solo necesitamos este método. Como se puede observar se devuelve la información en formato Json, esto es debido a que esta información será enviada a una página con Javascript, y Json es el formato de información mas extendido y de los más simples de utilizar. En el siguiente apartado veremos como tratar esta información.

```
public class ResistenciaRest extends ServerResource{

  @Get
  public Representation get(){

     RestApplication a = (RestApplication) this.getApplication();
      SistemaProduccion sc =  a.getSistemaProduccion();
     boolean estado [] = sc.getEstadoResistencias();
     JSONObject res= new JSONObject();
     res.put("res1", estado[0]);
     res.put("res2", estado[1]);
     res.put("res3", estado[2]);
     res.put("res4", estado[3]);
     res.put("res5", estado[4]);


     return new StringRepresentation(res.toString(), MediaType.APPLICATION_JSON);

  }
```

Figura 23: ResistenciaRest

6.4.2 AJAX

Una vez hemos implementado el lado del servidor, tenemos que implementar el lado cliente, la interfaz web. Para hacer las llamas *REST* hemos empleado JavaScript. A continuación se muestra como hemos hecho las llamadas:

```
setInterval(function(){
            var dataURL = "http://localhost:8182/app/Resistencias/";
            var proxyUrl = "http://localhost/Arduino_proxy/proxy.php?get=";
            $.ajax({
                url: proxyUrl+dataURL,
                async: true,
                type: "GET",
                dataType: "JSON",

                success: function(data) {
                    if(data['res1'])
                    $('#llama1').fadeIn('fast');
                    else $('#llama1').fadeOut('fast');

                    if(data['res2'])
                    $('#llama2').fadeIn('fast');
                    else $('#llama2').fadeOut('fast');

                    if(data['res3'])
                    $('#llama3').fadeIn('fast');
                    else $('#llama3').fadeOut('fast');

                    if(data['res4'])
                    $('#llama4').fadeIn('fast');
                    else $('#llama4').fadeOut('fast');

                    if(data['res5'])
                    $('#llama5').fadeIn('fast');
                    else $('#llama5').fadeOut('fast');

            }
            } )
```

Figura 24: Llamada Rest en JavaScript

Como se puede observar primero hacemos una llamada a un archivo ".php", pasándole como parámetro en la URL la *Uri* del recurso que queremos solicitar, esto es debido a un problema de seguridad de acceso a Java desde JavaScript, y que se puede resolver usando PHP. Se llama *CORS* y en el siguiente apartado trataremos este tema.

Cuando hacemos la llamada, indicamos que hacemos un "Get", y que el tipo esperado es Json. Si la llamada tiene éxito y obtenemos una respuesta, la ejecución continúa por *"success"*. La respuesta llega por la variable *"data",* y para acceder a la información tan solo necesitamos acceder al campo que queramos. (Hay que recordar que desde Java respondemos con un Json precisamente por ésto, por la facilidad de acceso desde JavaScript).

En este caso en concreto lo que realizamos es mostrar o no una llama por cada resistencia si están apagadas o encendidas.

6.4.3 *CORS*

CORS, o *Cross-Origin Resource Sharing*, es un mecanismo que permite que un navegador muestre recursos de un dominio que son referenciados desde un dominio distinto. Sin embargo, como explicamos en el punto 2.8 Tecnologías: Ajax, hay tecnologías, como JavaScript, que están restringidas a utilizar este mecanismo por razones de seguridad. Sin embargo hay tecnologías que no están limitadas por este mecanismo, como por ejemplo PHP. A continuación vamos a explicar como saltarnos esta restricción.

Para resolverla hemos utilizado PHP, a continuación se muestra el código que se ha utilizado. Hemos separado en dos archivos, uno para hacer peticiones "Get" y otros para hacer peticiones "Put". Podríamos haber puesto todo junto en un mismo archivo, pero hemos preferido hacerlo de esta manera para separar y ordenar los dos distintos tipos de llamadas que realizamos.

```
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: *');
$curl = curl_init();

if (isset($_GET['get'])) {
curl_setopt_array($curl, array(
    CURLOPT_URL => $_GET['get']
));
}


curl_exec($curl);

curl_close($curl);
```

Figura 25: Programación PHP función Get

Las primeras cabeceras permiten el acceso, para poder hacer llamadas "Get". Lo siguiente que hacemos es obtener el parámetro que mandamos después de "Get", (que como se puede ver en el apartado anterior, en la llamada JavaScript mandamos la *Uri* del recurso solicitado). Ahora que ya tenemos la *Uri* de destino, hacemos la llamada, y la respuesta la trasladará a JavaScript.

En la siguiente página mostramos el código para procesar llamadas "Put".

```php
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: *');
    //datos a enviar
    $url = $_GET["url"];
    $data= array("action"=>$_GET["action"]);

    //url contra la que atacamos
    $ch = curl_init($url);
    //a true, obtendremos una respuesta de la url, en otro caso,
    //true si es correcto, false si no lo es
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    //establecemos el verbo http que queremos utilizar para la petición
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PUT");
    //enviamos el array data
    curl_setopt($ch, CURLOPT_POSTFIELDS,http_build_query($data));
    //obtenemos la respuesta
    $response = curl_exec($ch);
    // Se cierra el recurso CURL y se liberan los recursos del sistema
    curl_close($ch);
    if(!$response) {
        return false;
    }else{
        var_dump($response);
    }
```

Figura 26: Programación PHP función Put

En este caso, al igual que en el caso anterior, obtenemos la *Uri* de destino, pero además tambien obtenemos la *"action"*, que mandamos también por "Get".

Lo siguiente que realizamos es, una vez que obtenemos la respuesta, la devolvemos mediante el método "Put" a JavaScript.

6.4.4 Interfaz Web

Llegados a este punto ya lo único que nos quedaba por realizar era la interfaz web que verá el usuario. Hemos utilizado HTML y CSS para diseñarla y JavaScript para mostrar o cambiar las imágenes que actualizan el estado del sistema. A continuación se muestra la vista principal, donde podemos ver arriba a la izquierda un botón para encender o apagar todo el sistema de producción de calor. A la derecha de éste se encuentra un botón *"Room"* que sirve para cambiar de vista a la siguiente vista que explicaremos a continuación. Por último podemos ver las llamas que representan las resistencias que están activas, y unos iconos que muestran si hay flujo o no en las tuberías.

(Por simplificar, hemos supuesto que los motores están encendidos si hay flujo, si no hay flujo están apagados. En caso de que estén encendidos y no haya flujo por una obstrucción, el sistema lo detectará y apagará inmediatamente los motores)

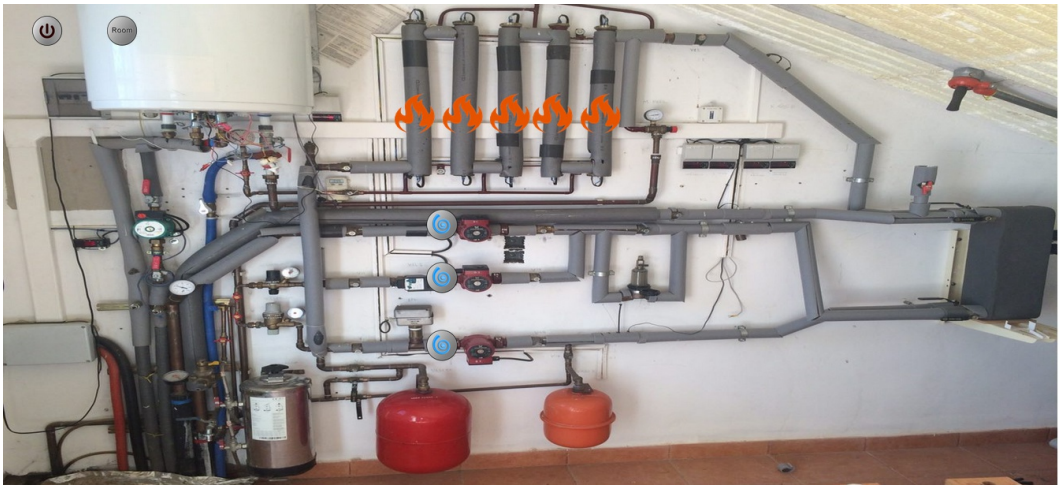

Figura 27: Vista Principal de Interfaz Web

La siguiente vista corresponde a la habitación. Aquí podemos apagar la calefacción en esta habitación únicamente con el botón superior izquierdo. Podemos seleccionar la temperatura de confort que deseamos mediante un desplegable donde se encuentran unos valores comunes. En la parte superior derecha se nos muestra las temperaturas actuales tanto de confort como la real de la habitación. Mediante un botón que se encuentra abajo a la izquierda podemos volver a la vista principal anterior.

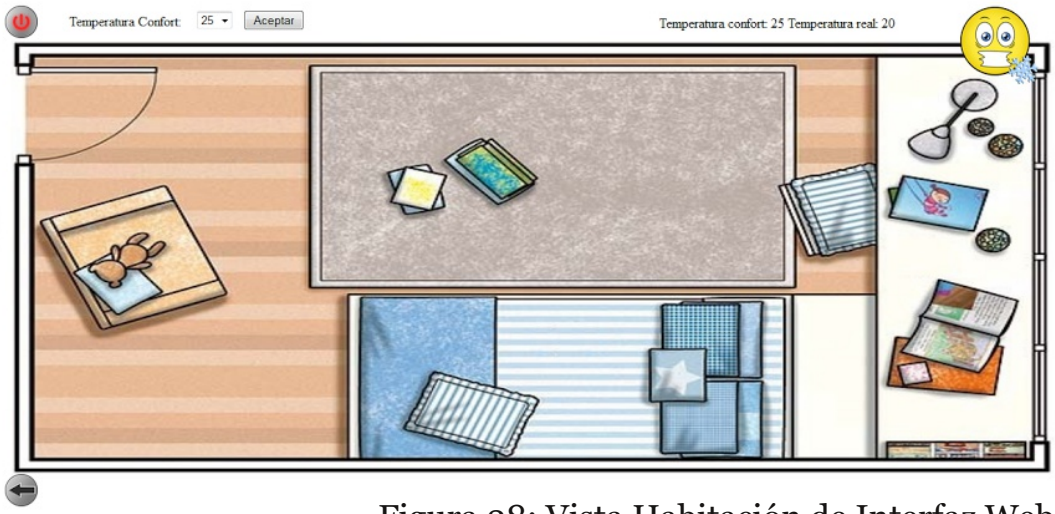

Figura 28: Vista Habitación de Interfaz Web

7 Conclusiones

Una vez hemos acabado con el proyecto, tenemos que hablar de las conclusiones a las que hemos llegado. Personalmente voy a hacer dos conclusiones, una sobre el "mundo" de la domótica, y otro sobre el desarrollo del proyecto.

El porqué elegí este proyecto se debe a que cuando me encontré en la situación de tener que elegir un proyecto, quería hacer uno que me permitiera realizar cosas "físicas", no sólo hacer un programa que se vea en la pantalla del ordenador. Por otra parte la domótica me llamaba la atención, porque me parecía un mundo interesante y desconocido, y porque cumplía mis deseos de hacer un proyecto "más físico".

Este proyecto me ha permitido iniciarme en el mundo de la domótica, he conocido diferentes tecnologías que hay actualmente en el mercado, y he descubierto que no es tan complicado como parece desarrollar un sistema de domótica. También me ha permitido iniciarme en la plataforma Arduino, una plataforma "barata" pero con muchas posibilidades en este y otros campos. A partir de ahora, espero aprovechar todos los conocimientos que he adquirido y seguir investigando, aunque sea por cuenta propia.

Sobre la domótica, era un campo que anteriormente solamente había visto en películas y series de televisión. Apenas había leído alguna cosa por Internet, y nunca había realizado ningún proyecto ni desarrollado ningún sistema ni aplicación. Ahora que he terminado, Mis son que es un mundo aún por descubrir. Hay muchas empresas haciendo investigaciones para crear y mejorar sus productos. Pienso que faltan aún algunos años para que llegue de verdad al mercado, pero al igual que hace 15 años era casi impensable que los móviles sirviesen para algo más que llamar y recibir llamadas, la domótica llegará a los hogares de todas las personas.

Mis conclusiones en cuanto a la domótica son que es un mundo aún por descubrir. Hay muchas empresas haciendo investigaciones para crear y mejorar sus productos. Pienso que faltan aún algunos años para que llegue de verdad al mercado, pero al igual que hace 15 años era casi impensable que los móviles sirviesen para algo más que llamar y recibir llamadas, la domótica llegará a los hogares de todas las personas.

Respecto al proyecto, me ha costado bastante desarrollarlo debido a que no tenía conocimientos previos en este campo. Pienso que he ido un poco lento en algunos momentos del proyecto, y eso no me ha permitido terminarlo como a mi me hubiera gustado, (en el siguiente apartado de trabajo futuro hablaré sobre este tema, donde nos hemos quedado desarrollando y cuáles son los puntos siguientes que quedan por realizar).

También, anteriormente no tenía mucha costumbre de realizar análisis y diseño previo a la etapa de implementación. Durante el proyecto he sufrido esta falta de costumbre, y en el momento de acabar, me he dado cuenta de que en algunos momentos tenía que haber realizado más trabajo de análisis y diseño. Ahora ya he aprendido la importancia de estas dos fases previas, y las tendré más en cuenta a los proyectos futuros que realice.

7.1 Trabajo Futuro

Primero vamos a hablar en que punto nos hemos quedado en el proyecto, y que queda por realizar.

Como hemos dicho en las conclusiones personales, nos ha faltado tiempo al final y solo hemos hemos podido realizar un pequeño prototipo de la instalación real. Este prototipo se ha desarrollado en la placa Arduino propia que tenemos. Lo hemos realizado tomando el sistema de producción y una sola habitación como sistema de consumo.

En el prototipo se representan con cinco LED's las cinco resistencias de las que se compone el sistema de producción. También hay tres LED's amarillos que representan las tres bombas de agua que mueven el agua por los circuitos. Y por último un LED blanco que simulan la electro-válvula que abre el paso de agua a la habitación, únicamente consideramos si está abierta totalmente o cerrada totalmente.

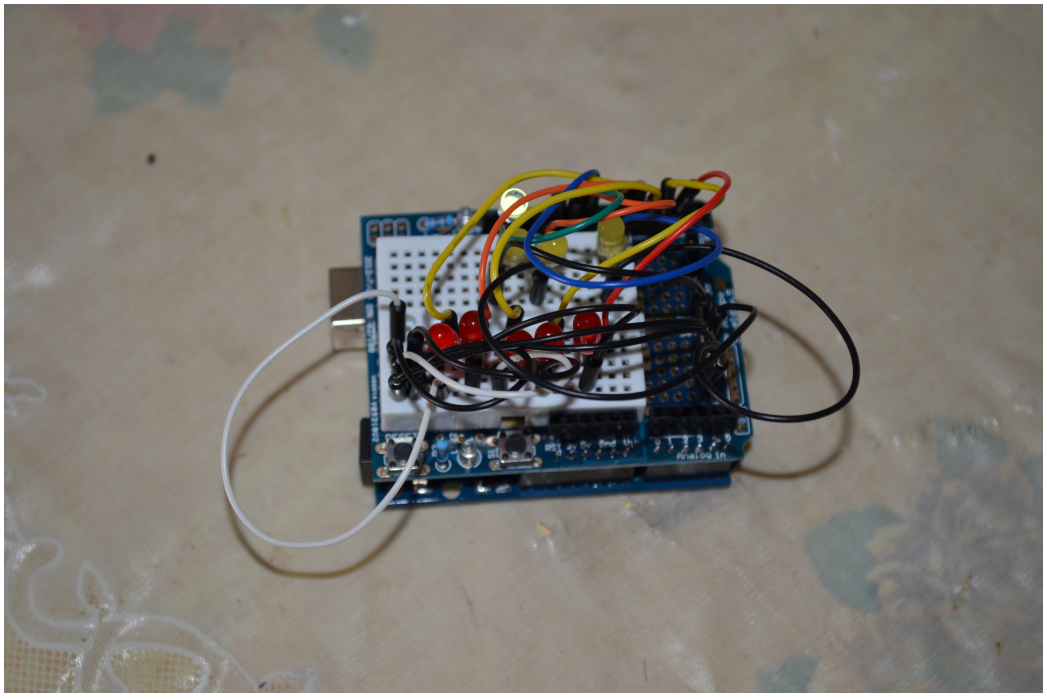

Figura 29: Prototipo sistema de calefacción

Ahora que hemos dicho dónde nos hemos quedado, hay que hablar del trabajo que queda por delante.

Lo primero que hay que mencionar, es que las reglas de funcionamiento las hemos diseñado de forma muy simple, posiblemente haya que perfeccionarlas analizando exactamente como funciona el sistema de calefacción actualmente. También hay que añadir más reglas, ya que las que hemos realizado únicamente son para el prototipo con una habitación.

Como hemos dicho, hemos hecho un prototipo, no hemos llegado a tocar el sistema de calefacción físico. Sin embargo, antes de empezar a instalar nada hay que tratar otro tema. Necesitaremos instalar placas Arduino por toda la casa, una en cada habitación y una más mínimo en la instalación de producción de calor. Estas placas tienen que conectarse entre si, para transmitir las acciones y los datos de los sensores. Para conectarse se pueden usar cables por toda la casa, una instalación WiFi, módulos de radio-frecuencia, etc.

También hay que añadir el sistema de producción de calor por placas solares, ya que por falta de tiempo no lo hemos añadido a nuestro prototipo.

Por último, hay que instalarlos por toda la casa. Conectar uno a los interruptores de las resistencias y bombas de agua, y conectar uno en cada estancia de la casa para controlar las temperaturas y las electroválvulas que dan paso.

8 Bibliografía

Apartado Caso de Estudio:

Sistemas de calefacción
http://www.enforce-een.eu/esp/category/tecnologias/sistemas-de-calefaccion/

Wikipedia - Calefacción
https://es.wikipedia.org/wiki/Calefacción

Suelo Radiante
http://www.emesico.com/es/eficiencia-energetica/suelo-radiante/

Suelo Radiante: Qué es y Cómo funciona
http://nergiza.com/suelo-radiante-que-es-y-como-funciona/

El confidencial
http://www.elconfidencial.com/tecnologia/2015-01-15/por-que-una-bomba-de-calor-es-el-sistema-de-calefaccion-mas-eficiente_621981/

Apartados Análisis y Diseño:

Apuntes Asignatura ISW, Tema 3: Modelado OO con UML. 4º ETSINF, UPV

Apartado Implementación:

Arduino
http://forum.arduino.cc/
http://playground.arduino.cc/

Foro domótica con Arduino
http://domotica-arduino.es/foro/

Installing RXTX for Serial Communication with Java
https://www.henrypoon.com/blog/2010/12/25/installing-rxtx-for-serial-communication-with-java/

Acceder al puerto serie programando en Java
http://www.hell-desk.com/acceder-al-puerto-serie-programando-en-java-2a-parte/

Using RXTX In Eclipse
http://rxtx.qbang.org/wiki/index.php/Using_RXTX_In_Eclipse

Tutorial Java+Arduino
https://www.youtube.com/watch?v=4Hr_LZ62SdY

Stackoverflow
http://stackoverflow.com/

Eventos personalizados en Java
https://www.youtube.com/watch?v=B-f7gewzJh8

Apuntes Asignatura IAP, Tema 4. 4º ETSINF, UPV

Restlet API
http://restlet.com/technical-resources/restlet-framework/javadocs/2.2/jse/api/index.html?org/restlet/package-summary.html

Wikipedia - CORS
https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

Manual PHP
http://www.php.net/manual/

Documentación Adicional:

How to do RF Communication between 2 Arduinos
http://www.hackshed.co.uk/rf-communication-between-2-arduinos/