Grado en Ingeniería en Informática

Trabajo Fin de Grado

Interfaz hombre-máquina para la programación y monitorización de un controlador programable

---

Autor: Daniel Rivero Guerra

Tutor: Pedro Medina Rodríguez

Julio, 2018

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/Dª Daniel Rivero Guerra, autor del Trabajo de Fin de Título, Interfaz hombre-máquina para la programación y monitorización de un controlador programable, correspondiente a la titulación Grado en Ingeniería informática.

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[X] No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 13 de julio de 2018.

El estudiante

Firmado por RIVERO GUERRA DANIEL - 45348162M el día 13/07/2018 con un certificado emitido por AC FNMT Usuarios

Fdo. Daniel Rivero Guerra

A rellenar y firmar **obligatoriamente** por el/los tutor/es

En relación a la presente solicitud, se informa:

Firmado por MEDINA RODRIGUEZ PEDRO - 42793260N el día 13/07/2018 con un certificado emitido por AC FNMT Usuarios

[X] Positivamente

[ ] Negativamente

(la justificación en caso de informe negativo deberá incluirse en el

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

AGRADECIMIENTOS

A mi familia que ha trabajado y me ha apoyado en todo momento para que pudiera llegar a realizar estos estudios.

A mi tutor Pedro Medina por toda la ayuda y dedicación que ha prestado a la realización de este trabajo.

Índice

Índice de ilustraciones

Resumen

En este Trabajo de Fin de Grado se ha diseñado e implementado una interfaz de usuario para un sistema empotrado de bajo coste que actúa como un controlador programable aplicado a tareas de automatización.

Este programa que actúa como interfaz entre el sistema empotrado y un computador de propósito general, es capaz de secuenciar un conjunto de instrucciones variadas orientadas al control de procesos a través de los sistemas físicos sobre los que actúa y además también permitirá monitorizar ciertos aspectos de estos sistemas.

La idea es que este sistema sirva para controlar un ebullómetro utilizado en estudios de equilibrios líquido-vapor en el sector de la química industrial, así como en otros procesos industriales susceptibles de ser automatizados.

Abstract

In this End-of-Degree Project, a user interface has been designed and implemented for a low-cost embedded system that acts as a programmable controller applied to automation tasks.

This program, which acts as an interface between the embedded system and a general purpose computer, can sequence a set of varied instructions oriented to the control of processes through the physical systems on which it operates and also will allow to monitor certain aspects of these systems.

The idea is that this system serves to control an ebullometer used in studies of liquid-vapor equilibria in the sector of industrial chemistry, as well as in other industrial processes susceptible to be automated.

1. Introducción

1.1 Procesadores embedded o empotrados

Puesto que en este Trabajo de fin de Grado la temática se centra en el uso de un procesador empotrado, en esta sección vamos a hablar un poco sobre ellos para familiarizar al lector con el concepto.

Un sistema empotrado hace referencia a un sistema de cómputo caracterizado por el bajo coste y bajo consumo energético que integra en un solo chip el procesador, memoria y el sistema de entrada salida conjuntamente con otros recursos como temporizadores, entradas-salidas analógicas y digitales y diferentes interfaces (SPI, I2C, CAN bus,…) entre otros recursos, que habilitan estos sistemas para el desarrollo de un gran número de aplicaciones que van desde simples tareas de control (electrodomésticos, TV, domótica, robótica, …) hasta aquellas otras, que exigen mayores prestaciones en capacidades de cómputo, comunicaciones y multimedia lográndose verdaderos computadores completos con tamaños muy reducidos, bajo consumo y precio razonablemente bajo.

Los procesadores empotrados dentro de otros sistemas han existido prácticamente desde los inicios de la informática. Un ejemplo es el Whirlwind, un procesador diseñado en el MIT entre finales de los años 40 y principios de los 50. Whirlwind también fue el primer procesador diseñado para soportar operaciones en tiempo real y fue concebido originalmente como un mecanismo para controlar simuladores de aviones. Sin embargo, este sistema era extremadamente grande comparado con los procesadores de hoy en día (contenía más de 4000 tubos de vacío). Más adelante, el primer microprocesador de la historia, el Intel 4004, fue diseñado para un sistema empotrado, era un microprocesador de propósito específico para una calculadora que proporcionaba una serie de operaciones aritméticas. Con el tiempo los desarrolladores se dieron cuenta de que los microprocesadores de propósito general podían ser reprogramados para el uso en otros productos y como tenían un tamaño bastante aceptable se empezaron a usar en industrias como la del automóvil. Por último, los microprocesadores fueron alcanzando distintos niveles de sofisticación y se clasificaron por su tamaño de palabra. Así los procesadores de 8 y 16 bits pasaron a llamarse microcontroladores y se diseñaron para ser de muy bajo costo, haciendo que los fabricantes empezaran a integrarlos cada vez más dentro de dispositivos como teléfonos, electrodomésticos, televisores, etc.

Hoy en día con el auge del IoT o Internet de las cosas, cada vez más aparatos de uso cotidiano poseen uno o más microcontroladores. Este auge ha sido tal que la venta de microcontroladores supera ampliamente a la de microprocesadores de propósito general y se espera que en 2020 haya en todo el mundo más de 25 mil millones de sistemas

empotrados. Hay que añadir que, debido a su bajo coste y a la aparición de herramientas de desarrollo gratuitas, muchos usuarios hayan comenzado a desarrollar también sus propias aplicaciones con estos sistemas ampliando aún más su mercado.

Con todo esto podemos ver que a pesar de que no son visibles y muchas veces no nos damos cuenta de su existencia, hoy día los sistemas empotrados suponen una de las ramas más importantes de la informática ya que dependemos mucho más de ellos que de los ordenadores convencionales.

1.2 Motivación

La motivación de este trabajo surge a raíz de una visita al Laboratorio de Química Industrial de la Escuela de Ingenieros Industriales y Civiles (EIIC) de la Universidad de Las Palmas de Gran Canaria (ULPGC) donde los responsables del mismo comentan ciertas necesidades en la parte informática de los sistemas que utilizan en las tareas de investigación que se centra, en gran parte, en el análisis de mezclas binarias de sustancias en equilibrio líquido-vapor a condiciones isobáricas constantes. Para ello, hacen uso intensivo de un ebullómetro con un sistema informático, ya obsoleto, que permite llevar el sistema a las condiciones de temperatura y presión deseadas por el investigador y, una vez estable, iniciar la toma de muestras de las fases líquido y vapor para su correspondiente estudio.

El objetivo planteado fue desarrollar un sistema de bajo costo y versátil que permitiese al investigador (o estudiantes en otros sistemas) realizar sus experimentos utilizando un interfaz hombre-máquina que le diese la libertad para secuenciar un conjunto de acciones de control y/o lectura de datos sensoriales de forma cómoda y amigable. El sistema informático, así planteado, proporcionaría al investigador flexibilidad para establecer el conjunto de acciones específicas que cada experimento o estudio en particular pudiese requerir.

1.3 Objetivos

Con la motivación anterior en mente, se plantea alcanzar los siguientes objetivos durante el desarrollo del trabajo:

- El principal es desarrollar una aplicación de escritorio compatible con múltiples plataformas. Esta aplicación debe ofrecer a los operarios del ebullómetro una interfaz de usuario sencilla e intuitiva que les permita programar diferentes secuencias de comportamiento para el hardware de control, dependiendo del experimento que se quiera realizar. Además, esta aplicación deberá permitir el control de la ejecución de las secuencias creadas, monitorizar ciertos parámetros de las ejecuciones y almacenar las secuencias en ficheros por si se quisieran reutilizar en algún momento.

- El objetivo secundario es que el sistema resultante sea lo suficientemente abierto para que su funcionalidad no se reduzca al control del ebullómetro, es decir, una vez terminado el sistema, este debería poderse aplicar a otras maquinarias susceptibles de ser automatizadas.

1.4 Justificación de las competencias

La realización de este trabajo ha hecho que se cubran ciertas competencias, dentro de ellas se encuentra la competencia propia del Trabajo de Fin de Grado:

TFG01: Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.

Y también se han cubierto las siguientes competencias específicas de la mención Ingeniería de Computadores:

IC01: Capacidad de diseñar y construir sistemas digitales, incluyendo computadores, sistemas basados en microprocesador y sistemas de comunicaciones.

IC02: Capacidad de desarrollar procesadores específicos y sistemas empotrados, así como desarrollar y optimizar el software de dichos sistemas.

2. Metodología y plan de trabajo

La metodología de trabajo empleada a la hora de realizar este proyecto ha sido un modelo de desarrollo iterativo e incremental.

En la primera iteración se desarrolló un prototipo mínimo que permitió materializar la concepción inicial del trabajo. Una vez el prototipo mínimo estuvo listo, se fueron aplicando sobre él sucesivas funcionalidades que permitieron alcanzar el sistema final deseado.

En el siguiente esquema se puede ver de un modo más claro las etapas de las que se compuso cada una de las iteraciones del proyecto:

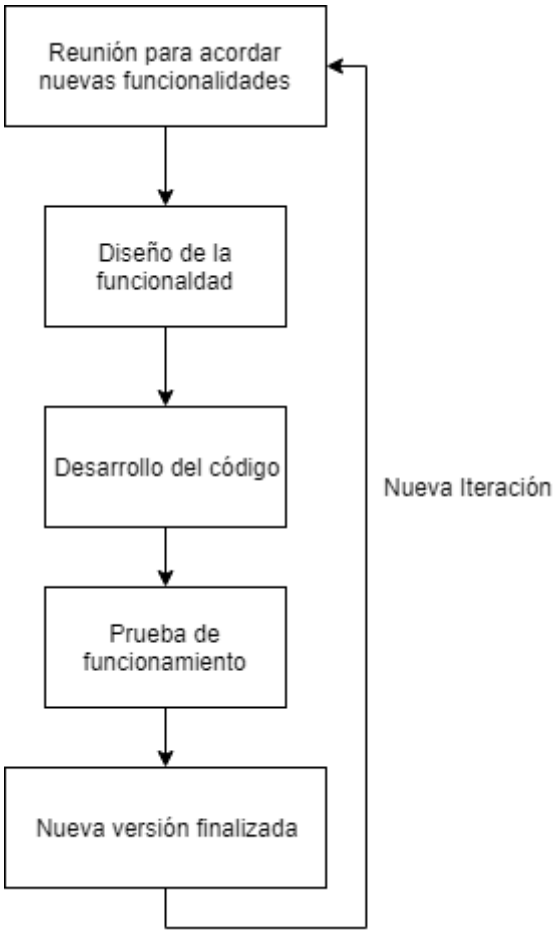

Ilustración 1: Esquema del modelo iterativo

3. Recursos utilizados

En la realización de este Trabajo de Fin de Grado se han utilizado diversos tipos de recursos tanto software como hardware. A continuación, se enumerarán y se justificará su uso:

3.1 Recursos Software

- **NetBeans IDE:** Es un entorno de desarrollo libre, pensado principalmente para desarrollo de programas en el lenguaje Java. Dentro de todos los entornos de desarrollo disponibles me decanté por NetBeans debido a que estoy bastante familiarizado con su uso y además proporciona herramientas que facilitan en gran medida el desarrollo de aplicaciones para Java Swing.

- **Arduino IDE:** Es un entorno de desarrollo de código abierto que proporciona las herramientas necesarias para escribir código para placas Arduino, así como compilar y cargar este código en las placas.

- **Draw.io:** Es una herramienta gratuita y online que permite hacer diagramas. Fue utilizada para realizar las figuras que aparecen en esta memoria.

- **Fritzing:** Herramienta utilizada para elaborar esquemas de conexión de la placa Arduino con diferentes componentes hardware.

- **Herramientas ofimáticas:** Utilizadas para la redacción de la memoria y la realización de la presentación.

3.2 Recursos Hardware

- **Placa Arduino Mega 2560:** Es una placa empleada generalmente para realizar prototipos de proyectos de electrónica. Esta placa posee el microcontrolador Atmega2560. El motivo principal por el cual se empleó esta placa en el desarrollo del proyecto es porque ofrece una gran cantidad de pines para manipular, cosa que es de gran importancia en este proyecto.

- **Hardware:** Además de la Arduino Mega, se han utilizado una serie de componentes hardware compatibles con la plataforma de desarrollo Arduino para que el sistema final fuese funcional. Entre los componentes utilizados están:

    - Módulos de relé opto aislados electromecánicos.

    - Módulos de relé opto aislados de estado sólido o MOSFET de potencia.

    - Circuitos supresores de arco (apaga chispas) en el control de cargas inductivas (motores y solenoides) para proteger los dispositivos de mando.

    - Componentes electrónicos varios

- **Fuentes de alimentación e instrumental (voltímetro, osciloscopio, …):** Durante las pruebas y ajustes del prototipo.

3.3 Otros

- **Libro:** Se empleó también el libro “Java 7” del autor Herbert Schildt. Este libro fue empleado para resolver todas las dudas que fueron surgiendo a medida que se desarrollaba el programa de Java.

- **Documentación técnica de dispositivos:** Material como hojas de características, esquemas de conexión, etc.

4. Análisis

4.1 Estado del arte

De entre las tecnologías que se encuentran hoy en día en funcionamiento que más se asemejan al proyecto realizado para este Trabajo de Fin de Grado se encuentran los controladores lógicos programables o PLC. Por tanto, vamos a ver qué son los PLC y en qué estado de desarrollo se encuentra la tecnología en estos momentos.

Un PLC es un computador utilizado en la en la automatización industrial enfocado a automatizar procesos electromecánicos tales como el control de la maquinaria en fábricas de diversos tipos. Estos computadores se diferencian de los de propósito general en varios aspectos clave:

- Están diseñados para tener múltiples señales de entrada y de salida.
- Funcionan en rangos de temperatura bastante amplios, ya que las condiciones en una fábrica pueden ser bastante adversas.
- Están inmunizados al ruido eléctrico.
- Son resistentes a vibraciones e impactos.

Los PLC son sistemas de tiempo real, es decir, los resultados de las salidas deben responder a las condiciones de entrada en un tiempo limitado, ya que de lo contrario no se producirían los resultados esperados.

En cuanto al desarrollo de la tecnología, los primeros PLC se diseñaron para reemplazar los sistemas de relés lógicos. Aquellos PLC se programaban mediante un listado secuencial de instrucciones en lenguaje máquina que el procesador se encargaba de ejecutar. Hoy en día han evolucionado para incluir controles de movimientos, de procesos, sistemas de control distribuido y comunicaciones por red, llegando algunos de ellos a ser prácticamente equivalentes a computadores de escritorio. Los PLC modernos pueden ser programados de diversas maneras, como diagramas de contactos y dialectos adaptados de los lenguajes de programación BASIC y C. Otro método utilizado también en su programación es la lógica de estado, que no es más que un lenguaje de alto nivel diseñado para programar PLC basados en diagramas de estado.

Una vez vistos los PLC, ahora veremos un poco el estado actual las tecnologías de prototipados de bajo costo que son con las que se ha desarrollado este proyecto. Últimamente estas tecnologías han ganado bastante protagonismo debido a la sencillez de su uso y que están al alcance de todo el mundo. Se utilizan principalmente en tareas de aprendizaje, por entretenimiento o incluso en ciertos proyectos para los que ofrecen unas prestaciones adecuadas.

Entre las plataformas más utilizadas se encuentran:

- **Arduino:** Es la plataforma utilizada en este proyecto. Consiste en una placa que monta un microcontrolador, normalmente Atmel, que permite al usuario acceder y manipular los pines de dicho microcontrolador. Hay varios formatos con distintos tipos de microcontroladores que ofrecen más o menos pines con varias funcionalidades dependiendo del uso que se le quiera dar.
- **Raspberry Pi:** Esta a placa a diferencia de la Arduino, no posee un microcontrolador sino un microprocesador de arquitectura ARM, lo que lo convierte realmente en un ordenador de placa reducida. Posee salidas de video y audio, puertos USB, interfaz de ethernet, Wifi, así como una serie de pines para la conexión de componentes electrónicos. Normalmente se usa con un sistema operativo Linux.
- **Udoo:** Es una plataforma que combina un Arduino 2 con computador de placa reducida de arquitectura ARM. La Arduino se encuentra conectada al microprocesador a través de un bus interno de la placa permitiendo la comunicación entre ambos y confiriendo al sistema de un gran potencial. Esta placa se puede utilizar tanto con un sistema operativo Linux como con Android.

Para terminar, vamos a mostrar algunas gráficas relativas a un estudio del año 2017 de Aspencore sobre el número de microcontroladores que usan los usuarios en sus aplicaciones de sistemas empotrados, así como los fabricantes más importantes.

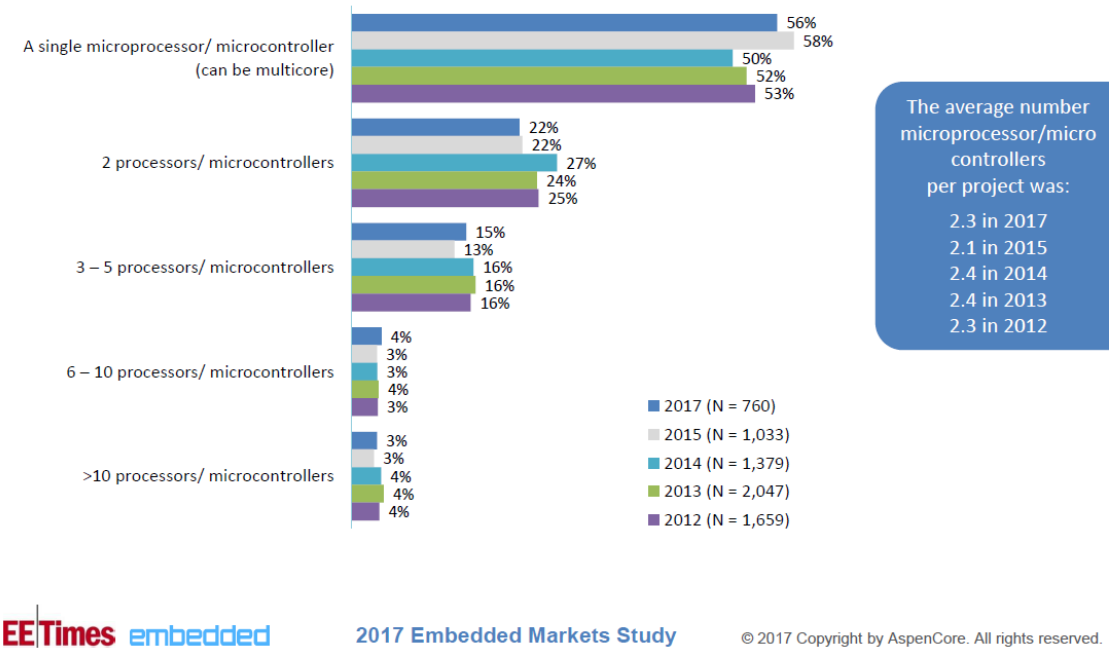

Ilustración 2: Número de procesadores en un sistema empotrado

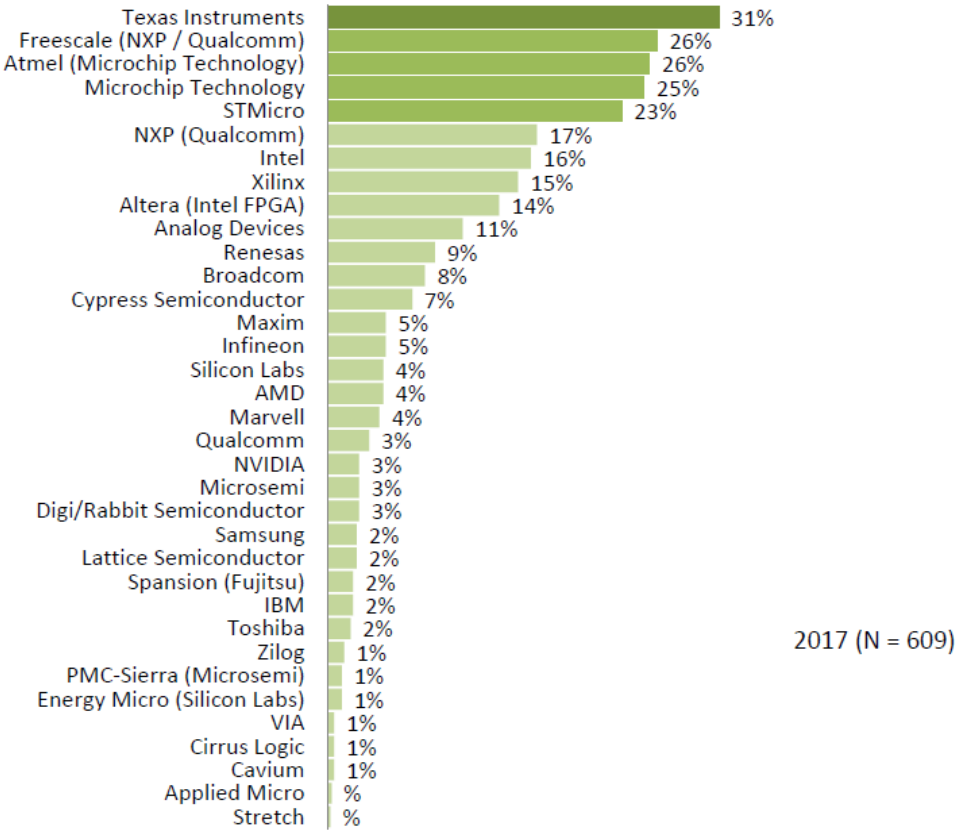

Ilustración 3: Porcentaje de ventas de procesadores por marca

4.2 Aportaciones de este trabajo

Una vez visto en el apartado anterior una idea general de en qué consiste la tecnología de los PLC y el estado en el que se encuentra en la actualidad, ahora veremos qué diferencias tienen estos con la tecnología desarrollada durante el trabajo, así como las ventajas que aporta respecto a la tecnología existente.

Como se puede ver, los PLC requieren de un cierto nivel de conocimientos informáticos para poder programar su funcionamiento ya que muchas veces esto se hace mediante lenguajes de programación. Si recordamos los objetivos mencionados en la introducción, uno de ellos era conseguir un sistema que pudieran manejar usuarios sin conocimientos informáticos y queda patente que un PLC no es la mejor opción en este caso. Por ello, el sistema resultante en este proyecto se programa mediante una secuencia de instrucciones predefinidas que indican con claridad el cometido de estas.

Otra diferencia entre este proyecto y los PLC radica en la forma de tratar los programas. Normalmente, un PLC se conecta a un ordenador para ser programado, luego se desconecta y en su funcionamiento normal es totalmente autónomo, ejecutando continuamente el programa cargado en su memoria. En el caso del sistema implementado, el ordenador está en todo momento conectado al controlador y es el que

va enviando sucesivamente todas las órdenes que componen el programa. Por un lado, puede suponer una desventaja debido a que necesitaremos siempre un ordenador corriendo el programa para que el sistema funcione correctamente. Por otro, teniendo en cuenta que su fin no es un proceso industrial, sino experimentos que casi siempre serán diferentes, es de esperar que los programas se modifiquen con asiduidad, cosa que en este sistema será inmediata y se podrá hacer tan sólo parando la ejecución en curso, a diferencia de los PLC donde requerirá parar los sistemas, conectar ordenadores, etc.

Por último, cabe destacar que los PLC son sistemas bastante costosos y además suelen requerir de infraestructuras para su montaje que no están al alcance de grupos de investigación como el del laboratorio anteriormente mencionado. Sin embargo, el sistema resultante de este trabajo tan sólo requiere de un ordenador con la máquina virtual del Java y un montaje hardware cuyo valor no supera los cien euros para su funcionamiento. Por tanto, a pesar de sus limitaciones, puede resultar un sistema muy atractivo para lugares con presupuestos limitados.

4.3 Requisitos

Después de visitar el laboratorio que motivó este trabajo y luego de sucesivas reuniones con el tutor, se establecieron una serie de requisitos que definieron el sistema y que serán expuestos a continuación.

Lo primero que se estableció fue la estructura de comunicaciones entre el sistema central y el controlador programable. Se llegó a la conclusión de que estos debían conectarse a través del bus USB que proporciona la placa del controlador, obteniéndose un sistema como el del esquema siguiente:

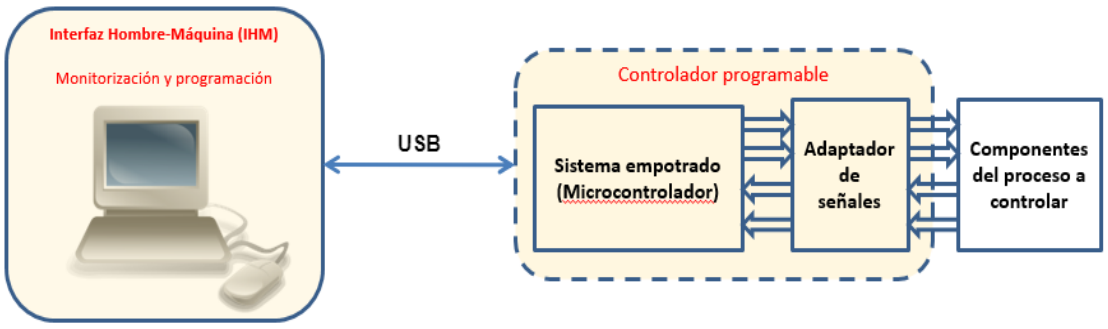

Ilustración 4: Esquema de la estructura del sistema

El sistema central es el que llevará la iniciativa en todas las operaciones que realice el controlador, siguiendo así un esquema similar al paradigma de maestro-esclavo.

El modo de actuación en las comunicaciones se ajustará a las siguientes etapas:

1. El sistema central envía un comando al controlador.

2. El controlador decodifica el comando.
3. Si el comando es válido se ejecutará, si no se enviará un error y abortará la ejecución.
4. Una vez ejecutado el comando se enviará una señal que indique su correcta ejecución y si fuese necesario se enviarían datos adicionales.

Cabe destacar que en caso de que los comandos sean de control de flujo, no se enviarán al controlador, sino que serán tratados directamente por el sistema central.

Una vez establecidas las bases de la comunicación entre los dos elementos principales que componen el sistema, se procedió a definir los diferentes comandos necesarios para poder obtener un sistema funcional. Se llegó a la conclusión de que se necesitarían los siguientes tipos de comandos:

- **Entrada/Salida binaria:** Todos los comandos que permitan manipular las entradas y salidas digitales del controlador, activándolas, desactivándolas o leyéndolas de diversas maneras.
- **Entrada/Salida analógica:** Comandos que permiten manipular las entradas estableciendo valores en el caso de las salidas o leyéndolos en caso de las entradas.
- **Reguladores:** Comandos que se activarán, desactivarán o cambiarán valores de consigna de reguladores de diversa índole como temperatura, presión, etc.
- **Control de flujo:** Comandos que no se enviarán al controlador y que dotarán de un mayor potencial a los programas que se elaboren en el sistema. Estos comandos deben permitir crear estructuras de código repetitivas, asignar valores a variables o crear y llamar subrutinas entre otras funcionalidades.

Para finalizar con este apartado, hay que señalar que también se establecieron ciertos requisitos a la hora de distribuir para su uso las entradas y salidas del controlador. En los siguientes apartados se explicará más en detalle la distribución elegida y el por qué de esta.

5. Desarrollo del proyecto

5.1 Visión general del desarrollo

Como se ha venido mencionando en los apartados anteriores de este documento, el proyecto consta de dos partes bien diferenciadas:

- Un controlador programable implementado con una placa Arduino Mega.
- Una aplicación de escritorio utilizada para programar el controlador.

Estas dos partes a su vez se interconectan mediante un bus USB para obtener el sistema final.

Debido a que estas dos partes pertenecen a tecnologías completamente diferentes, el desarrollo del proyecto se dividió en dos fases que fueron el desarrollo del software en Java para la aplicación de escritorio y el desarrollo del software en C para el controlador de Arduino así como su montaje hardware. Por ello en esta memoria dedicaremos un apartado a cada una de las fases del desarrollo.

5.2 Desarrollo de la aplicación de escritorio

5.2.1 Justificación del uso de Java

Como se ha explicado anteriormente, la aplicación de escritorio está desarrollada en el lenguaje de programación Java. El motivo por el cual se decidió hacer uso de este lenguaje es que con esto conseguimos una aplicación que se puede ejecutar en cualquier equipo. A diferencia de otros lenguajes de programación, los cuales deben ser compilados para la máquina en la que se van a ejecutar ya que corren directamente sobre esa arquitectura, los programas desarrollados para Java se ejecutan siempre sobre una máquina virtual, por lo tanto, el único requisito indispensable para ejecutar este programa es tener instalada en el equipo en cuestión la máquina virtual de Java. Con esto conseguiremos una aplicación multiplataforma que dotará de mayor flexibilidad al sistema final.

5.2.2 El entorno de desarrollo

Ya se ha mencionado que el entorno de programación utilizado para desarrollar la aplicación de java fue NetBeans, por tanto, ahora vamos a mencionar algunos aspectos importantes de este entorno.

Lo primero que se hace cuando se quiere comenzar a crear un programa es acceder a la sección de creación de nuevo proyecto como se ve en la imagen.

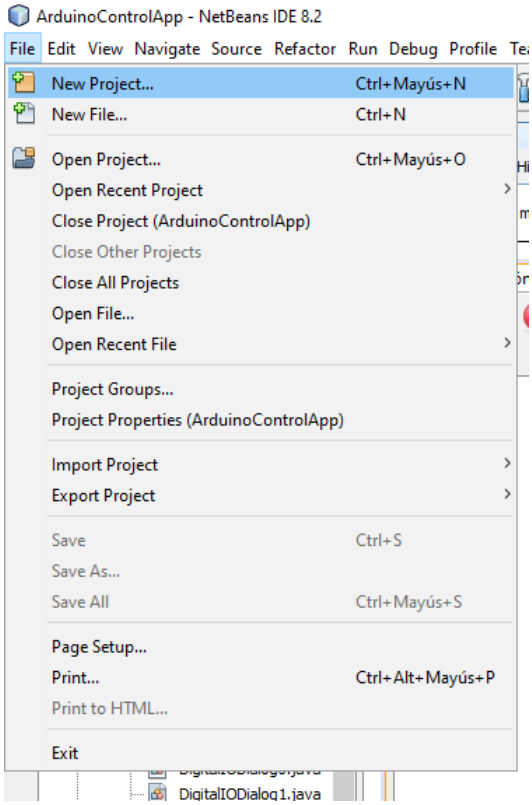

Ilustración 5: Creación de nuevo proyecto

Una vez se seleccione la creación de un nuevo proyecto, se desplegará un dialogo en el cual seleccionaremos el tipo de proyecto, el nombre y su localización entre otros parámetros.

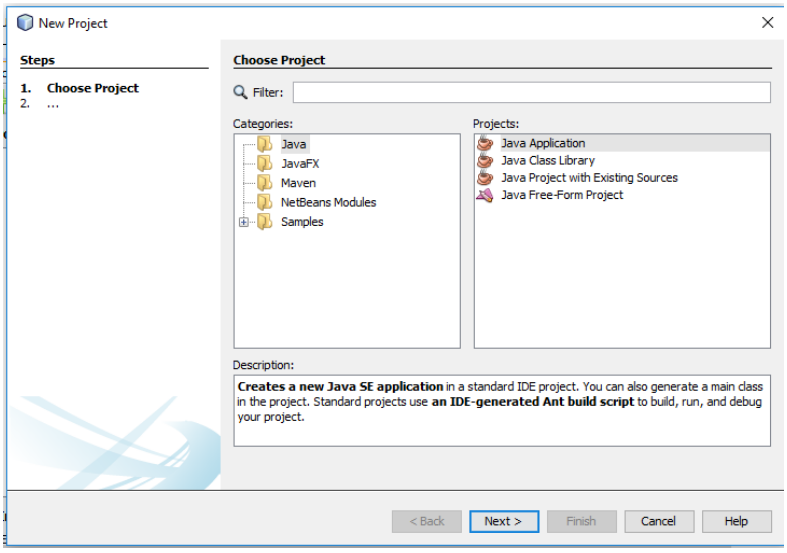

Ilustración 6: Dialogo de creación del proyecto 1

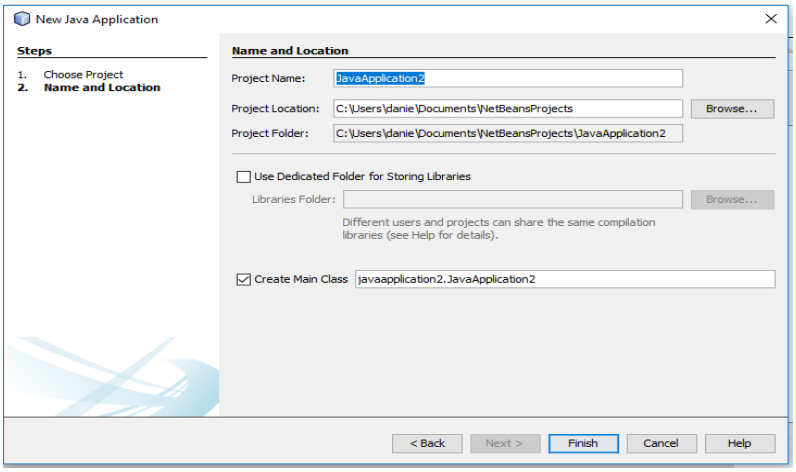

Ilustración 7: Dialogo de creación del proyecto 2

Una vez creado el proyecto, se nos mostrará una ventana compuesta de múltiples secciones que ayudarán en el desarrollo de este. Una de las más útiles es el árbol del proyecto. En él tendremos una visión jerárquica de los archivos que componen nuestro proyecto, además nos permite crear nuevos archivos, organizarlos en paquetes y añadir librerías entre otras cosas.

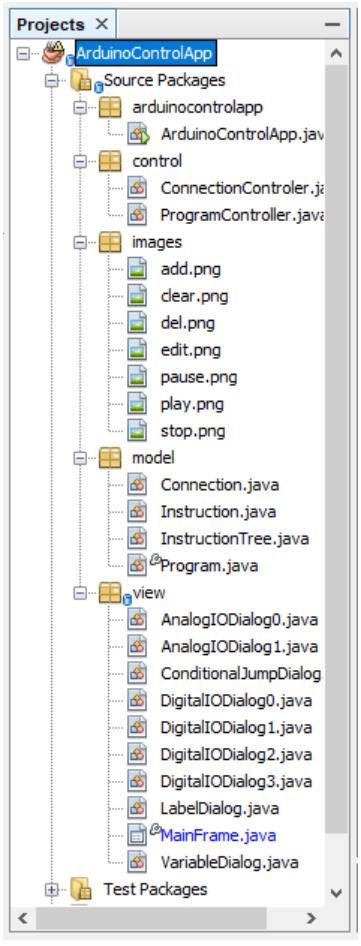

Ilustración 8: Árbol del proyecto

Como es de esperar, este entorno nos ofrece un campo de texto para escribir el código de nuestro proyecto, así como una sección que muestra la salida del programa mientras está en ejecución.

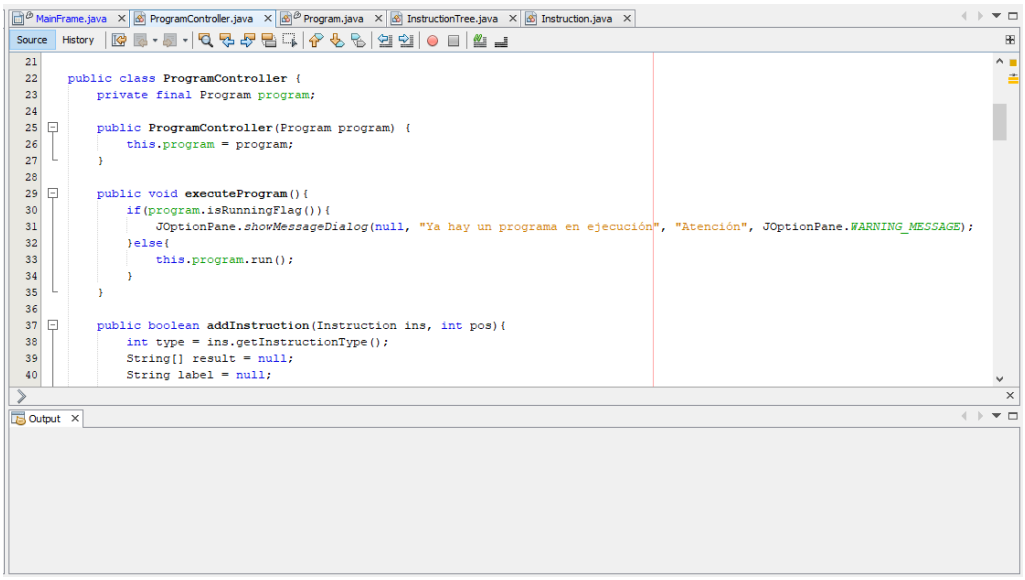

Ilustración 9: Campo de escritura y Salida de programa

Para finalizar, otra de las utilidades de NetBeans y uno de los motivos principales por el cual se eligió este entorno para el desarrollo de la aplicación es su utilidad para crear interfaces de usuario con Swing. En esta utilidad se muestra en todo momento una vista del estado actual de la interfaz, así como una paleta de componentes para añadir a la misma.

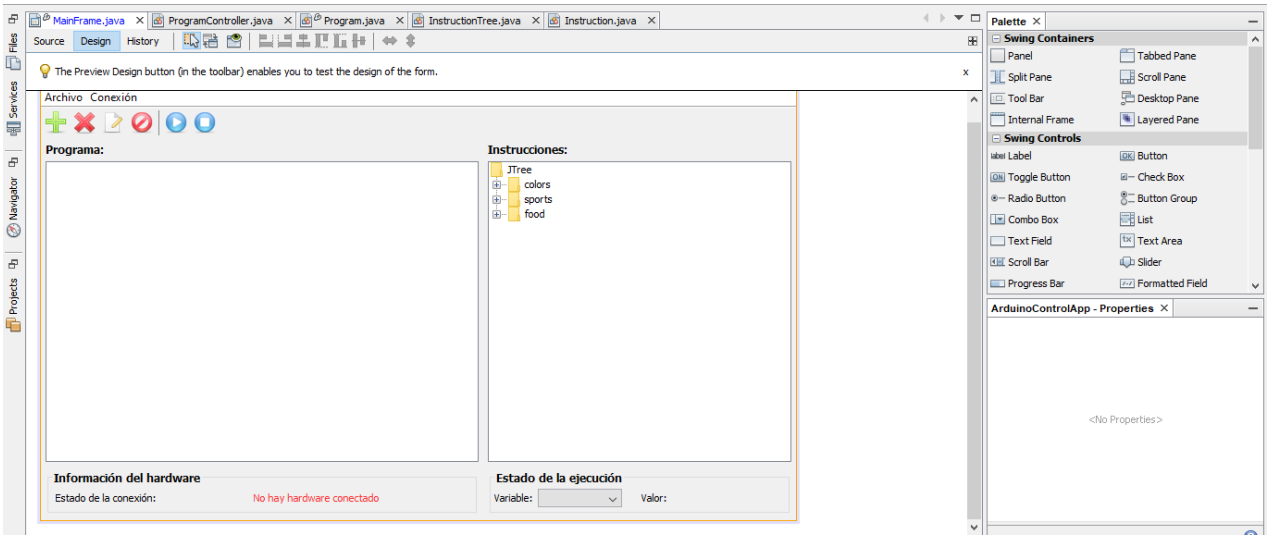

Ilustración 10: Utilidad para la creación de interfaces gráficas

5.2.3 Arquitectura del software

El diseño del programa se ha basado en la arquitectura modelo-vista-controlador (MVC), aunque tampoco se han seguido rigurosamente todas las recomendaciones de este patrón de diseño. En el siguiente esquema se puede ver el conjunto de clases que componen el programa y su disposición dentro del esquema MVC.

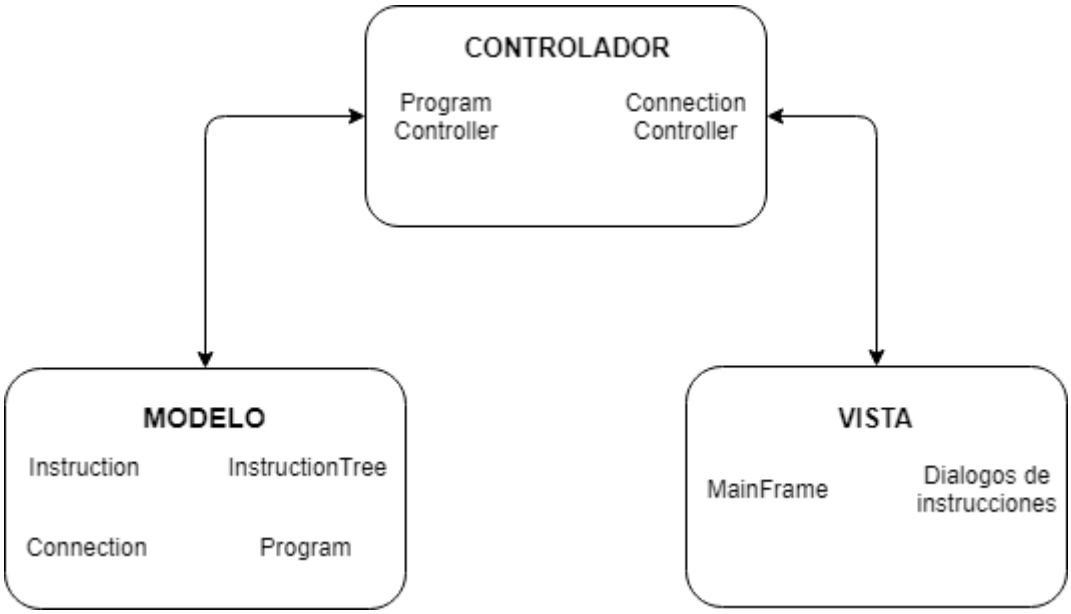

Ilustración 11: Esquema del MVC

Como se puede ver en el esquema, el controlador es el módulo del programa encargado de hacer de puente entre la vista, la cual engloba todas las clases orientadas a la interfaz de usuario, y el modelo, que contiene las clases que definen los datos que utiliza la aplicación. Hay que indicar que como se mencionó anteriormente, no se sigue del todo el patrón MVC ya que, aunque no esté plasmado en el esquema, hay un pequeño punto de unión entre el modelo y la vista que fue necesario realizar para obtener un código más simple. Este nexo se comentará y justificará más adelante en esta memoria.

5.2.4 Clases

Los programas desarrollados en Java están formados por clases, que son paquetes de código que nos permiten crear objetos. Estas clases deben crearse con cometidos claros y simples de modo que si las combinamos bien podamos realizar tareas más complejas con ellas.

Este proyecto está compuesto por una serie de clases que están distribuidas en diferentes paquetes dependiendo de su papel dentro del patrón MVC como indica la Figura 5.2.7. A continuación, pasaremos a desmenuzar el programa clase por clase viendo el papel que desempeña cada una dentro del funcionamiento global del sistema.

5.2.5 Clases del modelo

Puesto que son las clases que definen los datos en los que se basa el funcionamiento del programa comenzaremos con las clases asociadas al modelo del programa.

- **Connection:** Esta clase es la encargada de realizar todas las tareas relacionadas con la conexión a la placa Arduino. Su funcionamiento se basa en la librería de código libre RXTX. La librería RXTX está diseñada para simplificar la conexión con las placas Arduino, proporcionando clases como **SerialPort** y **CommPortIdentifier** las cuales permiten identificar y manipular los puertos a los que se conecta la placa Arduino y usarlos para enviar y recibir datos.

  Esta clase dispone de algunos métodos getter y setter los cuales no son relevantes como para comentarlos. A continuación, en el diagrama se muestran los atributos de la clase, así como sus métodos más importantes los cuales se explicarán luego.

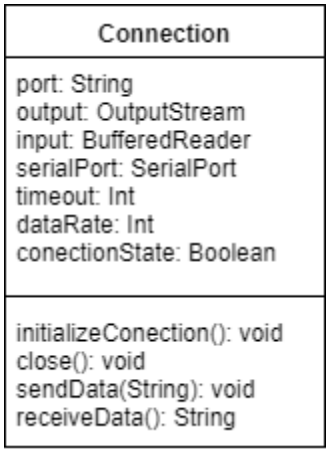

Ilustración 12: Diagrama de la clase Connection

Los métodos fundamentales de esta clase son:

InitializeConnection(): Este método toma el nombre del puerto que indique el atributo port (COM, ttyACM, …), comprueba que en el haya algún dispositivo conectado y si es así establece una conexión permanente entre el programa y la placa, permitiendo de este modo la comunicación entre ellos. Esta conexión la establece inicializando objetos como **SerialPort**, **BufferedReader** y **OutputStream**.

Close(): Es el encargado de cerrar la conexión entre la placa Arduino y el programa. Esto lo hace llamando a los métodos **close()** de las instancias de **SerialPort**, **BufferedReader** y **OutpuStream** inicializadas en el método **InitializeConnection()**. Este método es de gran importancia debido a que el programa no está preparado para la desconexión en caliente y en caso de hacerlo sin cerrar previamente la conexión se provocará un fallo que abortará la ejecución.

SendData(): Se encarga de enviar datos en forma de String a la Arduino por medio del **OutputStream**.

ReceiveData(): Lee los datos que lleguen de la Arduino en forma de String por medio del **BufferedReader**.

- **Instruction:** Es la clase que ofrece una estructura para la creación de las instrucciones, las cuales son los componentes que conformarán los programas que se envíen al controlador. Como en la anterior clase vamos primero a mostrar un esquema de su estructura para luego explicar los aspectos más importantes de la misma.

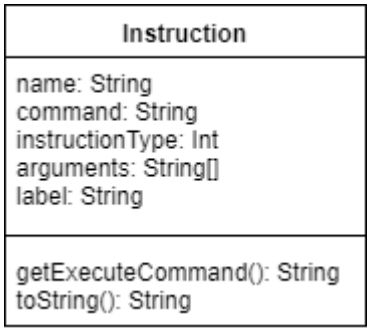

Ilustración 13: Diagrama de la clase Instruction

En esta clase además de los métodos, comentaremos algunos de los atributos de la clase que desempeñan un papel muy importante. Como ocurre con la anterior clase, hay más métodos de los que aparecen en el esquema, pero no se ha creído conveniente comentarlos todos.

En cuanto a los atributos de la clase:

Name: Será el nombre que se le dé a la instrucción para que el usuario sepa de que instrucción se trata.

Command: Es una String de dos letras mayúsculas precedidas por el símbolo del dólar. Esta String es la que le servirá al controlador para determinar el comando que debe ejecutar. Si las dos letras son XX, esto querrá decir que este comando no debe ser enviado a la Arduino, sino ser tratado directamente por el programa Java.

Arguments: Se trata de un array de String que almacena los diferentes parámetros que pueda tener una instrucción. El motivo de usar un array es que este número de parámetros puede variar de una instrucción a otra y además nos permitirá diferenciar más fácilmente cada parámetro.

InstructionType: Es un entero cuyo principal fin es diferenciar los distintos tipos de instrucciones para que a la hora de que el usuario añada una de ellas al programa se abra el dialogo adecuado para rellenar sus parámetros. También es importante para su uso en el método toString(), ya que dependiendo del número y tipo de parámetros la instrucción final puede tener diversos formatos.

Label: String utilizada en las instrucciones de control de flujo de etiquetas, subrutinas y saltos para facilitar la lógica de búsqueda de esas y mejorar así los tiempos de ejecución**.**

En cuanto a los métodos más importantes de la clase tenemos:

GetExecuteCommand(): Este método es vital para el funcionamiento del sistema, ya que devuelve una String que combina el atributo **command** con los elementos del array **arguments**, formando así el comando en su formato final que se le enviará al controlador para que lo ejecute.

ToString(): Esta redefinición del común método toString() se realizó porque era necesario que las instrucciones tuviesen un aspecto u otro según si tenía inicializado o no el array **arguments**. El motivo de esta necesidad radica en que mientras que a la hora de añadir una instrucción el usuario solo necesita saber cuál de ellas está añadiendo, una vez esta se ha añadido al programa debe mostrar toda la información relativa a sus parámetros. En la siguiente imagen vemos más claramente que mientras las instrucciones en que se encuentran en el árbol para su selección solo muestran su nombre, cuando se han añadido a un programa ya muestra los parámetros que el usuario les ha asignado.

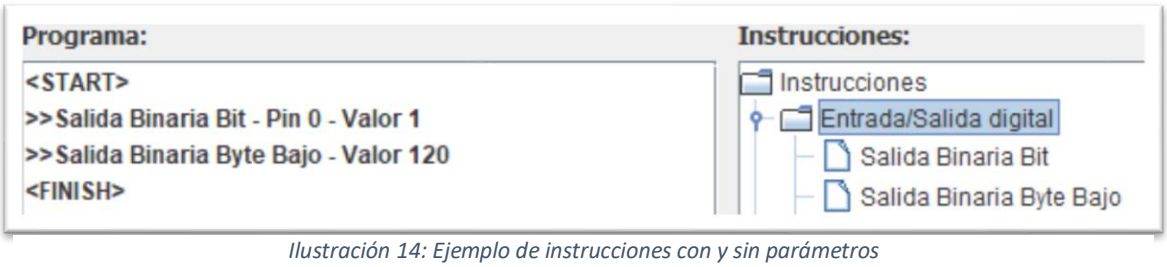

Ilustración 14: Ejemplo de instrucciones con y sin parámetros

- **InstructionTree:** Esta es una clase sencilla cuyo único cometido es proporcionar una estructura de árbol con todas las instrucciones disponibles en el sistema para que el usuario las añada a sus programas. Este es el aspecto que presenta en la interfaz de usuario dicho árbol.

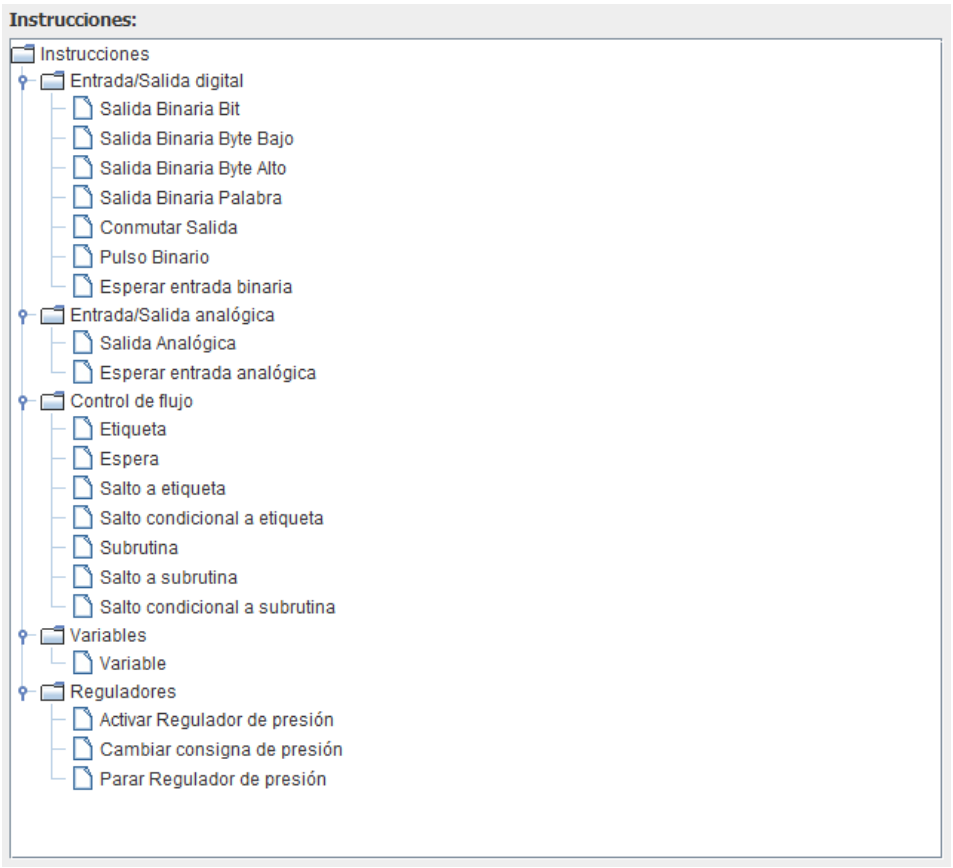

Ilustración 15: Árbol de instrucciones en la interfaz

Para generar este árbol la clase se vale de su único atributo y un método. Mostraremos un esquema como en las anteriores clases para tener una mejor visión y luego los explicaremos.

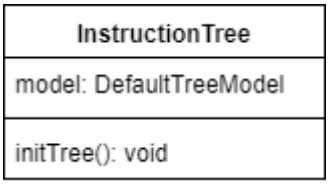

Ilustración 16: Diagrama de la clase InstructionTree

Model: Es un objeto del tipo **DefaultTreeModel**. Estos objetos están diseñados para asignárselos como modelo a los elementos gráficos **Jtree**, obteniendo así un árbol como el de la Ilustración 15. Este objeto supone uno de los puntos de conexión entre la vista y el modelo que se han mencionado anteriormente.

InitTree(): Es un método que se encarga de crear todas las instrucciones que vamos a tener disponibles por medio de la clase **Instruction** para luego insertar cada una de ellas dentro del **DefaultTreeModel**. Cabe destacar que todas estas instrucciones se crean sin el atributo **arguments** para que en el árbol sólo se muestre el nombre de la instrucción.

- **Program:** Es probablemente la clase más importante de todo el programa. Ofrece la estructura de datos para almacenar el programa y las variables que crea el usuario y también proporciona todos los métodos necesarios para ejecutar ese programa. De nuevo mostramos un diagrama con los métodos y atributos más importantes de la clase para luego dar las explicaciones pertinentes.

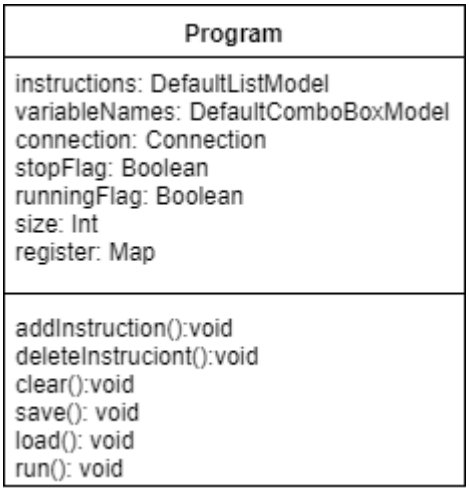

Ilustración 17: Diagrama de la clase Program

Instructions: Es uno de los puntos de conexión entre el modelo y la vista que se habían mencionado en un principio. El **DefaultListModel** es un objeto que permite mostrar sus datos directamente en el componente gráfico **Jlist**, además también ofrece la ventaja de que una vez vinculados, cada vez que se haga un cambio en el objeto, este se verá reflejado en la interfaz gráfica automáticamente sin necesidad de hacer operaciones extra. La lista resultante de esta estructura tiene este aspecto en la interfaz.

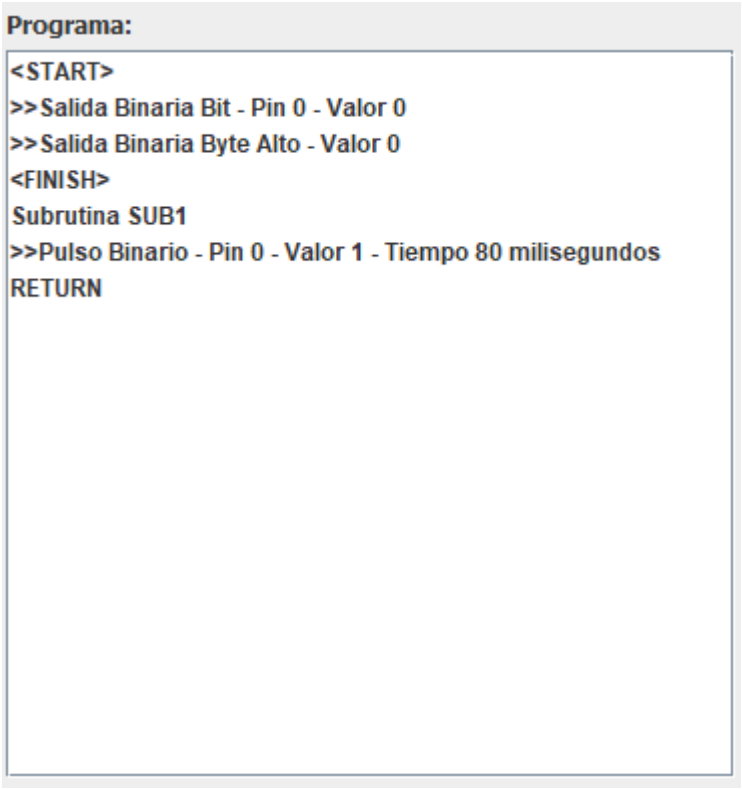

Ilustración 18: Lista del Programa

Además de servir para mostrar el programa al usuario, en el momento de la ejecución del programa en el controlador, las instrucciones se toman directamente de esta estructura.

VariableNames: Es el segundo y último punto de unión entre la vista y el modelo. Es un objeto del tipo **DefaultComboBoxModel** cuyo único objetivo es proporcionarle al usuario una lista desplegable en tiempo real para que pueda consultar los valores que tienen las variables del programa creado en ese momento. En la interfaz esta lista tiene el siguiente aspecto:

Ilustración 19: Lista de variables

Connection: Objeto de la clase **Connection** que utiliza para el envío de los comandos y recepción de respuestas del controlador durante la ejecución de los programas.

StopFlag: Booleano que indica al programa que se activa cuando un usuario desea parar la ejecución del programa creado.

RunningFlag: Booleano que indica que hay un programa ejecutándose en ese momento.

Register: Es un objeto del tipo **Map** cuya clave es una **String** y su valor un **Double**. Se utiliza para almacenar las variables creadas por el usuario ya que ofrece métodos que facilitan tanto su inserción como su búsqueda y no permitirá que se repitan variables con el mismo nombre.

AddInstruction(): Este método como su nombre indica, permite añadir instrucciones al programa que crea el usuario. Se encargará de tomar la instrucción nueva creada que recibe por como parámetro insertarla en **Instructions.** Si el usuario no ha seleccionado ninguna posición, la instrucción se insertará al por encima de *<FINISH>* y si la selecciona se insertará en esa posición desplazando la instrucción que la ocupaba hacia abajo. Además, si se añade una definición de subrutina, esta se colocará siempre debajo de *<FINISH>*.

DeleteInstruction(): Este método se encarga de borrar la instrucción de **Instructions** en la posición que indique el entero recibido por parámetro.

Clear(): Este método elimina todos los elementos de **Instructions** dejándola en su estado inicial.

Save() y Load(): Estos métodos se encargan de almacenar o cargar el programa contenido en **Instructions**. Para hacerlo, se emplea la librería de Google Gson, la cual es capaz de convertir un objeto en una String Json o viceversa.

Run(): Es el método encargado de ejecutar los programas que crea el usuario. Como durante la ejecución del programa no se quiere que la interfaz gráfica de la aplicación se quede congelada, este método lo que hace es lanzar un hilo que se que tomará el programa almacenado en **Instructions** y lo ejecutará del modo que se muestra en el diagrama de flujo de la Ilustración 20. Para explicar un poco lo que se ve en el diagrama, hay que decir que los programas siempre contienen dos instrucciones que nunca se ejecutan y son *<START>* y *<FINISH>*. Estas indican el punto de inicio y fin del programa. La existencia de estas instrucciones se debe a que a que las subrutinas se definen por debajo de *<FINISH>* y nunca se deben ejecutar a menos que sean llamadas desde el programa principal. Para terminar con esta sección sólo queda indicar que el modo de ejecución es un bucle que recorre la lista de instrucciones que compone el programa, y las instrucciones de control de flujo tan sólo manipulan el índice del bucle para modificar el flujo de la ejecución.

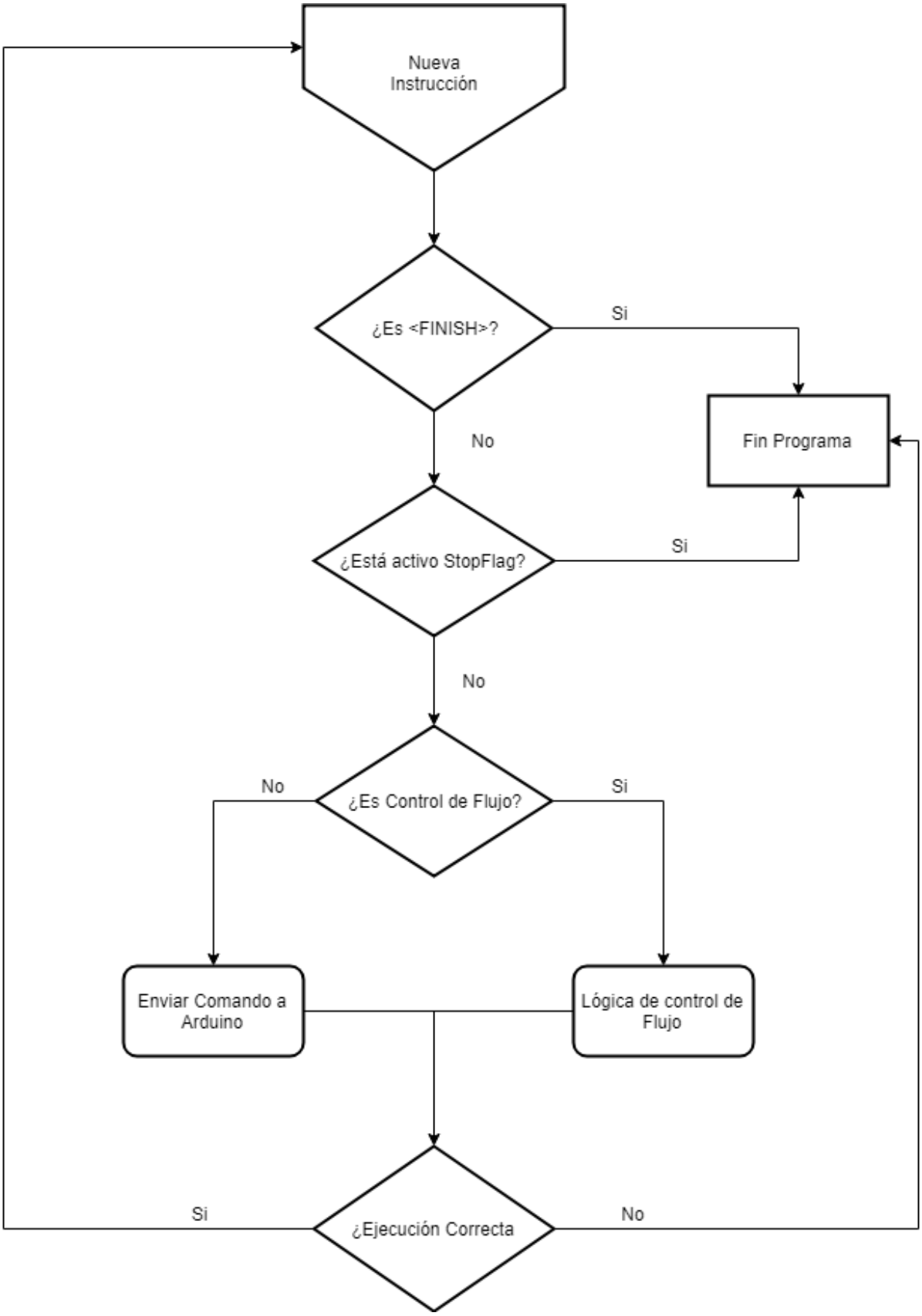

Ilustración 20: Diagrama de la ejecución de un programa

5.2.6 Clases del controlador

Ahora pasaremos a ver las clases del controlador, que son las encargadas de establecer un puente de unión entre los datos del modelo y la vista. El controlador tan sólo dispone de dos clases y son bastante simples.

- **ConnectionController:** Esta clase proporciona los métodos necesarios para que el usuario pueda establecer y cerrar la conexión con la Arduino mediante los eventos de la interfaz gráfica. A continuación, veremos su esquema y la explicaremos algo más en detalle.

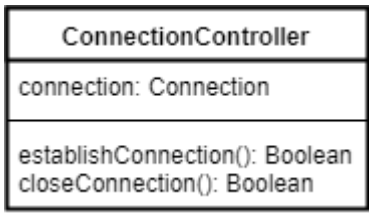

Ilustración 21: Diagrama de la clase ConnectionController

Connection: Es el mismo objeto **Connection** que utiliza la clase **Program** para enviar los comandos a la placa Arduino.

EstablishConnection(): Es el método que se llama cuando el usuario intenta establecer una conexión con la Arduino por medio de la interfaz gráfica. Cuando se le llama, despliega un dialogo en el que el usuario debe indicar el puerto en el que se encuentra la Arduino, luego llama al método **InitializeConnection()** de **Connection**. Por último, comprueba que la conexión se haya realizado de forma correcta enviando un mensaje a la Arduino y esperando por su respuesta. Si la conexión se ha realizado con éxito devolverá el valor true, en caso contrario devolverá false.

CloseConnection(): Método al que se llama cuando el usuario trata de cerrar la conexión con la Arduino por medio de la interfaz gráfica. Lo que hace este método es llamar al método **Close()** de **Connection** y devuelve true en caso de que el cierre de la conexión vaya bien.

- **ProgramController:** Esta clase ofrece los métodos que se necesitan para poder manipular todos los aspectos relacionados con la creación y ejecución de los programas cuando el usuario de las órdenes a través de la interfaz gráfica. Como

siempre, mostramos el esquema de su estructura y luego pasamos a explicarlo más en detalle.

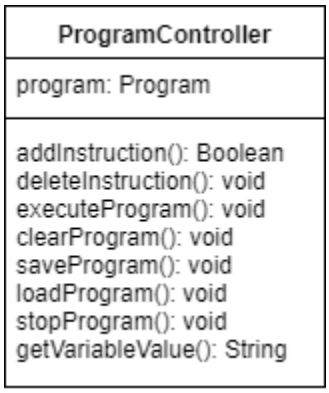

Ilustración 22: Diagrama de la clase ProgramController

Program: Es la única instancia de esta clase que se utiliza en la ejecución de la aplicación. La usa para poderla manipular con los eventos que provengan de la interfaz gráfica.

AddInstruction(): Es el método más complejo de toda la clase. Su cometido es tomar la instrucción y dependiendo del tipo de instrucción que sea desplegar el diálogo adecuado para que el usuario pueda agregar los argumentos necesarios. Una vez obtiene los datos del diálogo crea una nueva instrucción con los atributos **name**, **command** e **instructionType** de la instrucción seleccionada en el árbol y le añade como **arguments** el array que proviene del dialogo relleno por el usuario. Si la inserción de la instrucción tiene éxito, el método devolverá true.

GetVariableValue(): Este se encarga de mostrar el valor de una variable cuando el usuario la selecciona en la lista desplegable.

DeleteInstruction(): Llama al método **deleteInstruction** del **program** con la posición que haya seleccionado el usuario.

LoadProgram() y SaveProgram(): Ambos llaman a un **JfileChooser**, que es la típica ventana que despliegan todos los programas para seleccionar un archivo ya sea para abrirlo o guardarlo. Luego en el caso de **LoadProgram()** toma el archivo de la ruta seleccionada y lo envía al método **Load** de **program**. En cambio, **SaveProgram()** crea un nuevo archivo en la ruta seleccionada y lo envía al método **Save()** de **program**.

ClearProgram(): Sólo llama a la función **clear()** de program cuando el usuario selecciona esa opción en la interfaz.

ExecuteProgram(): Llama al método run() de program cuando el usuario desea iniciar la secuenciación del programa.

StopProgram(): Convierte a true el atributo **stopFlag** de **program**, haciendo así que se pare la ejecución si hay un programa corriendo.

5.2.7 Clases de la vista

Puesto que en las clases de la vista hay una infinidad de atributos que representan los componentes gráficos y que todos los métodos que hay, que también son bastantes, se limitan a captar eventos y a llamar a los métodos de las clases de control, en este caso no comentaremos el código de las clases, sino que se mostrarán cada una de las ventanas que generan esas clases y se comentará su funcionalidad.

- **MainFrame:** Es la ventana principal del programa y tiene el aspecto que se ve en la siguiente imagen:

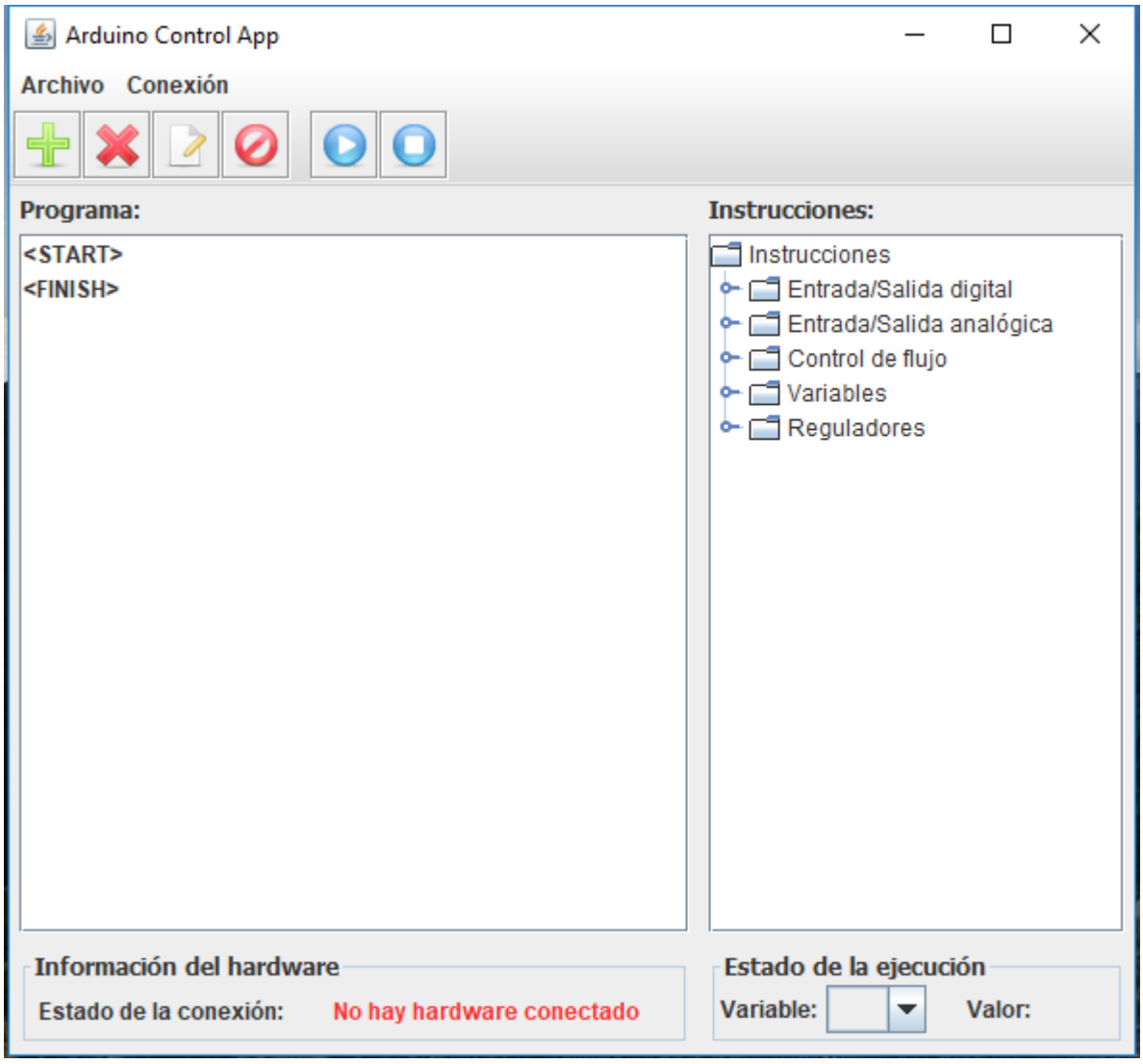

Ilustración 23: Ventana de la clase MainFrame

Como se puede observar, es una ventana bastante convencional. Está compuesta de una barra de menús, una barra de herramientas, una lista que contendrá el programa que realice el usuario, un árbol con las instrucciones disponibles para formar un programa y unos campos que nos muestran cierta información del sistema.

En la barra de menús hay dos menús, el de archivo y el de conexión. El menú de archivo nos permite guardar un programa en un archivo o cargar un programa desde un archivo.

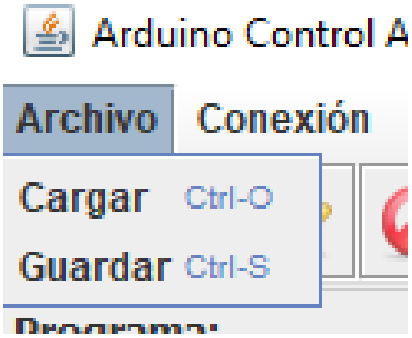

Ilustración 24: Menú de Archivo

Por otro lado, el menú de conexión nos permite tanto establecer una conexión con la placa Arduino como cerrarla.

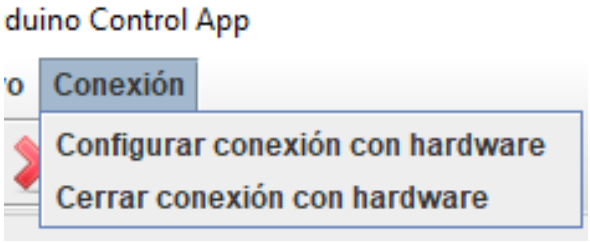

Ilustración 25: Menú de Conexión

La barra de herramientas nos proporciona los controles necesarios para editar el programa y para ejecutarlo.

Ilustración 26: Barra de Herramientas

El primer botón de izquierda a derecha nos permite añadir al programa la instrucción que se ha seleccionado en el árbol. El segundo borrará la instrucción del programa que se encuentre seleccionada. El tercer botón permitirá al usuario editar los valores de los argumentos de la instrucción seleccionada del programa. El cuarto cuando se acciona borra el programa actual por completo.

Como se puede observar, los dos últimos botones se encuentran separados de los cuatro anteriores, y es que estos botones no sirven para editar el programa. El cometido del primero es hacer que se comience a ejecutar el programa actual y el del segundo es parar la ejecución en curso del programa.

Las secciones de Programa e Instrucciones ya se comentaron cuando se habló de las clases **Program** e **Instruction**. Su cometido es mostrar el programa actual y el repertorio de instrucciones que necesita el usuario para construir programas.

También se comentó en la clase **ProgramController** la sección de monitorización de variables, que permite que el usuario pueda ver el valor de una variable mientras el programa se ejecuta o cuando ha terminado su ejecución.

Por último, queda por mostrar la sección que muestra información del hardware, es tan sólo una etiqueta que nos indica si hay o no una conexión establecida con la placa Arduino.

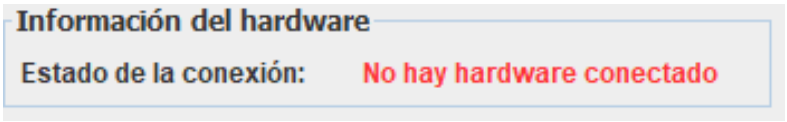

Ilustración 27: Información del hardware

- **DigitalIODialog0:** Este dialogo estaba pensado originalmente sólo para la instrucción de salida binaria, pero posteriormente se observó que también era útil para la instrucción de esperar por entrada binaria. Nos permite indicar el pin de entrada o salida y el valor de salida o valor de espera. El número de pines viene dado por un parámetro en el constructor de la clase, por tanto, se puede modificar y reutilizar en varios tipos de instrucción.

Ilustración 28: Dialogo DigitalIODialog0

- **DigitalIODialog1:** Dialogo utilizado para introducir un único valor numérico. Es de utilidad en las escrituras de bytes, introducir tiempos de espera y además también para introducir valores de consiga en la activación de reguladores. El número límite que se puede introducir viene dado por un parámetro del constructor de la clase, lo que lo confiere de potencial para ser usado en otras instrucciones nuevas que puedan aparecer.

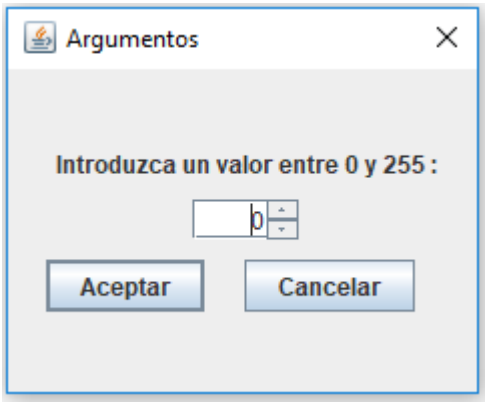

Ilustración 29: Dialogo DigitalIODialog1

- **DigitalIODialog2:** Este dialogo sólo se utiliza para la instrucción de conmutar salida y lo que nos permite es seleccionar el pin que se quiere conmutar.

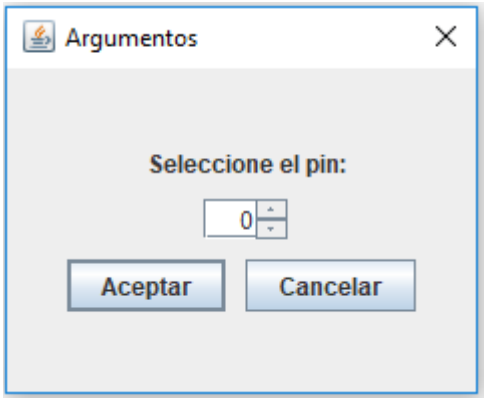

Ilustración 30: Dialogo DigitalIODialog2

- **DigitalIODialgo3:** Dialogo utilizado únicamente para la instrucción de pulso binario. Permite al usuario elegir el pin, el tipo de pulso (de encendido o de apagado) y por último permite elegir la duración del pulso que se encontrará entre 0 y 999 milisegundos.

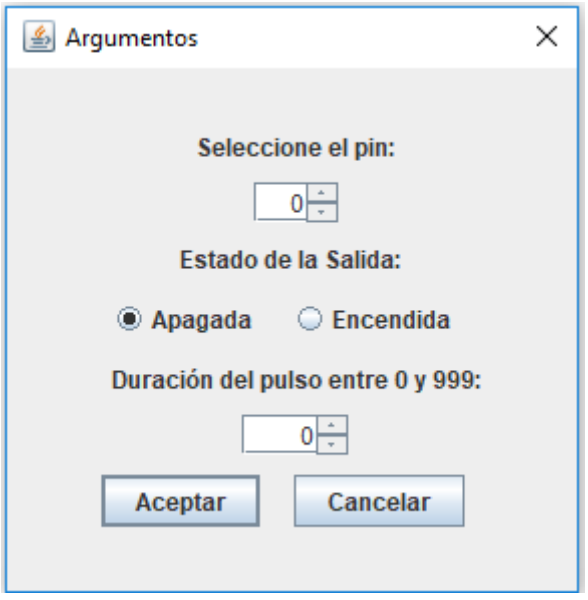

Ilustración 31: Dialogo DigitalIODialog3

- **AnalogIODialog0:** Se utiliza para la instrucción de salida analógica. Permite elegir el pin de salida y su valor que estará comprendido entre 0 y 255.

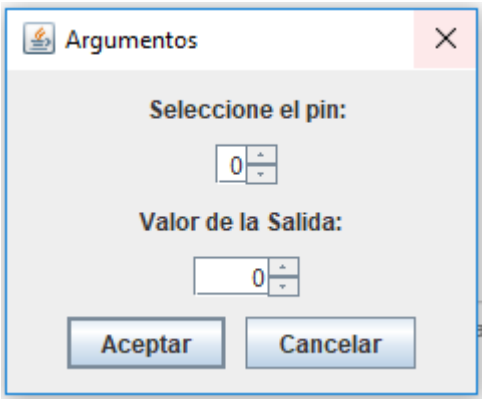

Ilustración 32: Dialogo AnalogIODialog0

- **AnalogIODialog1:** Sirve para las instrucciones de espera por entrada analógica. Nos permite seleccionar el pin, establecer un valor y elegir si la condición se cumplirá cuando el valor sea mayor o menor que el valor establecido por el usuario.

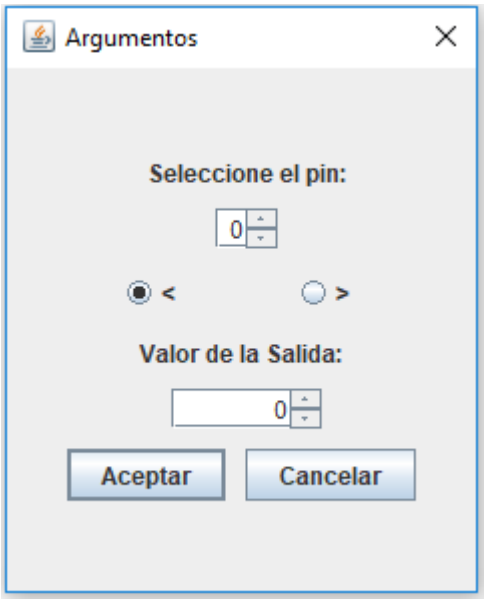

Ilustración 33: Dialogo AnalogIODialog1

- **LabelDialog:** Este dialogo solo ofrece un campo de texto. Primero se utilizó para definir la instrucción Etiqueta, pero luego también se aplicó a las definiciones de subrutinas y para los saltos incondicionales a subrutinas y etiquetas.

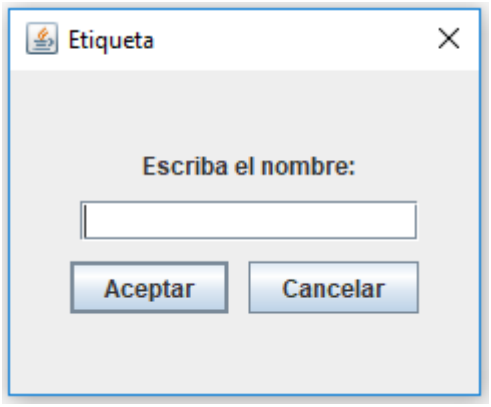

Ilustración 34: Dialogo LabelDialog

- **ConditionalJumpDialog:** Dialogo utilizado para definir un salto condicional, ya sea a etiquetas o a subrutinas. Permite escribir el nombre de la etiqueta o subrutina a saltar, el nombre de la variable a comparar, el valor de comparación y el tipo de comparación, mayor, menor o igual.

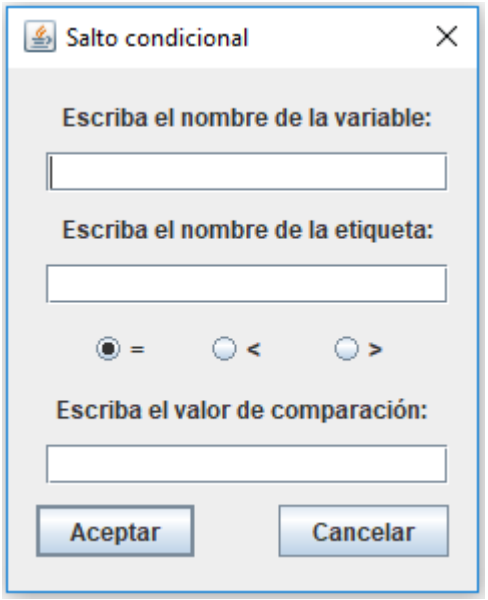

Ilustración 35: Dialogo ConditionalJumpDialog

- **VariableDialog:** Este es el último dialogo del programa. Permite al usuario definir una variable y de donde va a tomar esta su valor. El valor de la variable puede ser inmediato, es decir, lo introduce el usuario, puede provenir de un pin digital, de un pin analógico o de la lectura de un byte.

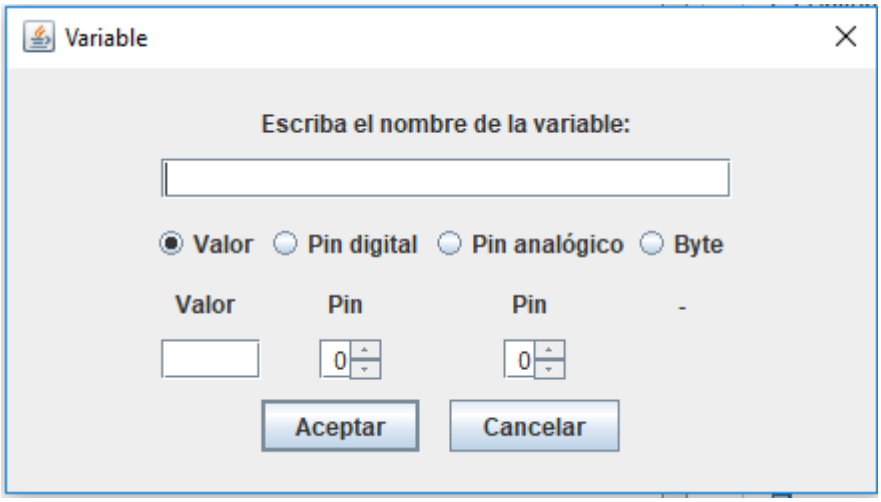

Ilustración 36: Dialogo VariableDialog

5.2.8 Repertorio de instrucciones

Llegados a este punto, ya ha sido explicada toda la estructura y parte del código que compone la aplicación de escritorio del sistema desarrollado. Para terminar con su explicación falta mencionar todas las instrucciones que forman parte del repertorio que ofrece la aplicación para que los usuarios puedan construir sus programas. A continuación, mostraremos unas tablas con los nombres de las instrucciones, su comando, los parámetros que contienen y que valores pueden tener esos parámetros.

Primero se mostrarán las instrucciones que se utilizarán para manejar las entradas y salidas digitales del sistema. Para que se entiendan mejor los rangos de valores en cuanto a salidas y entradas, hay que tener en cuenta que el sistema tiene configuradas para ser usadas de esta manera un total de 16 pines que funcionan como salidas digitales y 7 pines que funcionan como entradas digitales.

Entrada/Salida Digital		
Descripción	**Comando**	**Parámetros**
Salida Binaria Bit	$OB Salida, Valor	Salida: 0-15 Valor: 0-1
Salida Binaria Byte Bajo	$OL Valor	Valor: 0-255
Salida Binaria Byte Alto	$OH Valor	Valor: 0-255
Salida Binaria Palabra	$OW Valor	Valor: 0-65535
Conmutar Salida	$OT Salida	Salida: 0-15
Pulso Binario	$OP Salida, Tiempo, Valor	Salida: 0-15 Tiempo: 0-999 Valor: 0-1
Esperar Entrada Binaria	$WB Entrada, Valor	Entrada: 0-7 Valor: 0-1

Ilustración 37: Tabla de Instrucciones E/S Digital

Ahora veremos las instrucciones disponibles para el manejo de las entradas y salidas analógicas del sistema. En este caso el sistema dispone de 7 pines que actúan como salidas analógicas y 4 pines que actúan como entradas analógicas.

Entrada/Salida Analógica		
Descripción	**Comando**	**Parámetros**
Salida Analógica	$PW Salida, Valor	Salida: 0-7 Valor: 0-255
Esperar Entrada Analógica	$WA Entrada, Valor, Comparador	Entrada: 0-4 Valor: 0-1024 Comparador: <, >

Ilustración 38: Tabla de Instrucciones E/S Analógica

A continuación, se mostrarán las instrucciones de control de flujo. Hay que tener en cuenta que este tipo de instrucciones no se envía a la Arduino para tratarlas, sino que se procesan directamente en el hilo de ejecución del programa de Java. Por ello, todos los comandos de este grupo de instrucciones son $XX, aunque si es interesante observar sus parámetros.

Control de Flujo		
Descripción	**Comando**	**Parámetros**
Etiqueta	$XX Nombre	Nombre: texto
Espera	$XX Milisegundos	Milisegundos: 0-100000
Salto a etiqueta	$XX Nombre	Nombre: texto
Salto condicional a etiqueta	$XX Nombre, Variable, Valor, Comparador	Nombre: texto Variable: texto Valor: número Comparador: <, >, =
Subrutina	$XX Nombre	Nombre: texto
Salto a subrutina	$XX Nombre	Nombre: texto
Salto condicional a subrutina	$XX Nombre, Variable, Valor, Comparador	Nombre: texto Variable: texto Valor: número Comparador: <, >, =

Ilustración 39: Tabla de Instrucciones Control de Flujo

Para instrucciones del tipo variables como tal sólo existe una, pero la variable puede tomar su valor por distintas vías como se ha mencionado a lo largo de esta memoria. Para obtener ciertos valores como los de entrada digital, analógica y byte se precisa que se envíe un comando a la Arduino para efectuar esa lectura. En la tabla mostraremos esos comandos indicando para que tipo de variable son.

Variables		
Descripción	**Comando**	**Parámetros**
Variable	$XX Nombre, Tipo, Valor	Nombre: texto Tipo: Inmediato, Binario, Analógico, Byte Valor: -
Inmediato	No lleva comando	Valor: número
Binario	$IB Entrada	Entrada: 0-7 Valor: 0-1
Analógico	$AI Entrada	Entrada: 0-4 Valor: 0-1024
Byte	$IL	Valor: 0-255

Ilustración 40: Tabla de Instrucciones Variables

Para terminar, quedan las instrucciones de manipulación de los reguladores. En este caso sólo se ha implementado un regulador que es el de presión, así que hay que tener en cuenta que los valores de consigna que se indiquen serán en kilo Pascales. También hay que señalar que el primer parámetro del comando identifica al tipo de regulador, por tanto, es fijo y el usuario no lo podrá modificar. También el segundo valor es fijo porque indica el tipo de acción sobre el regulador. Al igual que el anterior, este tampoco se puede modificar

Reguladores		
Descripción	**Comando**	**Parámetros**
Activar regulador de presión	$RP 0 0 Valor	Valor: 0-100000
Cambiar consigna de presión	$RP 0 1 Valor	Valor: 0-100000
Desactivar regulador de presión	$RP 0 2	

Ilustración 41: Tabla de Instrucciones Reguladores

5.3 Desarrollo del Controlador

Como se lleva comentando a lo largo de toda la memoria, el controlador hardware se ha implementado con una Arduino Mega 2560 y varios componentes electrónicos. En esta sección explicaremos el montaje Hardware que se ha llevado a cabo, así como el software que se ha cargado en la placa Arduino.

5.3.1 Configuración del hardware

Lo primero que se hizo a la hora de configurar el hardware fue planificar la distribución de los pines disponibles de Arduino para las distintas necesidades que había como pines analógicos, digitales, etc. En un esquema veremos su distribución y luego se justificará.

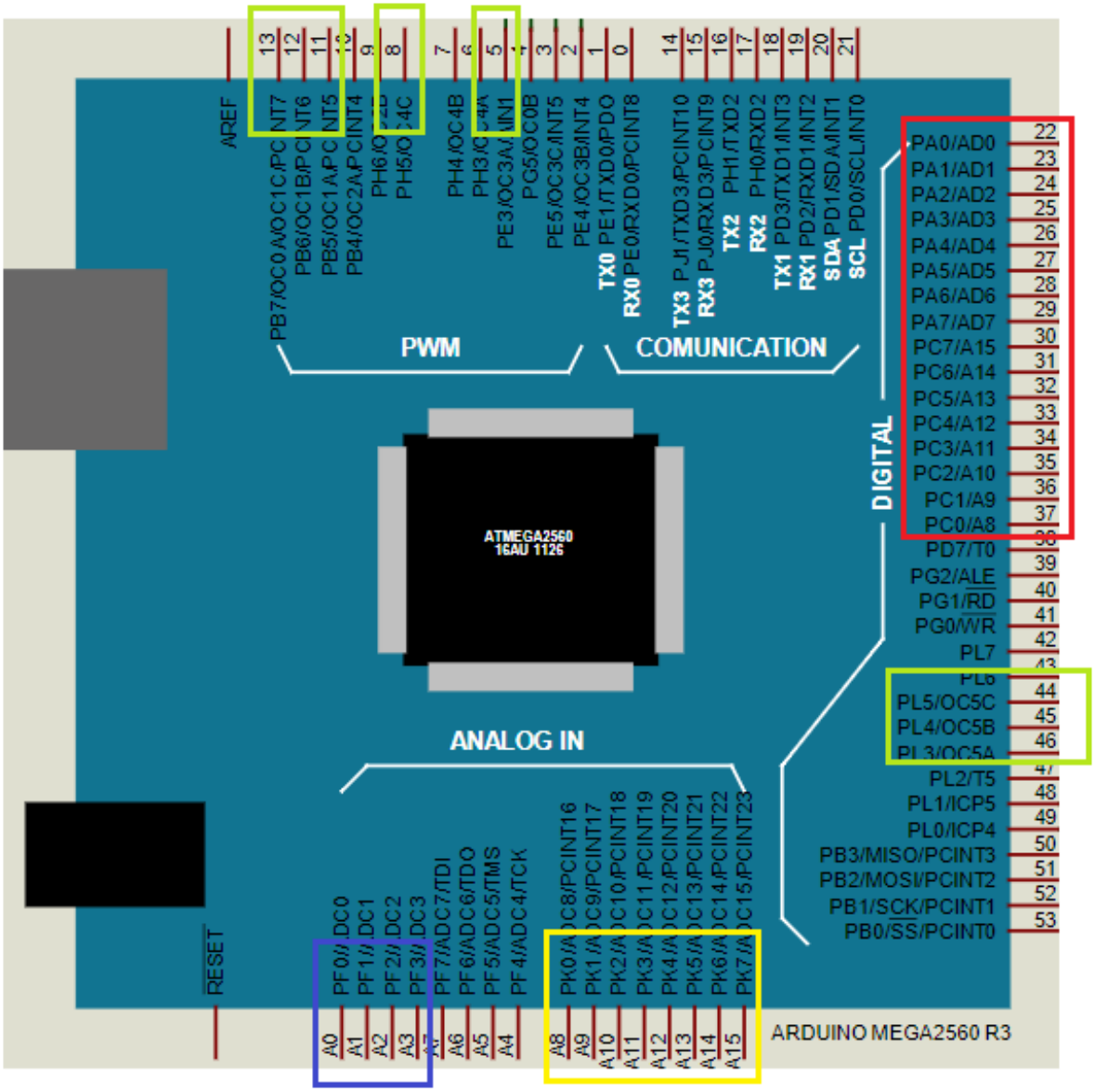

Ilustración 42: Esquema de distribución de pines

En la figura anterior se pueden ver todos los pines que posee la placa Arduino mega y también que algunos de ellos están rodeados por cuadros de distintos colores, que significan lo siguiente:

- **Rojo:** Son los pines empleados para las salidas digitales. Pertenecen a los puertos A y C del microcontrolador y por ese motivo han sido seleccionados. Al ser dos puertos completos se pueden inicializar los 16 pines con dos únicas instrucciones de Arduino DDRA y DDRC ahorrando así una gran cantidad de código. También esto facilita las instrucciones de escrituras de bytes y palabras, dado que se puede escribir a la vez todos los pines de los puertos con las instrucciones PORTA y PORTC.
- **Amarillo:** Estos pines corresponden a las entradas digitales. Pertenecen al puerto K y a pesar de que están marcadas como entradas analógicas en la placa, también se pueden usar en modo digital. Al igual que antes como todos los pines pertenecen al mismo puerto se pueden inicializar con una sola instrucción. Además, esto facilita las instrucciones de lecturas de bytes puesto que con la instrucción PINK se pueden leer todos los pines a la vez, obteniendo un valor entre 0 y 255 directamente.
- **Verde:** Son los pines de las salidas analógicas. Son 8 en total, pero en este caso no se encuentran todos en el mismo puerto. Esto se debe a que se han querido dejar ciertos pines libres porque tienen funciones de Timers o Interrupciones que podrían ser de utilidad, tanto para las instrucciones de reguladores como para posibles ampliaciones futuras del proyecto.
- **Azul:** Son los pines correspondientes a las entradas analógicas. En este caso son sólo 4 porque se ha considerado que no es necesario un tener un gran número de entradas analógicas disponibles y es posible que sean más útiles para poder añadirle al sistema utensilios como sensores más adelante.

Cabe destacar por último que todos los pines orientados a comunicaciones serie se han dejado libres por si en un futuro fuese necesario añadir al sistema dispositivos que para enviar datos precisen de una comunicación serie.

Para hacer pruebas de funcionamiento mientras se desarrollaba el sistema, se conectaron los módulos de relés opto aislados a las salidas digitales de la placa ya que probablemente ese sea siempre su uso más común. También se colocaron pulsadores en las entradas digitales para simular entradas de pulsos y ver que efectivamente se podían realizar lecturas de estos. Para probar las entradas analógicas, estas se conectaron a diferentes resistencias para averiguar si realmente oscilaban los valores de lectura. Por último, para comprobar las salidas analógicas se conectaron algunas a un osciloscopio para observar si generaban correctamente la señal PWM indicada. Con esto, quedó una configuración del hardware bastante similar a la que se puede ver en el siguiente esquema.

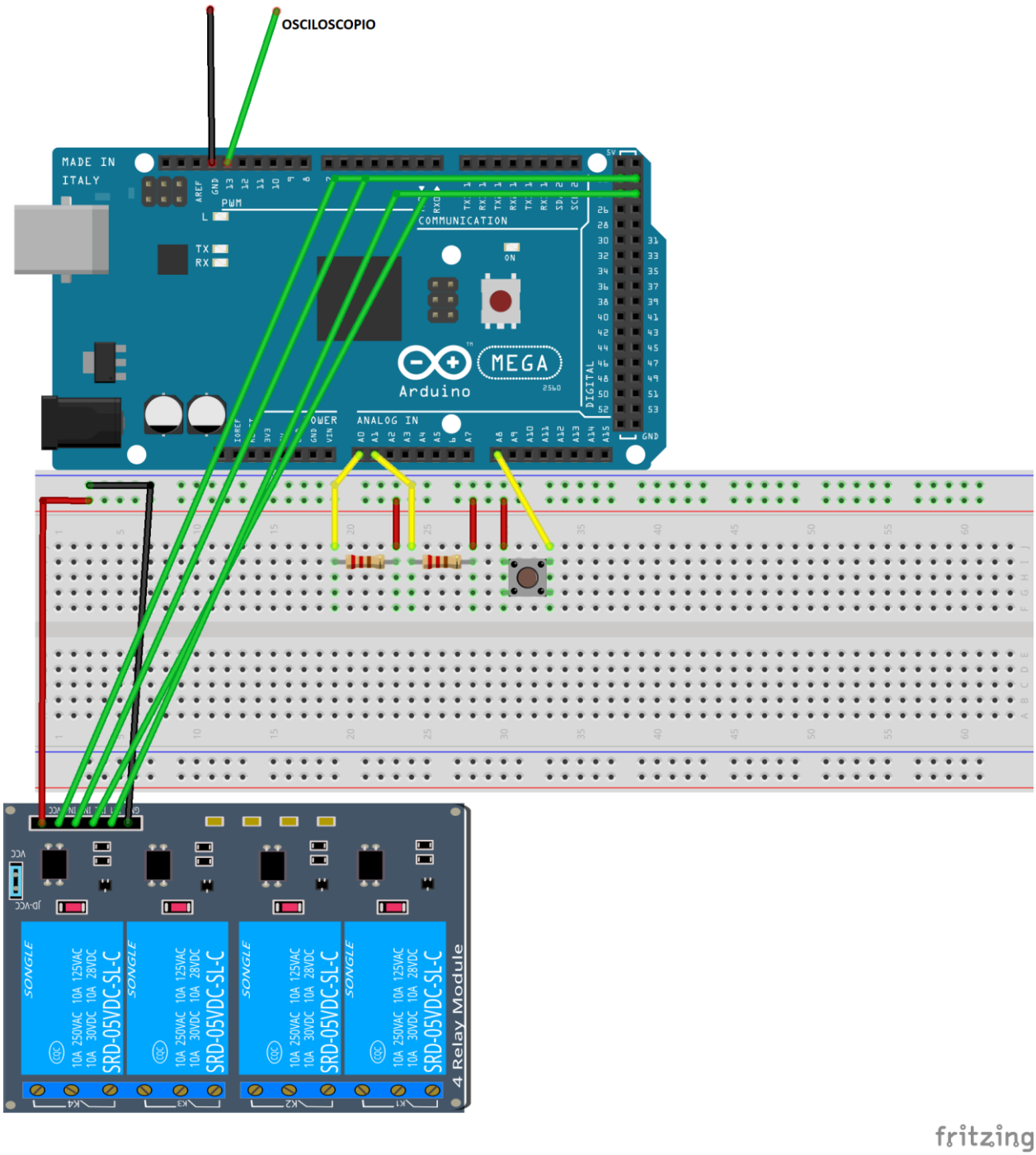

Ilustración 43: Esquema de Conexiones de Arduino

5.3.2 Entorno de desarrollo

Como se ha venido mencionando, el software de Arduino se ha desarrollado con el software Arduino IDE. Hay otras formas de generar códigos para Arduino, pero se ha elegido esta porque es la manera más sencilla que hay de crear el código, importar librerías, compilar y cargar el código en la placa. Este entorno está basado en Processing

y usa la estructura del lenguaje de programación Wiring. El aspecto que presenta el programa es el siguiente.

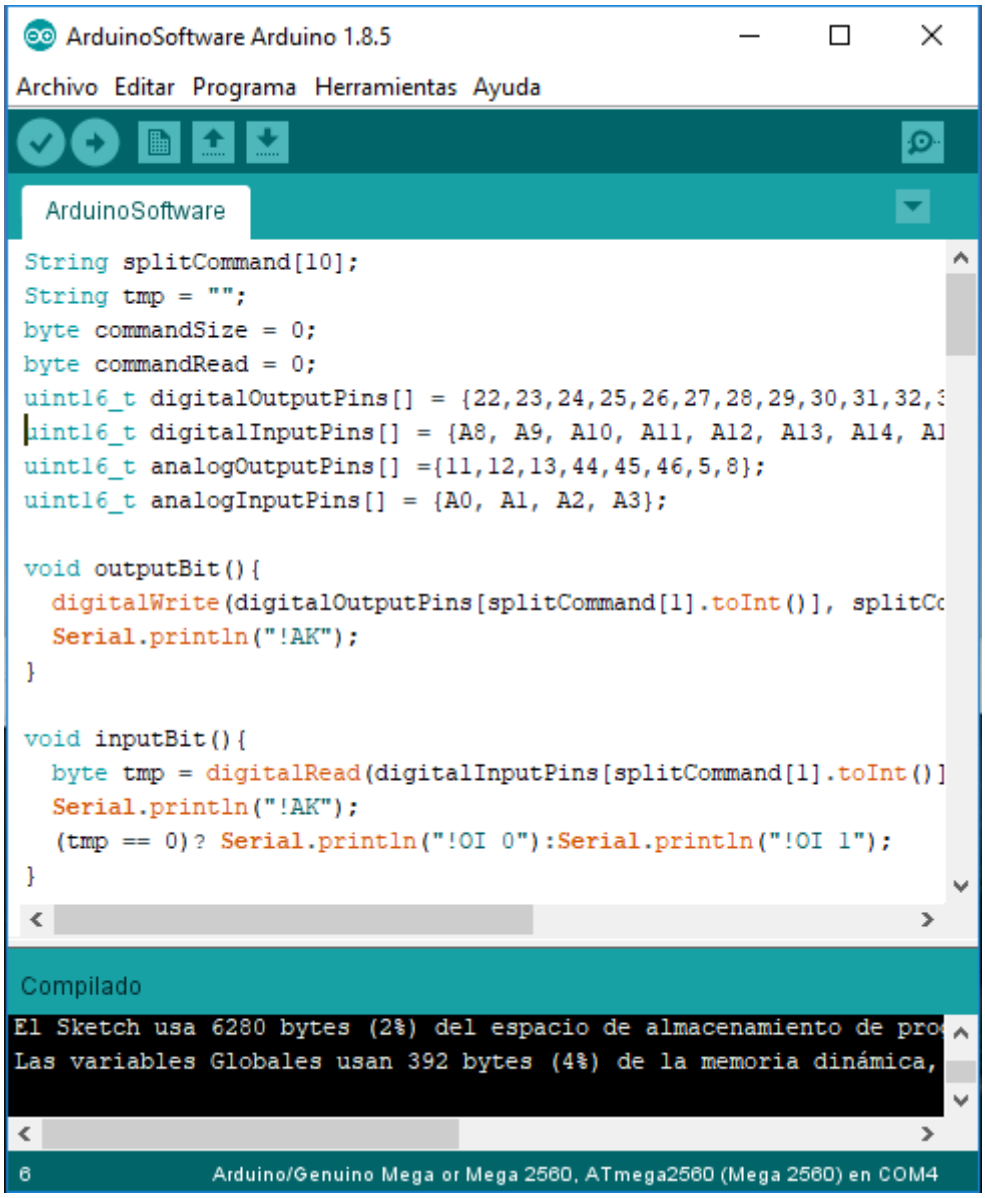

Ilustración 44: Arduino IDE

Como se puede ver, el entorno ofrece un campo para escribir todo el código del programa que se desea cargar en la placa, así como una salida que nos muestra el estado de la compilación y de la carga del programa, siendo de gran ayuda a la hora de detectar errores. Por último, en la parte de debajo de la venta se muestra información relativa al tipo de placa para el que se compila el código y el puerto del ordenador al que se encuentra conectada la placa. Hay que destacar que el usuario es el que debe elegir dentro del menú la placa para la que se quiere compilar y el puerto, ya que en caso contrario no se podrá compilar y cargar correctamente el código.

5.3.3 Desarrollo del Código

Una vez el hardware ya estuvo configurado se pasó a desarrollar el software que se ejecutará en la placa Arduino. Ya se ha mencionado en otras ocasiones que el sistema total funciona bajo el paradigma maestro-esclavo y que la Arduino desempeñaría el papel del esclavo. Antes de entrar en profundidad en el código desarrollado, es interesante ver en el siguiente diagrama cual debe ser el comportamiento de la Arduino en todo momento.

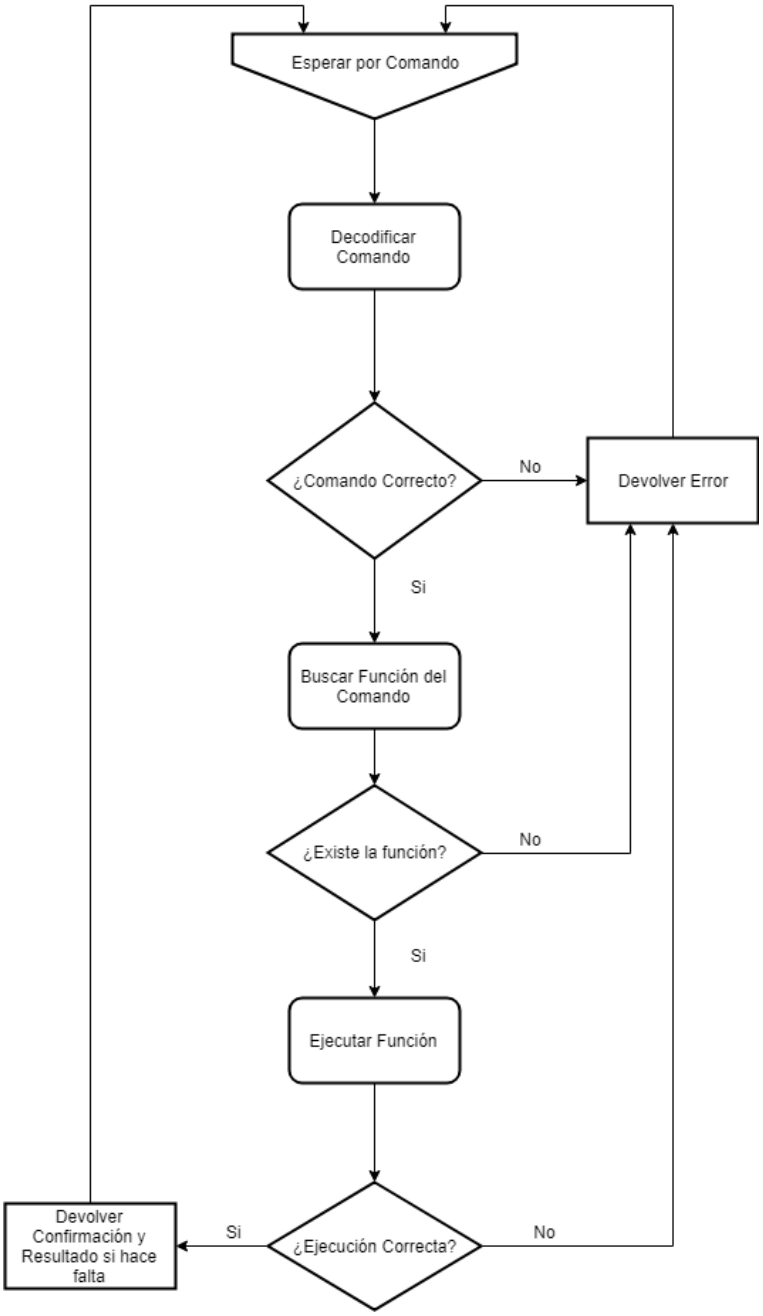

Ilustración 45: Diagrama de comportamiento de la Arduino

Una vez se ha visto cual es el comportamiento del software de Arduino, es más sencillo explicar el código y su funcionalidad que iremos desglosando por partes ahora.

- **Variables globales:** En el programa hay 8 variables globales que son:
  - **splitCommand:** Es un array de String, albergará el comando recibido después de ser decodificado, con el nombre del comando en la primera posición, que serán dos letras mayúsculas y en el resto de posiciones los argumentos del comando.
  - **tmp:** Es una String utilizada en el momento de decodificar el comando, en su momento se explicará mejor su uso.
  - **commandSize:** Entero que indica el número de elementos que compone el comando. Se explicará mejor su uso en más adelante.
  - **commandRead:** Variable que nos indica que se está decodificando un comando. Al igual que en anteriores su uso se explicará más adelante.
  - **digitalOutputPins:** Array de enteros que guarda el número de cada uno de los pines de salida digital. Esto nos permite numerar en el programa de Java los pines del 0 al 15. Luego este número lo usamos de índice para acceder al array en el momento en el que se ejecuta el comando haciendo así un mapeo interno y accediendo al pin correcto de forma transparente para el usuario.
  - **digitalInputPins:** Array de enteros que guarda el número de cada uno de los pines de entrada digital. Misma utilidad que en el caso anterior.
  - **analogOutputPins:** Array de enteros que guarda el número de cada uno de los pines de salida analógica. Misma utilidad que en el caso anterior.
  - **analogInputPins:** Array de enteros que guarda el número de cada uno de los pines de entrada analógica. Misma utilidad que en el caso anterior.
  - **pressure:** Entero que se utiliza para almacenar el valor de consigna del regulador de presión.
- **Setup():** Este método se ejecuta en el momento de arrancar la placa. Su tarea consiste en inicializar los pines como entrada o salida mediante las operaciones DDR, establecer las salidas por defecto a 0 con operaciones PORT e inicializar el puerto serie a 9600 baudios para las comunicaciones con la aplicación de escritorio. Además, se encarga de inicializar los registros del Timer 3 para que cuando se habiliten las interrupciones, genere una cada dos segundos.

- **Loop():** Este método está continuamente ejecutándose en bucle y se aprovecha para decodificar los comandos provenientes de la aplicación Java. Puesto que el método es algo complejo se mostrará una captura de su código para tener una mejor perspectiva y luego se comentará su funcionalidad.

```
void loop() {
  if(Serial.available()){
    char c = Serial.read();
    if(commandRead){
      if(c == '\n'){
        splitCommand[commandSize] = tmp;
        tmp = "";
        commandSize++;
        executeCommand();
        commandRead = 0;
        commandSize = 0;
      }else if(c == ' '){
        splitCommand[commandSize] = tmp;
        tmp = "";
        commandSize++;
      }else{
        tmp += c;
      }
    }else if(c == '$'){
      commandRead = 1;
    }else{
      Serial.println("!NK");
    }
  }
}
```

Ilustración 46: Código del método Loop()

En cada iteración, el método consulta el buffer de entrada para ver si hay datos disponibles. Si hay datos, se extrae un carácter y si **commandRead** es 0 querrá decir que actualmente no se está leyendo ningún comando. Luego se comprueba que el primer carácter sea $, que marca el inicio de un comando. Si lo es **commandRead** tomará el valor 1, si no se devolverá un error. Cuando **commandRead** es 1 en la variable **tmp** se van concatenando los caracteres leídos en cada iteración hasta que se encuentre o bien un espacio o un salto de línea. Si se encuentra un espacio, **tmp** se almacena en la posición del array **splitCommand** indicada por la variable **commandSize**, luego se incrementa **commandSize** y a **tmp** se le asigna una String vacía. Si se encuentra un salto de línea querrá decir que se ha alcanzado el final del comando y se hará lo mismo que en el caso del espacio, además se llama al método **executeCommand()** y una vez este método termine **commandRead** y **commandSize** se igualarán a 0, dejando el sistema listo para recibir un nuevo comando.

- **ExecuteCommand():** Este método se encarga de llamar a la función adecuada dependiendo del comando. Lo hace tomando la primera posición de **splitCommand**, la cual contiene el identificador del comando que son dos letras mayúsculas. Luego compara esa String con todas las opciones posibles y llama a la función que se encuentre en la sentencia if donde ocurra la coincidencia. Si no hay coincidencia se devolverá un error.

- **OutputBit():** Esta función es la utilizada para las instrucciones de salidas digitales. Hace uso de la función **digitalWrite()** de Arduino. Esta función requiere el número del pin y el valor a escribir. El número del pin lo obtiene accediendo a la posición de **digitaOutputPins** que se inidica en **splitCommand[1]**, y el valor de la escritura es directamente el valor de **splitCommand[2]**. Hay que tener en cuenta que los valores almacenados en **splitCommand** son Strings, por lo que antes de hacer las asignaciones se deben convertir a entero. Al final devuelve la confirmación de ejecución correcta a la aplicación de escritorio.

- **InputBit():** Se utiliza para leer el valor de una entrada digital. Hace uso de la función **digitalRead()** de Arduino. Esta función requiere el número del pin que se desea leer, y este lo proporciona la posición de **digitalInputPins** indicada por **splitCommand[1]**. La lectura del pin la almacena en una variable, luego envía la confirmación de ejecución correcta y para terminar, envía a la aplicación de escritorio el valor de la lectura precedida de ***!OI***.

- **OutputToggle():** Es bastante similar a **outputBit()**. Su única diferencia es que no toma el valor de escritura de un parámetro del comando, sino que lee el valor que tiene el pin en ese momento y lo niega, haciendo que el valor de salida de ese pin sea el contrario.

- **InputAnalog():** Es prácticamente igual a **inputBit()**. Las únicas diferencias entre estos dos métodos es que en vez de usar **digitalRead()**, esta función usa **analogRead()** y además en vez de tomar el valor del pin del array **digitalInputPins** lo toma de **analogInputPins**.

- **OutputPulse():** La función comienza de la misma manera que **OutputBit()**, es decir, escribiendo el en un indicado por **splitCommand[1]** el valor especificado por **splitCommand[2]**. Luego llama a la función **delay()** con el valor indicado en **splitCommand[3]**. Esto dormirá el programa los milisegundos que contenga su parámetro. Por último, vuelve a hacer una escritura en el mismo pin, pero esta vez con el valor contrario al que se hizo la primera vez.

- **InputByte():** Esta función lee a la vez todas las entradas digitales y devuelve un número cuyo valor es el del byte leído. Para leer los todos los pines a la vez se utiliza la sentencia **PIN** seguida del puerto a leer que es el K. Esto se puede hacer

porque todos los pines de lectura se encuentran en el mismo puerto. Luego, al igual que **inputBit()** o **inputAnalog()**, envía a la aplicación de escritorio el resultado de la lectura precedida de ***!OI***.

- **OutputByte():** Esta función sirve para la ejecución de tres instrucciones diferentes, que son la de byte alto, byte bajo y palabra. Estas funciones lo que buscan es escribir el valor de un byte en los 8 pines de salida digital más bajos, en los 8 más altos o en los 16 a la vez. Dependiendo de la instrucción que lo llame, le mandará un número a la función para que haga lo que se espera de ella. Si la función recibe un 0, querrá decir que es la instrucción byte bajo, que se corresponde con el puerto A, así que le asignará a la sentencia **PORTA** el valor de **splitCommand[1]**. Si la función recibe un 1, será la instrucción byte bajo y se hará lo mismo pero con **PORTC**. Por último, si la función recibe un 2 se referirá a la instrucción palabra. En este caso se asignarán los ocho bits menos significativos de **splitCommand[1]** a **PORTA** y los ocho bits más significativos a **PORTC.** Como en todas las funciones, al final envía la señal de confirmación a la aplicación de escritorio.

- **OutputAnalog():** Esta función es prácticamente igual a **outputBit()**. Las únicas diferencias están en que usa la función **analogWrite()** en vez de **digitalWrite()** y que en este caso toma los valores de los pines del array **analogOutputPins**.

- **WaitBit():** Es la función usada para la instrucción de espera por entrada digital. Está continuamente ejecutando un bucle vacío que tiene como condición de salida que la lectura del pin digital indicado por **splitCommand[1]** sea igual al valor de **splitCommand[2].** En otras palabras, el sistema se queda en un estado de bloqueo hasta que en el pin se lea el valor esperado. Cuando termina envía la confirmación de comando correcto, desbloqueando también la ejecución en la aplicación de escritorio.

- **WaitAnalog():** Esta función tiene la misma filosofía que la anterior pero orientada a las entradas analógicas. A diferencia del anterior en este caso no se busca igualdad, sino que dependiendo de lo que haya elegido el usuario, la ejecución se va a quedar congelada hasta que el valor de lectura sea o bien mayor o bien menor que el indicado por el usuario. Como era de esperar en vez de usar **digitalRead()** se usará **analogRead()**. También al final de su ejecución envía la confirmación de comando correcto, haciendo que se desbloquee la aplicación de escritorio.

- **StartRegulator():** Habilita las interrupciones del Timer 3 para que el regulador de presión comience a funcionar. Además almacena en **pressure** el valor de consigna del regulador que llega como parámetro.

- **ChangePressure():** Simplemente cambia el valor de **pressure** por el valor indicado en el parámetro del comando.

- **StopRegulator():** Deshabilita las interrupciones del Timer 3 para que el regulador de presión deje de actuar. Además desactiva el pin asociado al controlador por si este se encontrase activo en el momento de parar el regulador.

- **ISR(TIMER3_COMPA_vect):** Es la rutina de servicio de la interrupción del Timer 3. Su cometido es leer el valor actual de presión del sistema y si la presión se encuentra por debajo del valor de consigna actuar sobre la válvula de entrada de presión. En caso de que sea mayor que el valor de consigna debe actuar sobre la válvula de desahogo.

6. Experimentos y resultados

6.1 El ebullómetro

Para comprender mejor el objetivo final de este trabajo, es de suma importancia entender qué es un ebullómetro, de que tipo es el ebullómetro con el que vamos a trabajar y por supuesto cómo funciona el mismo.

El fin para el que se emplea el ebullómetro en el laboratorio es la determinación experimental de datos isobáricos de equilibrio líquido-vapor. El método consiste en la generación de un vapor a partir de una mezcla líquida en ebullición que luego se condensa y se recoge como líquido en un depósito para, posteriormente, retornar a la cámara de ebullición. El sistema se mantiene en operación hasta que, además de la constancia de las variables intensivas de presión y temperatura, la concentración en ambos recipientes permanece constante; en ese momento se extraen muestras de líquido y vapor para su análisis. A continuación, veremos el esquema del ebullómetro y nos haremos una idea general de su funcionamiento.

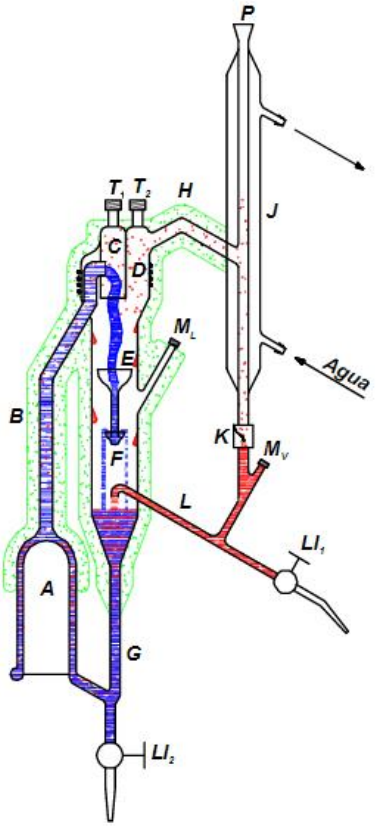

Ilustración 47: Esquema de un ebullómetro

La ebullición de la mezcla líquida se lleva a cabo en el vaso invertido de doble pared (A), mediante calentamiento eléctrico. El tubo (B), evita el recalentamiento de la mezcla, hace ascender al líquido y al vapor hasta la desembocadura del ensanchamiento (C), donde está situado el sensor de temperatura T1 que señala la temperatura de ebullición de la mezcla. Dicho tubo (C), hace de pantalla para evitar, junto al ensanchamiento del equipo (D) y la salida lateral (H) el arrastre de gotas de líquido por el vapor que se dirige al refrigerante. Por un lado, el vapor después de atravesar (H) se condensa en el refrigerante (J) y gotea sobre el colector (L). Por otro, el líquido cae sobre el embudo (E) que lo canaliza hacia (F), el vapor condensado y el líquido rebosa hacia (G), donde se mezclan, regresando el líquido de nuevo hacia el calderín (A) a través de un estrechamiento, con el que se evita el retroceso de la mezcla fluida. La parte superior del refrigerante (P) va conectada a un sistema de regulación de presión.

Es en el sistema de regulación de presión donde más toma importancia el cometido de este proyecto. En los experimentos que se realizan es vital el equilibrio líquido-vapor isobárico, por ello el sistema debe mantener establecida durante las fases del experimento que sean necesarias. Teniendo en cuenta que en un sistema cerrado donde un material está en ebullición la presión se verá afectada, se debe idear un sistema que sea capaz de tomar un valor de consigna y mantenerlo ya sea extrayendo o añadiendo presión al sistema.

6.2 Experimento

Para comprobar la funcionalidad del sistema desarrollado durante el trabajo, se confeccionó un experimento que pusiera a prueba distintos aspectos del mismo. Debido a motivos de disponibilidad no se tuvo acceso al ebullómetro para realizar las pruebas del sistema, por ello se dispuso un panel de leds para simular la activación y desactivación de las válvulas que componen el ebullómetro. También para simular el sistema de control de presión se controló un calentador de agua, dado que la acción de mantener una temperatura constante calentando y enfriando se asemeja bastante a la de mantener una presión constante en un sistema ya sea inyectando o liberando. En la siguiente figura se puede observar un esquema con los componentes del sistema que es muy importante para entender la secuenciación del experimento.

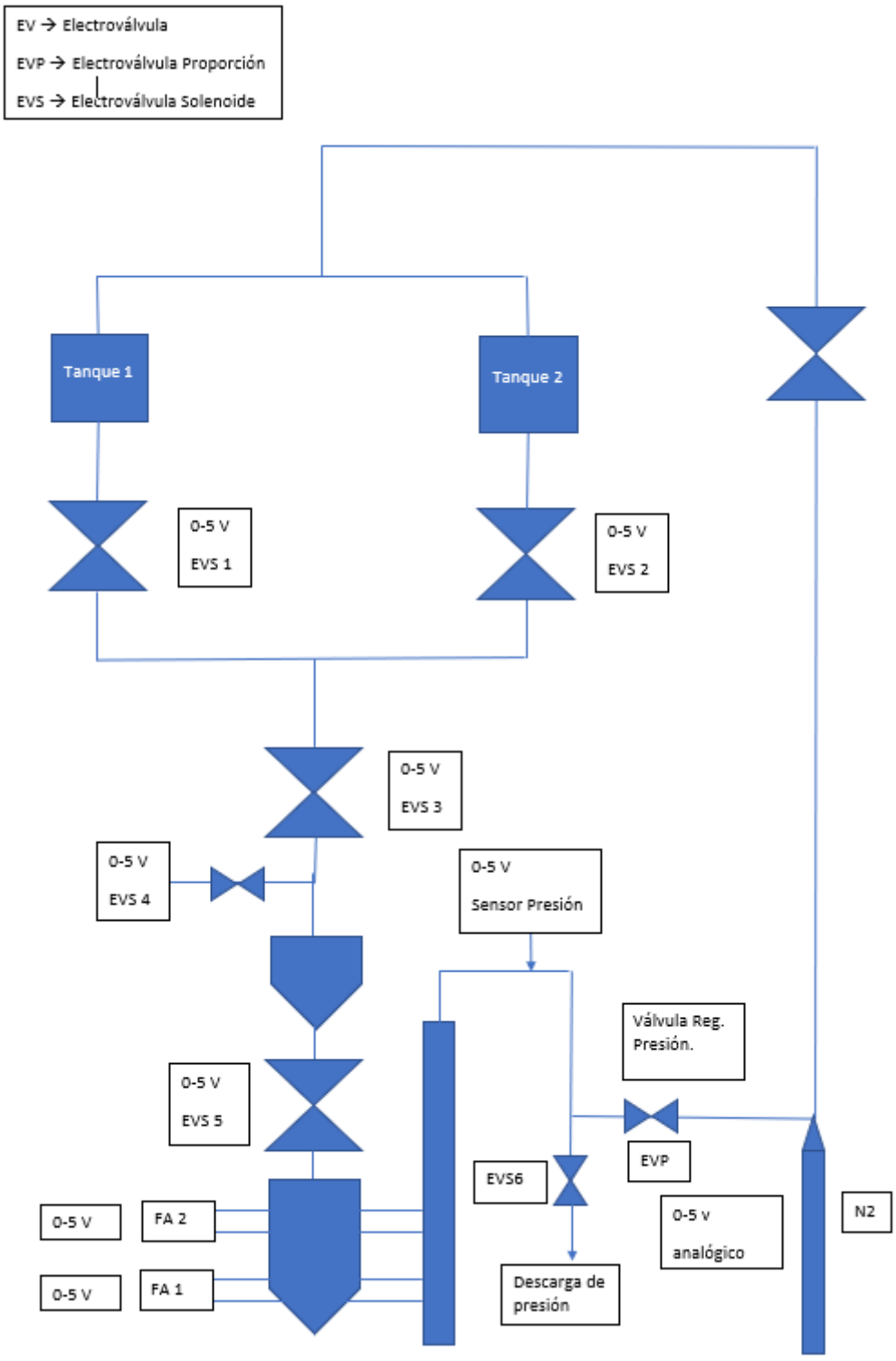

Ilustración 48: Esquema de los elementos del ebullómetro

Una vez realizada la construcción hardware del experimento se procedió a usar la aplicación de escritorio para programar la secuencia de comportamiento del controlador. Para entender mejor la secuencia programada, primero se va a mostrar cual es la secuencia normal de funcionamiento de un ebullómetro.

1. Se comprueba que existan las condiciones iniciales y se rellenan los tanques 1 y 2 con las sustancias deseadas. En este paso no interviene el controlador.
2. Se cierran todas las válvulas.
3. Se deja pasar parte de la sustancia del tanque 1 al calderín del ebullómetro.
4. Se le proporciona energía al sistema por medio de las fuentes de alimentación para calentar la sustancia.
5. Se ingresa presión en el sistema en forma de gas.
6. Se espera a unas condiciones estacionarias determinadas por el experimento que se esté realizando.
7. Se deja pasar parte de la sustancia del tanque 2 al calderín del ebullómetro.
8. Se mantienen las condiciones de equilibrio un tiempo a determinar por el experimento.
9. Se extraen muestras de líquido y gas del ebullómetro para analizar los resultados.
10. Se repiten los pasos 7, 8 y 9 las veces que determine el experimento.

Con la anterior secuencia en la cabeza, se ha confeccionado el siguiente programa en la aplicación de escritorio.

Programa:

```
<START>
>>Salida Binaria Bit - Pin 0 - Valor 0
>>Salida Binaria Bit - Pin 1 - Valor 0
>>Salida Binaria Bit - Pin 2 - Valor 0
>>Salida Binaria Bit - Pin 3 - Valor 0
>>Salida Analógica 0 = 0
>>Salida Binaria Bit - Pin 0 - Valor 1
>>Salida Binaria Bit - Pin 2 - Valor 1
>>Salida Binaria Bit - Pin 3 - Valor 1
>>Espera: 5000 milisegundos
>>Salida Binaria Bit - Pin 0 - Valor 0
>>Salida Binaria Bit - Pin 2 - Valor 0
>>Salida Binaria Bit - Pin 3 - Valor 0
>>Salida Analógica 0 = 150
>>Activar Regulador de presión a 250 kPa
>>Esperar entrada binaria - Pin 0 - Valor 1
BUCLE:
>>Salida Binaria Bit - Pin 1 - Valor 1
>>Salida Binaria Bit - Pin 2 - Valor 1
>>Salida Binaria Bit - Pin 3 - Valor 1
>>Espera: 5000 milisegundos
>>Salida Binaria Bit - Pin 1 - Valor 0
>>Salida Binaria Bit - Pin 2 - Valor 0
>>Salida Binaria Bit - Pin 3 - Valor 0
>>Esperar entrada binaria - Pin 0 - Valor 1
>>Salto a etiqueta a BUCLE
<FINISH>
```

Ilustración 49: Programa para el experimento

Las primeras cuatro instrucciones del programa se corresponden con las señales de accionamiento de las electroválvulas 1, 2, 3 y 5, con ellas lo que hacemos es cerrarlas como se especifica en los pasos del experimento. Luego también se pone a 0 la salida analógica 0 correspondiente al control de las fuentes de alimentación de los calentadores con el objetivo de asegurarse que no están actuando en ese momento.

El siguiente paso es activar las válvulas 1, 3 y 5 para dejar pasar al calderín la sustancia del tanque 1. En el programa se especifican 5 segundos de espera antes de volver a cerrar de nuevo las válvulas aunque esto es orientativo, ya que el tiempo dependerá de la cantidad de sustancia que se quiera introducir.

Más adelante se le da un valor a la salida analógica asociada a las fuentes de alimentación para que comience a calentarse la sustancia en el calderín. Además, se activa el regulador de presión para que se mantenga constantemente al valor de consigna que se considere oportuno. Hay que destacar que como se explicó anteriormente, en este caso el sistema

no controló presión, sino un calentador de agua. En este momento el programa se queda a la espera de recibir un valor alto por la entrada digital 0, en esta entrada habrá un pulsador conectado. Esto se hace debido a que el sistema debe llegar a unas condiciones estacionarias antes de seguir adelante con la secuencia, es por ello que sólo cuando el usuario determine que estas condiciones se han alcanzado, pulsará el botón y se reanudará las secuencia.

Por último el programa entra en un bucle dentro del cual introduce parte de la sustancia del tanque 2 dentro del calderín. Esto lo hace de la misma manera que cuando se introduce la sustancia del tanque 1 al principio. Luego el programa permanece en espera hasta que el usuario lo determine mediante el pulsador utilizado anteriormente. Durante esa espera el experimento podrá mantener la condición de equilibrio y el usuario podrá extraer muestra de vapor y líquido para analizarlas. Cuando el usuario pulse el botón se repetirá de nuevo el bucle y continuará de esta manera hasta que se crea conveniente, parando desde la interfaz gráfica la ejecución de la secuencia cuando se desee.

6.3 Resultados

Después de ejecutar el programa expuesto anteriormente se puede observar que el sistema desarrollado durante este Trabajo de Fin de Grado funciona correctamente y a pesar de tener ciertas limitaciones, resulta funcional. Dentro de las limitaciones que se pueden observar está la sensibilidad al ruido eléctrico para el uso del regulador. Concretamente afectaba a la hora de leer la temperatura del sensor, dado que esta oscilaba de vez en cuando, haciendo que cuando los valores de temperatura se acercaban a los de consigna, el regulador tuviera un comportamiento algo errático. También se analizaron los tiempos de respuesta del sistema con los comandos dando resultados de unos 30 milisegundos cuando no había esperas de por medio, lo cual es un tiempo bastante aceptable en un sistema de estos que no requiere de una gran rapidez. Cabe destacar que al no hacer el experimento sobre el ebullómetro real, no se puede valorar con total seguridad el sistema, pero por lo general, los resultados del experimento de funcionamiento se pueden catalogar como satisfactorios.

7. Conclusiones y trabajo futuro

7.1 Conclusiones

Al comenzar este trabajo se plantearon los objetivos de crear una interfaz hombre máquina que permitiese monitorizar y secuenciar acciones sobre un controlador programable. El entorno gráfico a diseñar debía ser sencillo de usar por arte de los usuarios, debía haber un repertorio de instrucciones que permitiese al sistema tener una funcionalidad mínima para ciertos procesos y ofrecer ciertas características para monitorizar el sistema.

Una vez realizado el trabajo se puede concluir que la mayoría de los objetivos han sido alcanzados, ya que el sistema desarrollado es sencillo de usar y ofrece una funcionalidad

básica para la tarea que fue concebido. El único aspecto del trabajo donde creo que no se alcanzaron todas las metas planteadas fue el sistema de monitorización, que es algo más pobre de lo que en un principio se había planteado debido a la falta de tiempo para su completo desarrollo.

En general, el desarrollo Trabajo de Fin de Grado ha supuesto un gran enriquecimiento para ciertas cualidades que me pueden ser de gran utilidad en mi vida profesional. Por ejemplo me ha desarrollado a la capacidad de idear un sistema nuevo que cubra unas necesidades que no estén resueltas con tecnologías actuales. También me ha enseñado a enfrentarme dificultades aprender a usar sobre la marcha tecnologías o patrones que eran desconocidos para mí. Por tanto, se puede concluir que la realización de estos trabajos supone una muy buena experiencia para los estudiantes.

7.2 Trabajo futuro

Al comienzo de esta memoria se indicó que la idea era desarrollar un sistema lo suficientemente abierto como para que en un futuro se pudiesen hacer las modificaciones necesarias para adaptar el sistema a otras maquinarias o bien para extender sus funcionalidades.

Dentro de los posibles nuevos trabajos que se pueden plantear a partir de este podría estar como se menciona anteriormente, una mejora de la monitorización del sistema. Esta mejora podría ir orientada a mostrar de forma gráfica los distintos elementos del sistema a controlar así como parámetros relativos a ellos en directo.

Otra posible vía de trabajo futuro sería el desarrollo de comunicaciones remotas con el sistema, por ejemplo a través de un servidor web.

Bibliografía

A continuación, se pasará a mencionar las distintas fuentes consultadas a la hora de elaborar esta memoria.

[1] Ana María Blanco Marigota: “Análisis del equilibrio liquido-vapor a 141,3 kpade mezclas binarias que contienen metanol conn-alcanos (c5,c6) y con esteres alquílicos”. [Fecha de consulta: 10 de junio de 2018].

[2] Andreas Eckel: “It is a Disruptive World with Exponential Development: Expected Trends in Embedded Systems, with a Special Focus on the Automotive Industry and its likely Effects on Industry and Society”. [Fecha de consulta: 15 de junio de 2018].

[3] Marilyn Wolf: “Computers as Components: Principles of Embedded Computing System Design”. Elsevier. 12 de junio de 2012. [Fecha de consulta: 23 de junio de 2018].

[4] Wikipedia. https://es.wikipedia.org/wiki/Wikipedia:Portada [Fecha de consulta: 20 de junio de 2018].