UNIVERSITAT JAUME I

Grado en Ingeniería Informática

Trabajo Final de Grado

Simulador de Impresora Industrial de Alta Velocidad

Autor:
Víctor Sánchez Terrasa

Supervisor:
Javier Oltra Puchol
Tutor académico:
José Vicente Martí Avilés

Fecha de lectura: 1 de julio de 2018
Curso académico 2017/2018

Resumen

En el presente documento se recoge el trabajo realizado en la empresa EFI-Cretaprint tratándose sobre el desarrollo de una aplicación capaz de simular las funciones de un controlador lógico programable. En concreto, se describirá la función que realiza dicho controlador para manejar cada impresora de cartón corrugado que fabrica y vende la compañía.

Las etapas del desarrollo de la aplicación que efectúa la función de controlador lógico programable dentro de la impresora, desde aquí en adelante PLC, se encuentran disponibles para el lector en el presente documento, con cada etapa como la definición del proyecto: documentación de los elementos necesitados para el desarrollo, requisitos del proyecto, análisis del proyecto, diseño de la aplicación, desarrollo, problemas que han surgido durante el desarrollo y puesta en marcha junto con la aplicación con la que ha de funcionar, así como también se encontrará al final una opinión del proyecto.

El alcance de este sistema incluye todo el manejo con el apartado del controlador lógico programable, pero no incluye los servicios de las capas de más nivel que interactúan con el usuario, al tratarse de una herramienta interna de desarrollo.

La herramienta interna del departamento de desarrollo de software de la empresa es el principal motivo de todo el trabajo realizado. La necesidad de esta herramienta viene por el hecho que la empresa solo dispone de una impresora de desarrollo en la que se realizan las distintas pruebas de la siguiente generación de impresoras. Esta única impresora es utilizada por los distintos departamentos para la realización de sus pruebas, teniéndose que repartir las horas de pruebas entre los distintos empleados de cada departamento. A partir de este problema surge la idea de crear una aplicación capaz de simular la comunicación realizada con el PLC que dispone la impresora.

Palabras clave

Simulador, modbus, PLC

Keywords

Simulator, modbus, PLC

Índice general

Capítulo 1

Introducción

1.1. Contexto del proyecto

El proyecto cuya propuesta se presenta en este documento, se ha desarrollado en EFI-Cretaprint S.L.U. subsidiaria de la empresa multinacional Electronics for Imaging, Inc.(EFI) con una sede central de más de 3100 trabajadores ubicada en la ciudad de Fremont del estado de California, Estados Unidos. Dicha empresa dispone de principales subsidiarias importantes para el desarrollo de la compañía empleando un sistema de desarrollo Spring, proponiendo objetivos a corto plazo, así de esta forma corregir rápidamente los errores que se puedan encontrar durante su desarrollo. Con este sistema, EFI es la compañía líder mundial en el sector en la fabricación de impresoras de cartón corrugado gracias a la subsidiaria EFI-Cretaprint S.L.U. que tiene la sede principal en el Carrer dels Ibers nº 54 ubicado en el Polígono Industrial SUPOI 8 del municipio de Almazora, provincia de Castellón, para todo el territorio español. Con una cantidad en aumento de 80 trabajadores investigan, desarrollan y fabrican las impresoras de cartón corrugado de la compañía que se encuentran actualmente por todo el mundo.

EFI permite que EFI-Cretaprint S.L.U. pueda actuar con total autonomía en numerosos aspectos, como si se tratara de una empresa externa, empleando apoyo de la matriz EFI en ciertos tipos de recursos como pueden ser el uso de aplicaciones administrativas, correo electrónico, organigrama empresarial... En el caso del organigrama empresarial, la empresa dispone de los departamentos básicos que requiere cualquier empresa en expansión: recursos humanos, contabilidad, inventario y administración. Después nos encontramos con los departamentos que le dan valor a la empresa en el mundo de investigación: software, ingeniería, estructuras, electricidad, mecánica, materiales y tintas. Todos estos departamentos trabajan de forma horizontal colaborando entre ellos cuando se cruzan las partes del proyecto. Cada departamento se divide equipos especializados en una parte de desarrollo, para una mayor eficacia en la distribución de conocimientos, así como también tiene apoyo de empresas externas en las que los trabajadores de estas empresas externas cuentan como un trabajador más de EFI.

Las impresoras de EFI son capaces de imprimir tanto en cartón corrugado de diversas ondas como en cartón compacto. Dicho cartón es el que muchas veces envuelve aquellos productos que compramos en las tiendas de electrodomésticos, de ahí la demanda de este tipo de impresoras

que con un tamaño de 180m^2 poseen una tasa de impresión de hasta 75m/minuto lineales (unos 7224m^2/hora) con las que pintan las cajas contenedoras de los productos que tanto anhelamos tener en perfecto estado.

EFI investiga la creación de nuevas impresoras de cartón corrugado, siendo un producto que no se puede disponer en cuantía in situ por su gran tamaño y alto coste por cada unidad. Esto supone un problema a la hora de poder experimentar con ellas, generando muchos problemas para la continuación del proyecto. Por tanto una de las soluciones planteadas es la de simular la mayor cantidad de partes de la máquina, para que esta forma la dependencia física sea la menor posible.

1.2. Motivación del proyecto

Partiendo de la necesidad de depender lo mínimo posible de la impresora física en el momento de su desarrollo, al departamento de software de EFI se le ocurrió que sería una buena idea poder simular aquellos elementos que dependen de partes mecánicas de la impresora. Una de estas partes es el controlador lógico programable, también conocido en la industria como *PLC*, que es el encargado de controlar los elementos mecánicos de la impresora.

El sistema debe permitir interactuar de la misma forma que con una impresora real, incluyendo los estados de funcionamiento en los que pudiera estar, generando los posibles errores y alarmas que puedan existir dentro de la máquina, además de disponer de una fácil instalación de la aplicación desarrollada. El sistema ha de gestionar todas las comunicaciones mediante un protocolo de comunicación específico, generar las posibles alertas, e informar del estado de cada parte de la máquina, así como también la configuración dada a cada parte. Un sistema como éste permitirá que el tiempo de la lista de espera con la máquina sea nulo, al disponer de la impresora virtual siempre preparada para su uso. Con ello, se agilizarán las pruebas con la máquina, consiguiendo así reducir el tiempo de los posibles errores que se generen en el instante de realizar las pruebas.

El desarrollo de este sistema incluye solo la simulación del controlador lógico programable, pero no incluye los servicios de las capas de nivel superior las cuales interactúan con el usuario, ya que esta herramienta solo va a ser de uso interno para el departamento de software.

1.3. Objetivos del proyecto

El principal objetivo de este proyecto es reducir al máximo el tiempo de espera para uso de la impresora, disponiendo un simulador de esta, para hacer todas las pruebas software posibles con ésta aplicación como herramienta.

El objetivo principal se puede desglosar en los siguientes subobjetivos:

- Comunicación con el dispositivo mediante el protocolo *modbus*.

- Responder a las solicitudes, conforme dicta el protocolo interno de la empresa.
- Generación de posibles resultados óptimos en función de los parámetros configurados, el estado de la máquina y los trabajos solicitados.
- Generar una interfaz para el uso del programa.
- Comprender el programa cedido por la empresa del cual se va a partir.
- No depender de medios físicos mecánicos.

1.4. Estructura de la memoria

En este documento se encuentran las etapas del desarrollo de la aplicación pudiendo encontrar en el capítulo 2 la definición del proyecto, como también los requisitos y una descripción de las principales tecnologías usadas en el proyecto, junto con su motivo de uso. El capítulo 3 muestra la metodología que la empresa emplea, la misma empleada en el proyecto, así como también la planificación y el seguimiento que se ha realizado del proyecto. En el capítulo 4 ya se encuentra más información de campo de la informática como el análisis realizado por la aplicación, además del diseño realizado posteriormente del análisis, tanto de la arquitectura del sistema, como de la interfaz. El documento continúa con el capítulo 5 en el que se muestra la implementación de la aplicación, como la puesta en marcha y los errores encontrados en ésta. Finalmente, en el capítulo 6, se tiene a disposición las conclusiones del producto.

Capítulo 2

Descripción del proyecto

Para el siguiente proyecto EFI-Cretaprint ha permitido una gran flexibilidad en muchos ámbitos, con la restricción de emplear la tecnología de la propia empresa, así como el lenguaje de programación *C#*, distintas bibliotecas como puede ser *NModbus4*, *LightMVVM* y algunas internas de la empresa, así como también la metodología de desarrollo *Scrum* y la organización de los estados del proyecto haciendo uso de *git flow*, siendo descrito cada uno en este apartado por ser tecnologías empleadas en el desarrollo del proyecto.

2.1. Punto de partida: Aplicaciones y herramientas de EFI Cretaprint

Para el desarrollo de la aplicación se ha hecho uso de aplicaciones, herramientas y bibliotecas de la empresa donde se ha hecho la estancia en prácticas, **la cuales no se dispone de la autorización sujetas al acuerdo de confidencialidad.**

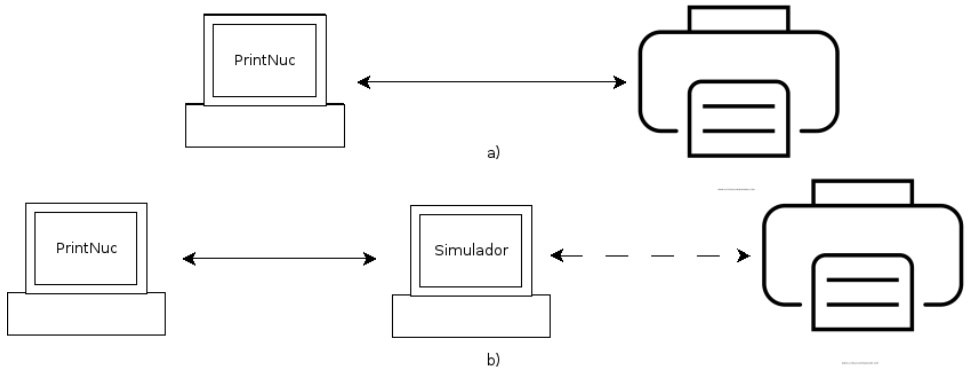

Figura 2.1: En la imagen a) se ve como hacen pruebas actualmente con la aplicación en el *PLC* directamente, por contra en la opción b) es la solución que se da, con la opción de conectar al *PLC* o no.

En todas estas herramientas y aplicaciones se sustenta desarrollo de la aplicación. Se ha hecho uso de nombres ficticios para conservar el anonimato de los productos de la compañía. Un ejemplo es la aplicación de la empresa la cual se conecta a la aplicación desarrollada, empleando el nombre *PrintNuc* para esta aplicación a lo largo de todo el documento.

La idea que se plantea es sustituir las pruebas que se realizan entre la aplicación *PrintNuc* y el *PLC* de la impresora, cambiando el *PLC* de la impresora por un *PLC* simulado virtual. En la Figura 2.1 podemos observar que la aplicación simulada también tiene como opción conectarse al *PLC*, esto permite ver la información que realmente va desde *PrintNuc* hasta el *PLC*.

2.2. Controlador lógico programable (PLC)

Un **controlador lógico programable** o autómata programable [1], también comúnmente denominado **PLC** de sus siglas en inglés (**P**rogrammable **L**ogic **C**ontroller), es un dispositivo empleado principalmente en la industria para automatizar procesos electromecánicos. Los *PLCs* controlan la lógica a partir de procesar y recibir señales digitales y analógicas, además de poder aplicar estrategias de control.

Figura 2.2: PLC Siemens Simatic S7-400, de arriba a abajo: fuente de alimentación, CPU, módulo de interfaz y el procesador de comunicaciones [1].

Los *PLCs* son utilizados en muchas industrias y máquinas por estar diseñados para múltiples

señales de entrada y de salida, rangos de temperatura ampliados, inmunidad al ruido eléctrico y resistencia a la vibración y al impacto. Los programas para el control de funcionamiento de la máquina se suelen almacenar en baterías, copia de seguridad o en memorias no volátiles. Un *PLC* es un ejemplo de un sistema de tiempo real «duro», donde los resultados de salida deben ser producidos en respuesta a las condiciones de entrada dentro de un tiempo limitado, el resultado no valdrá de nada, aún siendo correcto [2].

2.3. Metodología de trabajo Scrum

Scrum [3] es una metodología de trabajo ágil caracterizada por adoptar una estrategia de desarrollo incremental, basada en la calidad del resultado junto el conocimiento tácito de las personas en equipos autoorganizados y solapando las diferentes fases del desarrollo.

2.3.1. JIRA

JIRA [4] es la herramienta perteneciente a la empresa Atlassian, empleada por EFI, para poder implementar la metodología de trabajo *Scrum*. Trata de ser una herramienta para la administración de tareas, el seguimiento de errores e incidencias y para la gestión operativa de proyectos.

2.4. Formato de texto JSON

JSON [5] es un formato de texto ligero para el intercambio de datos, como *XML*. Es común ver el uso de *json* como sustituto de *XML* o complementado junto a *XML*. Destaca por la facilidad de desarrollo en analizadores y por el uso de pocos recursos empleados en las máquinas.

2.5. Microsoft Visual Studio

Microsoft Visual Studio [6] se trata de una herramienta que es un entorno de desarrollo integrado para sistemas operativos *Windows*. Soporta múltiples lenguajes de programación, entre los que destaca *C#*, por ser el empleado en el desarrollo del proyecto. Se emplea con la finalidad de escribir la aplicación, facilitando distintas tareas, como pueden ser la autocorrección, la compilación, el depurado y la inclusión de otras bibliotecas a través de un repositorio en el cual otros desarrolladores han depositado sus trabajos de forma gratuita.

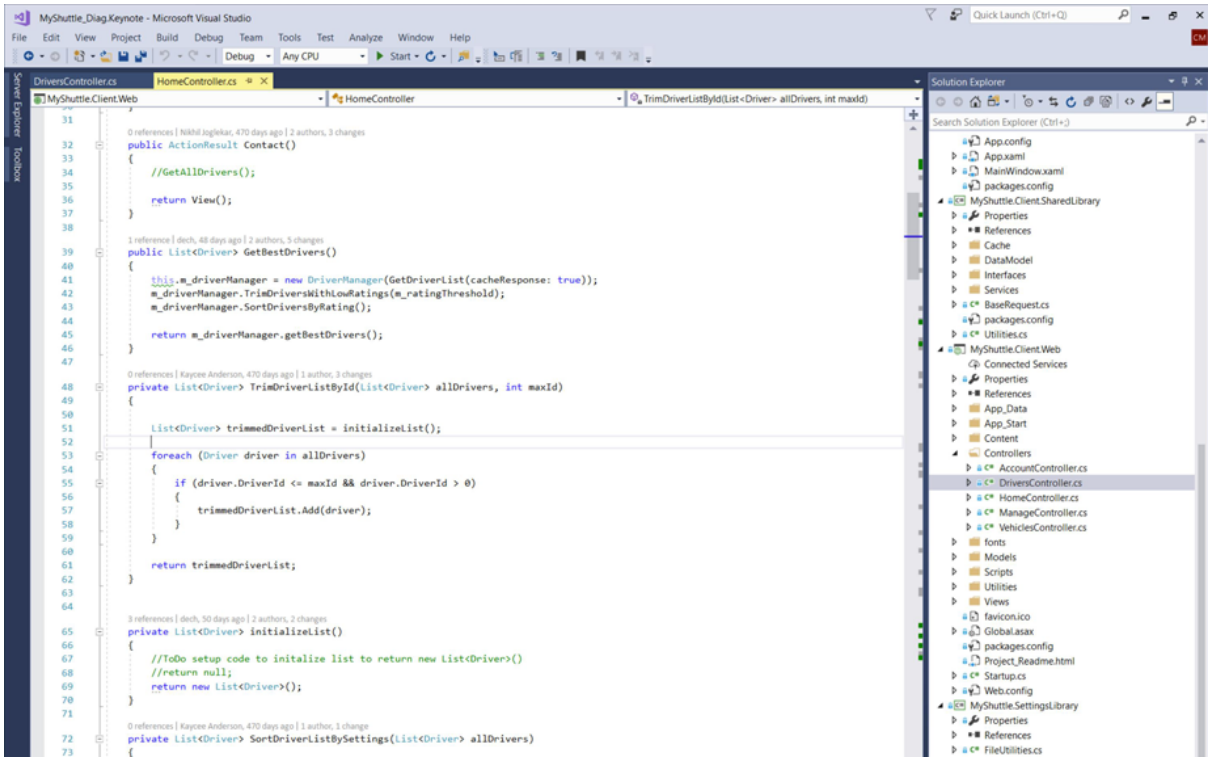

Figura 2.3: Microsoft Visual Studio en ejecución mientras se desarrolla una aplicación.

2.6. Lenguaje C#

C# [7] es un lenguaje de programación orientado a objetos desarrollado y estandarizado por *Microsoft* como parte de su plataforma *.NET*, diseñado para la infraestructura de lenguaje común. Su sintaxis básica deriva de *C/C++*, empleando el modelo de objetos de la plataforma *.NET*, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes como pueden ser *Eiffel*, *Modula-3* o *Pascal*.

2.7. Windows Presentation Foundation(WPF)

Windows Presentation Foundation (WPF) [8] es una tecnología de *Microsoft* que permite el desarrollo de interfaces de interacción en *Windows* tomando características de aplicaciones *Windows*. Además ofrece una amplia infraestructura y potencia gráfica con la que es posible desarrollar aplicaciones visualmente atractivas, con facilidades de interacción, siendo fácilmente integrable con el lenguaje *C#*.

2.8. Patrón Modelo-Vista-Modelo de Vista

El patrón modelo–vista–modelo de vista [9] (en inglés, model–view–viewmodel, abreviado **MVVM**) es un patrón de arquitectura de software. Se caracteriza por tratar de desacoplar todo lo posible la interfaz de usuario de la lógica de la aplicación.

- El **modelo** es la capa de datos y/o la lógica de negocio, que se comunica con la aplicación para así obtener la información sin manipular o realizar acciones directamente, además de no poder depender de la vista.
- La **vista** representa la información a través de los elementos visuales que la componen, **siendo activas junto a comportamientos, eventos a datos** que tienen conocimiento de las capas inferiores.
- El **modelo de vista** es la capa intermediaria entre el modelo y la vista. Contiene toda la lógica de presentación, siendo estos procesos completamente ajenos a la interfaz, generando una comunicación entre la vista y el modelo de vista a través de enlaces de datos.

2.9. MVVM Light Toolkit(lightmvvm)

MVVM Light Toolkit [10] es una biblioteca que ayuda a reducir el tiempo de desarrollo en aplicaciones que emplean *WPF*, forzando el uso del patrón *MVVM* para el desarrollo de la interfaz visual, separando la interfaz (capa vista) del resto de la aplicación(capa modelo), creando una aplicación con un código mucho más legible a través de una capa intermedia (modelo de vista) mejorando el posterior mantenimiento de la aplicación.

2.10. Git

Git [11] es un software de control de versiones pensando en la eficiencia y la fiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente y/o equipos de desarrollo. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos a través de las ramas de trabajo que dispone.

El desarrollo de *git* se realiza a través del uso de **ramas**, donde cada rama es un espacio de trabajo donde se realiza una parte del trabajo que posteriormente se juntará con otra rama para completar entre varias ramas una tarea propuesta.

Cada vez que se abre una nueva rama, normalmente se hace una copia de la rama en la que se encuentra actualmente para realizar cambios que no afectan de la rama heredada. Normalmente cuando se cierra una rama, se indica a qué rama han de afectar esos cambios, pues de otra forma, todo el trabajo realizado se perdería.

2.10.1. Git-flow

Git-flow es una extensión que añade comandos a *git* para simplificar la gestión de ramas empleando la metodología de trabajo *Scrum*. Cuando se inicia un nuevo proyecto, nos encontramos con dos ramas principales, *master* y *develop*, creadas inicialmente por defecto, las cuales irán sufriendo cambios haciendo uso de las ramas *feature*, *release* y *hotfix*, teniendo cada una un uso distinto. A continuación describiremos la utilidad de cada rama:

- La rama **master** es la que contiene la última versión del proyecto que está en producción. Esta rama no se modifica directamente.
- La rama **develop** contiene el último estado del programa que se está desarrollando actualmente, siendo modificada principalmente por los resultados de las ramas *feature*. Esta rama no se modifica directamente.
- Las ramas **feature** se usan para desarrollar nuevas funcionalidades, parten de la rama *develop*, Cuando se cierra una rama, se fusiona otra vez con *develop*.
- Las ramas **release** se usan para lanzar una versión de producción, empleándose para realizar los cambios que sean necesarios para liberar la nueva versión, como cambios de configuración o de versión. Esta rama se crea a partir de la rama *develop*, una vez se cierra, es fusionada con las ramas *master* y *develop*.
- Las ramas **hotfix** se usan para cambios rápidos, con carácter urgente para solucionar problemas graves que requieren inmediatez de la rama de producción, requiriendo el arreglo de dicho fallo lo antes posible. Esta rama se crea a partir de la rama *master*, una vez se cierra, es fusionada con las ramas *master* y *develop*.

2.11. Protocolo modbus/TCP

Modbus [12] es un protocolo de comunicaciones basado en la arquitectura maestro-esclavo, diseñado para controladores lógicos programables (*PLCs*). Convertido en un protocolo de comunicaciones estándar de facto en la industria. Es el que goza de mayor disponibilidad para la conexión de dispositivos electrónicos industriales. En el desarrollo vamos a emplear *modbus/TCP* ya que el despliegue de la aplicación va a realizarse sobre redes *ethernet*.

En el caso de usar el protocolo *modbus/TCP* se emplea la estructura cliente-servidor donde el cliente es el maestro y el servidor es el esclavo. Este punto suele ser confuso, ya que cuando se hace el cambio de la nomenclatura se suele cruzar el concepto.

2.11.1. Modelo de Datos Modbus

El **modelo de datos en modbus** distingue entre entradas digitales, salidas digitales (también denominadas *coils*), registros de entrada y registros de retención (también denominados *holding registers*). Las entradas y salidas digitales ocupan un bit, mientras que los registros ocupan 16 bits o 2 bytes, pudiéndose ver en el Cuadro 2.1 esta información.

Tipo de objeto	Acceso	Tamaño
Entrada digital	Lectura	1 bit
Coil	Lectura/Escritura	1 bit
Registro de entrada	Lectura	16 bits
Holding registers	Lectura/Escritura	16 bits

Cuadro 2.1: Tipos de objetos proporcionados por un dispositivo esclavo *modbus* a un dispositivo maestro *modbus*.

2.11.2. Formato de la trama

En el Cuadro 2.2 se muestra el estándar de la trama *modbus* que emplean en la comunicación el cliente y el servidor. Cabe destacar que el orden de bytes es *big-endian*.

Nombre	Longitud (Bytes)	Función
ID de la transacción	2	Sincronización entre mensajes de servidor y cliente
ID del protocolo	2	0 para *modbus/TCP*
Campo de longitud	2	Número de bytes en esta trama
Identificador de unidad	1	Dirección del esclavo (255 si no se usa)
Código de función	1	Códigos de función como en otras variantes
Bytes de datos	n	Datos como respuesta o comandos

Cuadro 2.2: Visualización del formato estándar de trama *modbus/TCP*.

Se ha de comentar que el campo *identificador de unidad* se utiliza con dispositivos *modbus/TCP* que están compuestos por varios dispositivos *modbus* en las pasarelas *modbus/TCP* a *modbus/RTU*. Cuando se emplea de esta forma, el identificador de unidad indica la dirección de esclavo del dispositivo detrás de la pasarela. Aún así, en muchas ocasiones los dispositivos compatibles con *modbus/TCP* ignoran el identificador de unidad.

2.11.3. Biblioteca NModbus4

NModbus4 [14] es una biblioteca que implementa el protocolo *modbus* en el lenguaje de programación *C#*. Dispone de la comunicación en distintos medios, siendo *modbus/TCP* de nuestro interés, pudiéndose emplear tanto si queremos implementar un servidor o esclavo, o para implementar un cliente o maestro.

Esta biblioteca no solo ofrece soporte a la comunicación *modbus*. Además también ofrece un sistema de almacenamiento para la información de la comunicación en el caso de usar la biblioteca con el fin de crear un servidor *modbus*, pudiendo leer y escribir en cualquier posición del almacenamiento. Se ha de hacer hincapié en que el apartado de cliente o maestro no requiere utilizar el sistema de almacenamiento, puesto que solo es necesario para el servidor o esclavo, ya que necesitan un lugar donde almacenar la información que recibe.

2.12. Aplicación Modbus PLC Simulator

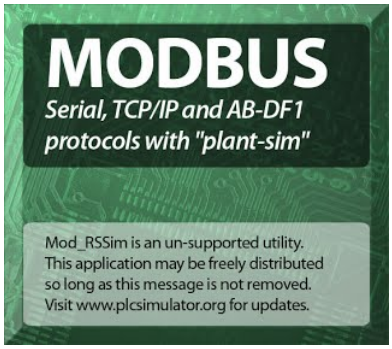

Figura 2.4: Icono de la aplicación *Modbus PLC Simulator*, antes llamada *Mod_RSSim*.

La aplicación **Modbus PLC Simulator** [15] ha ayudado mucho en la labor de puesta en marcha de la aplicación, ya que como bien indica su nombre, realiza las labores de simulación servidor *modbus*. Esta aplicación ha sido necesaria ya que en ningún momento se ha tenido acceso a ningún *PLC*.

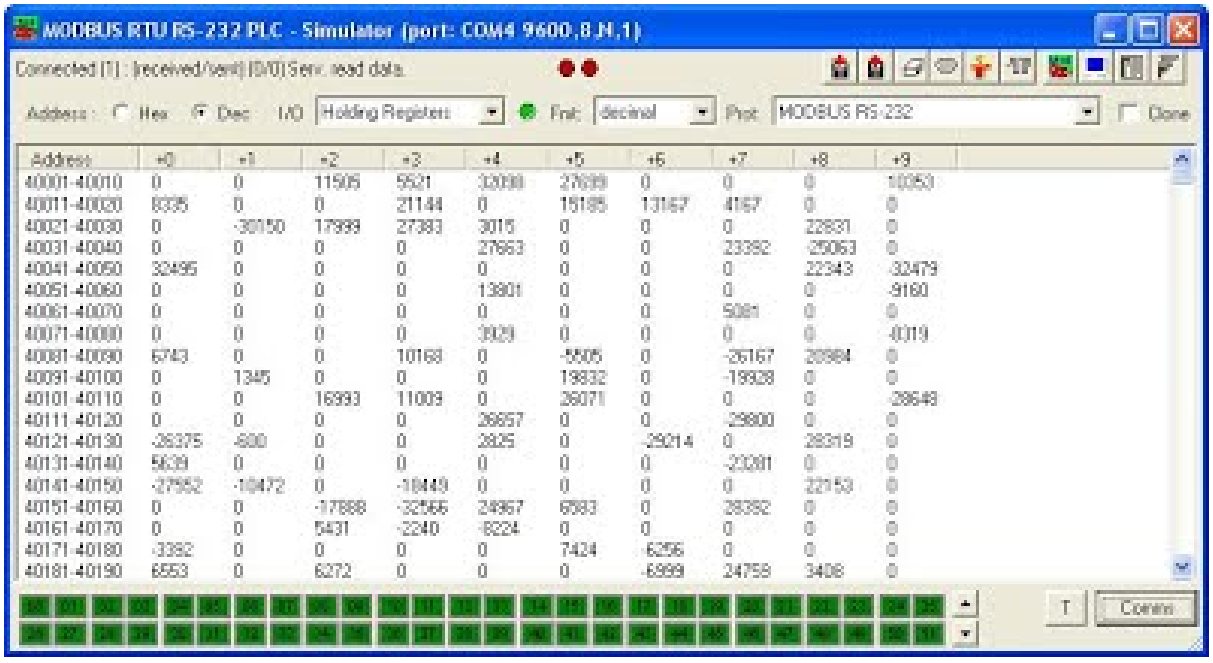

Figura 2.5: *Modbus PLC Simulator* en funcionamiento.

Presentando esta aplicación, el lector puede llegar a pensar que con el uso de esta aplicación no hubiese hecho falta la necesidad del proyecto, respecto a esa idea, se puede ver en la Figura 2.5 una muestra caótica de la representación de los datos, queriendo evitar esto con la aplicación que se ha propuesto desarrollar, aportando una interfaz mucho más amigable, sencilla y útil respecto a *Modbus PLC Simulator* .

2.13. Aplicación cliente Modbus

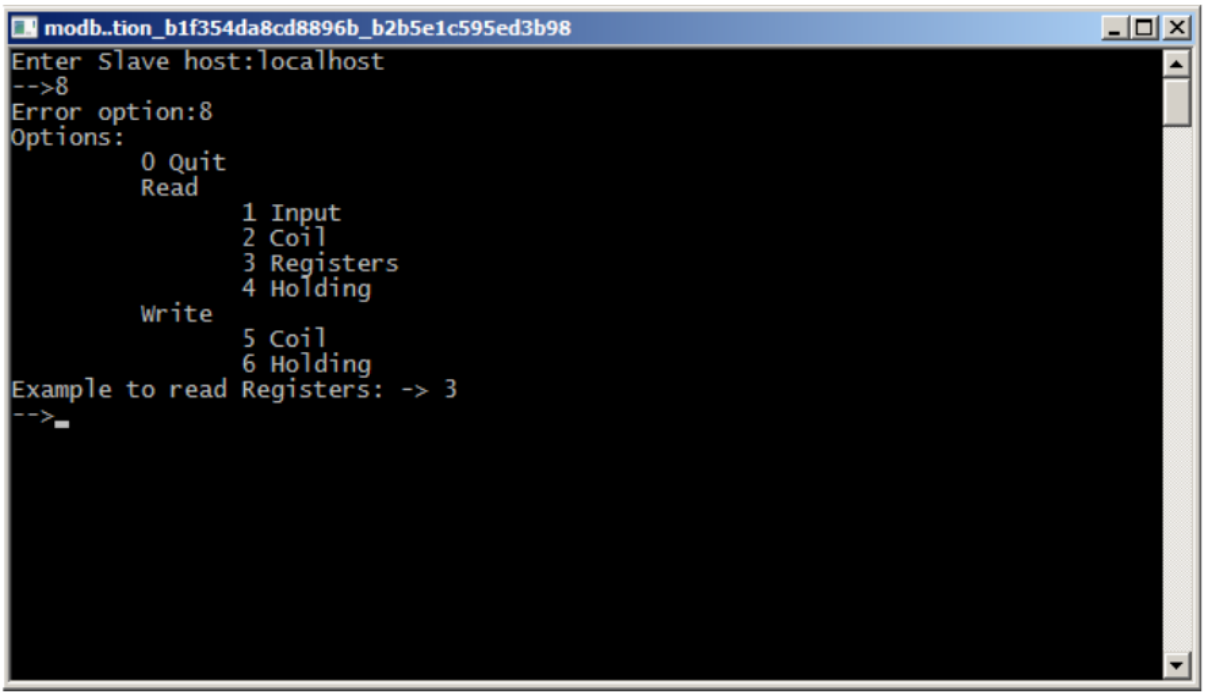

Figura 2.6: Aquí se visualiza el funcionamiento del cliente *modbus*.

Esta aplicación es de desarrollo propio, realizada los primeros días de estancia en la empresa para comprobar los conocimientos obtenidos acerca del patrón *MVVM* y del protocolo *modbus*. Ha sido de utilidad para la puesta en marcha, puesto que en un principio no se tenía acceso a la aplicación que la compañía ofreció posteriormente para la puesta en marcha, empleándola para las comunicaciones *modbus*. Como se observa en la Figura 2.6 es una aplicación sencilla que funciona mediante consola.

2.14. Wireshark

Wireshark [16] es una aplicación que actúa sobre las redes de comunicación para realizar un análisis de los protocolos que circulan en el momento que la aplicación está conectada.

Ofrece una funcionalidad semejante a al comando *tcpdump*, con la intuición que da una interfaz gráfica, permitiendo mostrar todo el tráfico que pasa a través de una red, además de ayudar con opciones de organización y filtrado de información. También permite examinar datos de una red, incluso desde un archivo de captura almacenado.

Esta aplicación se usa para analizar que el tráfico que se transmite entre el maestro y el esclavo con el protocolo *modbus* es el esperado, puesto que supone de gran ayuda poder filtrar el

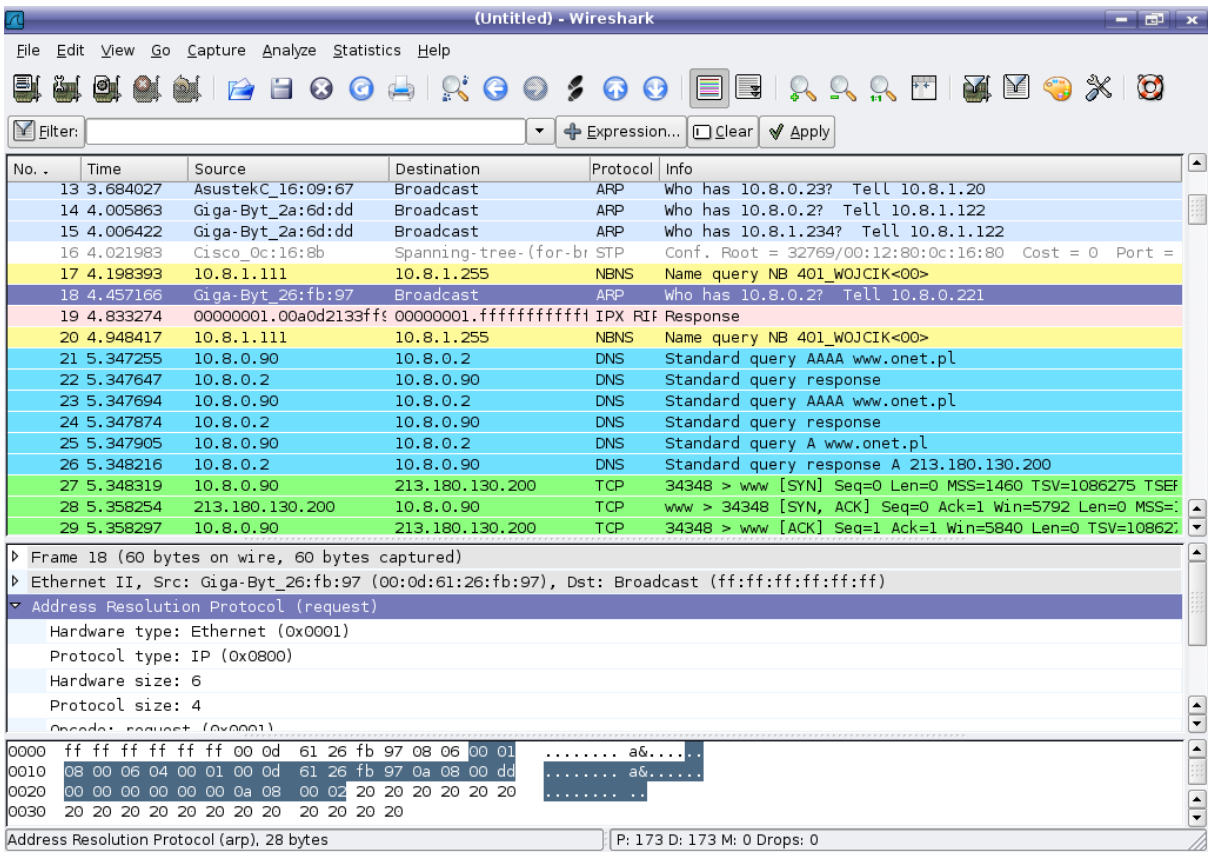

Figura 2.7: Wireshark mientras captura paquetes.

tráfico solo a paquetes *modbus*, ya que suelen ser una minoría de todos los paquetes que recibe el ordenador.

Capítulo 3

Planificación del proyecto

3.1. Metodología

El departamento de software de EFI-Cretaprint emplea una metodología *Scrum* basada en tareas a través de la plataforma *JIRA*. El responsable del departamento es quien establece una tarea dentro de la plataforma y los propios desarrolladores la evalúan para que posteriormente asignen las horas de trabajo requeridas para esa tarea. Cabe decir que los propios desarrolladores también son capaces de generar subtareas a partir de las tareas. En caso de que la tarea sea muy compleja o requiera más de una semana de trabajo, se dividirá el trabajo entre los trabajadores, disminuyendo también el tiempo requerido para la tarea. La misma metodología de trabajo empleada por la empresa, va a ser la metodología empleada en el simulador de una forma mucho más informal, al tratarse de un producto interno del departamento y no seguir las líneas generales de desarrollo de la empresa.

3.2. Requisitos

Los requisitos no han sido muy concretos, puesto que el principal ha sido el uso de los mismos recursos y herramientas que emplea la empresa, pudiéndose ver parte de las herramientas utilizadas las encontramos en el capítulo 2. Otros de los recursos necesitados para la elaboración del proyecto han sido un ordenador de sobremesa, con sus respectivos periféricos, que sea capaz de soportar el funcionamiento de las aplicaciones y conexión a internet para poder acceder a los recursos de la empresa y de terceros.

3.3. Planificación

3.3.1. Planificación inicial

Simulador de Impresora Industrial de Alta Velocidad	Tiempo (h.)	Dependencias
1. Desarrollar la propuesta técnica.		
1.1. Inicio.	**13**	
1.1.1 Definir proyecto con el tutor y supervisor.	1	
1.1.2 Definir método de trabajo y documentación.	6	1.1.1
1.1.3 Definir formato y estándares de trabajo.	6	1.1.1
1.2. Documentar y planificar el proyecto.	**32**	
1.2.1 Revisar contexto y buscar información.	26	1.1
1.2.2 Identificar alcance y objetivos.	6	1.1
1.3. Planificar el proyecto.	**33**	
1.3.1 Definir tareas y estimar fechas	6	1.2
1.3.2 Crear diagrama de Gantt	6	1.3.1
1.3.3 Documentar la propuesta del proyecto	20	1.3.1
1.3.4 Presentación de la propuesta técnica	1	1.3.3
2. Desarrollo técnico del proyecto		
2.1. Definir requisitos del proyecto	**39**	
2.1.1 Crear diagrama de Casos de uso	16	1
2.1.2 Definir y documentar requisitos de datos	10	1
2.1.3 Definir Requisitos tecnológicos y de la plataforma	13	1
2.2. Análisis	**60**	
2.2.1 Crear diagrama de clases	15	2.1
2.2.2 Documentar clases	15	2.2.1
2.2.3 Diagrama de actividades	15	2.1
2.2.4 Validar el análisi	15	2.2.3
2.3. Diseño	**41**	
2.3.1 Identificar y clasificar datos	7	2.2
2.3.2 Diseñar interfaz gráfica	6	2.3.1
2.3.3 Refinar diagrama de clases de diseño	8	2.3.2
2.3.4 Propuesta plataforma tecnológica	4	2.3.3
2.3.5 Validar diseño	16	2.3.4
2.4. Desarrollo del producto	**70**	
2.4.1 Programación	50	2.3
2.4.2 Pruebas	20	2.4.1
2.5. Puesta en marcha	**12**	
2.5.1 Implantación	6	2.4
2.5.2 Formación	6	2.5.1
2.5.3 Presentación final	0	2.5.2
3. Documentación y presentación del TFG	**147**	
3.1. Redacción de informes quincenales	16	
3.2. Redacción de la memoria técnica	100	
3.3. Presentación de la memoria técnica	0	3.2
3.4. Preparación de la presentación oral	30	3.3
3.5. Presentación oral	1	3.4

Figura 3.1: Desglose en tareas, junto con la planificación temporal y las dependencias entre tareas de la planificación inicial.

3.3.2. Planificación final

Simulador de Impresora Industrial de Alta Velocidad	Tiempo (h.)	Dependencias
1. Desarrollar la propuesta técnica.		
1.1. Inicio.	**20**	
1.1.1 Definir proyecto con el tutor y supervisor.	1	
1.1.2 Definir método de trabajo y documentación.	7	1.1.1
1.1.3 Definir formato y estándares de trabajo.	12	1.1.1
1.2. Documentar y planificar el proyecto.	**40**	
1.2.1 Revisar contexto y buscar información.	29	1.1
1.2.2 Identificar alcance y objetivos.	11	1.1
1.3. Planificar el proyecto.	**26**	
1.3.1 Definir tareas y estimar fechas	6	1.2
1.3.2 Crear diagrama de Gantt	6	1.3.1
1.3.3 Documentar la propuesta del proyecto	12	1.3.1
1.3.4 Presentación de la propuesta técnica	2	1.3.3
2. Desarrollo técnico del proyecto		
2.1. Definir requisitos del proyecto	**20**	
2.1.1 Crear diagrama de Casos de uso	12	1
2.1.2 Definir y documentar requisitos de datos	6	1
2.1.3 Definir Requisitos tecnológicos y de la plataforma	2	1
2.2. Análisis	**27**	
2.2.1 Crear diagrama de clases	7	2.1
2.2.2 Documentar clases	14	2.2.1
2.2.3 Validar el análisi	6	2.2.2
2.3. Diseño	**51**	
2.3.1 Identificar y clasificar datos	3	2.2
2.3.2 Diseñar interfaz gráfica	27	2.3.1
2.3.3 Refinar diagrama de clases de diseño	5	2.3.2
2.3.4 Propuesta plataforma tecnológica	2	2.3.3
2.3.5 Validar diseño	14	2.3.4
2.4. Desarrollo del producto	**110**	
2.4.1 Programación	70	2.3
2.4.2 Pruebas	40	2.4.1
2.5. Puesta en marcha	**6**	
2.5.1 Implantación	3	2.4
2.5.2 Formación	2	2.5.1
2.5.3 Presentación final	1	2.5.2
3. Documentación y presentación del TFG	**200**	
3.1. Redacción de informes quincenales	22	
3.2. Redacción de la memoria técnica	140	
3.3. Presentación de la memoria técnica	7	3.2
3.4. Preparación de la presentación oral	30	3.3
3.5. Presentación oral	1	3.4

Figura 3.2: Desglose en tareas, junto con la planificación temporal y las dependencias entre tareas de la planificación final.

3.3.3. Diagrama de Gantt inicial

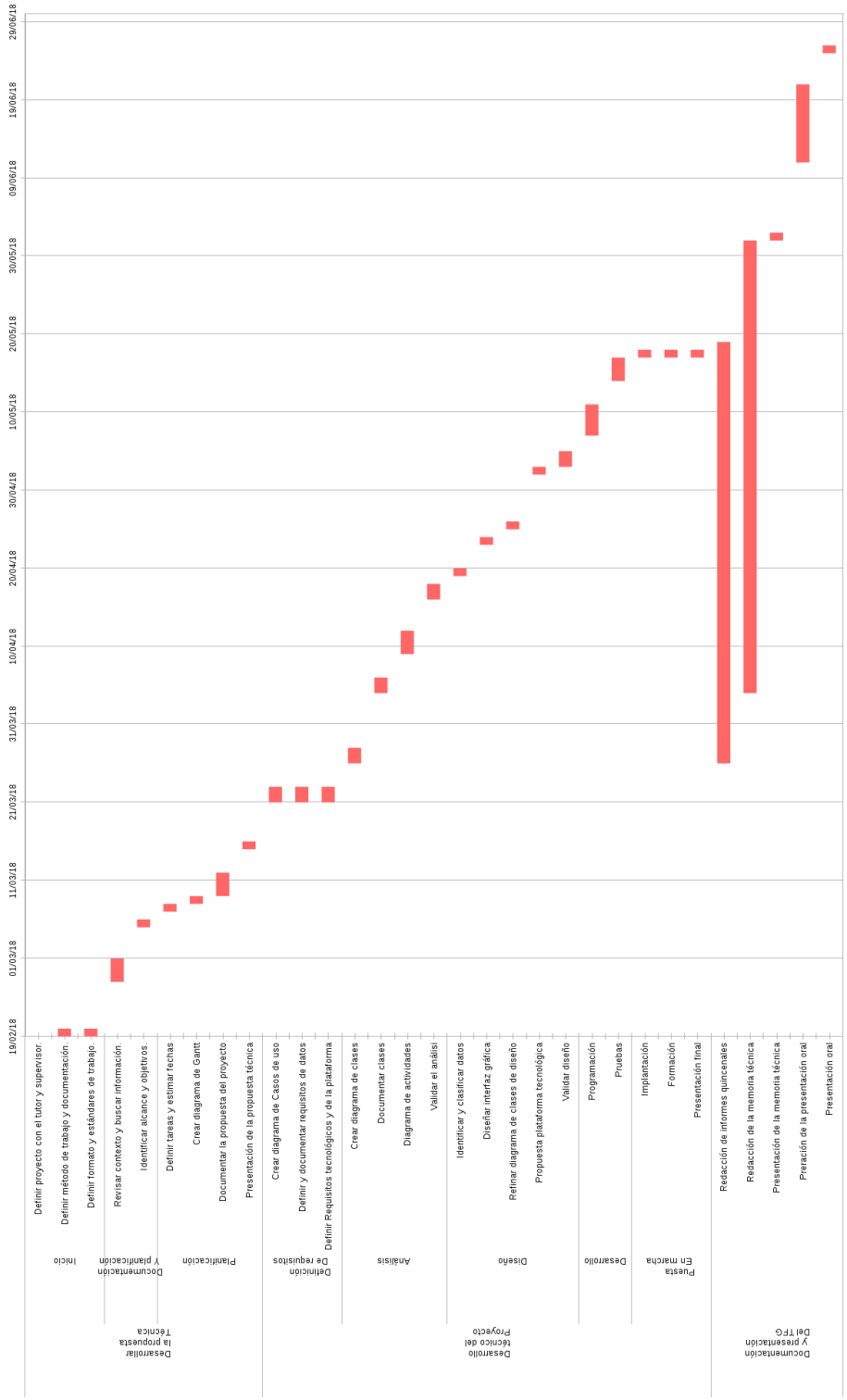

Figura 3.3: Diagrama de Gantt inicial indicando por días la división del trabajo.

3.3.4. Diagrama de Gantt final

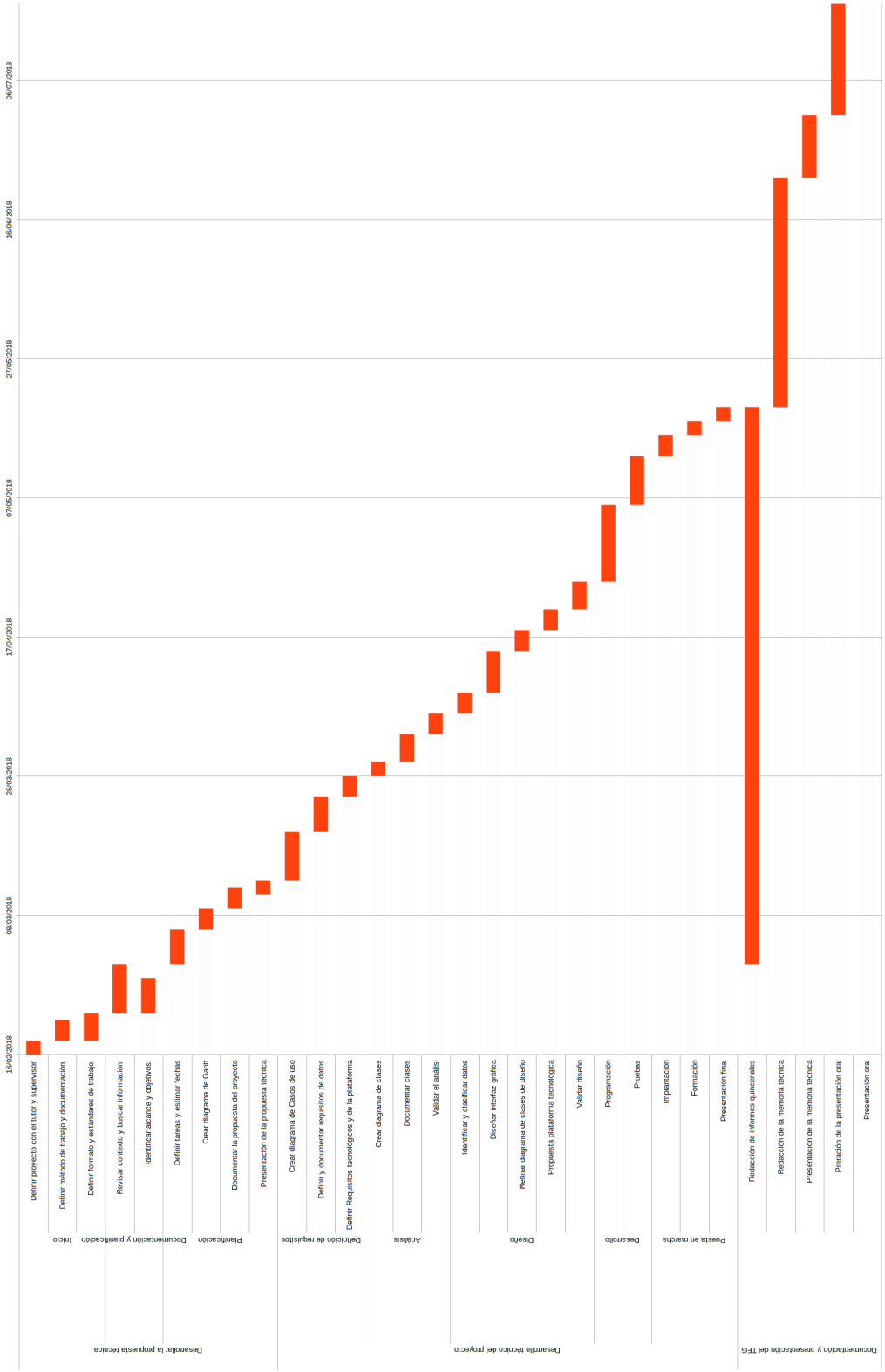

Figura 3.4: Diagrama de Gantt final indicando por días la división del trabajo.

3.3.5. Diferencias entre la planificación inicial y la planificación final

La planificación inicial del proyecto, tal como se puede ver en la Figura 3.1 ha sido bastante acertada al número de horas realizadas en cada tarea respecto a la planificación final que se muestra en la Figura 3.2 a pesar de los contratiempos ocurridos. Esto ha sido posible gracias a incluir en los ciertos puntos del proyecto que se consideraban críticos una mayor cantidad de horas respecto a otros puntos. En el diagrama de Gantt de la Figura 3.3 se puede observar el número inicial de horas que se iba a emplear para cada tarea. Por contra, se nota claramente un incremento de horas en determinadas tareas y reducción en otras tareas como se muestra en el diagrama de Gantt final de la Figura 3.4.

3.4. Estimación de recursos y coste del proyecto

Los recursos necesarios para desarrollar el proyecto han sido solamente aplicaciones de terceros o de la empresa de carácter gratuito, evitando así el coste económico por este proyecto. Por contra vamos a explicar cual seria el coste estimado del proyecto si hubiese que desarrollarlo partiendo de cero. Dos puntos que se han de aclarar para no dar confusiones son el coste de ciertos materiales, como puede ser el ordenador y el monitor, y el salario que hubiese percibido un trabajador, en función de la tarea asignada.

El precio del ordenador [17] de la tabla es de 599'00€ y del monitor [18] de 117'95€. Para tener el coste de estos productos durante el tiempo que se han necesitado, se ha contado con una duración de cada elemento de 60 meses. Para obtener el coste mensual se ha dividido el coste de cada producto entre 60 meses.

El cálculo del salario se basa en el salario mínimo interprofesional de un programador y un analista [22]. Se ha separado en dos tipos distintos de salarios, ya que la tarea de cada uno requiere unos conocimientos y aptitudes completamente distintos, creando diferencia entre los distintos salarios que pueda recibir uno u otro. El salario mínimo de un programador es de 1588€/mensuales, en cambio el de un analista es de 1924€/mensuales.

De otra forma en la Tabla 3.1 se encuentran todos los materiales y servicios necesitados, junto al precio, para poder evaluar el coste del proyecto.

Recurso	**Unidades**	**Coste**	**Total**
Ordenador HP Pavilion Power 570-P044NS [17](€/mes)	3	9'98€	29'95€
Monitor Samsung S24D330 [18](€/mes)	3	1'97€	5'90€
Ratón L-Link LL-2080-R [19]	1	1'95€	1'95€
Teclado Kloner KTU20 [20]	1	4'95€	4'95€
Netllar Internet 30 [21](€/mes)	3	19'99€	59'97€
Salario como programador(€/hora)	116	9'15€	1061'40€
Salario como analista(€/hora)	184	11'08€	2038'72€
Total			3245'59€

Cuadro 3.1: Tabla con los costes si el proyecto hubiese partido de cero.

En coste total del proyecto partiendo de cero sería de 3245'59€, siendo gran parte del presupuesto la mano de obra, puesto que los elementos de los cuales se necesitan no resultan muy costosos. Por la parte software todos los programas empleados son los del capítulo 2, siendo todos ellos de uso gratuito, tanto para particulares, como para empresas.

3.5. Seguimiento del proyecto

Todas las fases del proyecto han sido supervisadas por el tutor y el supervisor mediante informes quincenales donde están descritas las tareas realizadas durante cada quincena así como los objetivos propuestos durante la próxima quincena, los problemas encontrados durante el desarrollo y las tomas de decisiones ante estos problemas. Los problemas encontrados durante el desarrollo del proyecto también se encuentran explicados a lo largo de este documento, junto a una descripción de las acciones llevadas a cabo. No obstante en la lista que aparece a continuación se encuentran las incidencias más destacadas durante el desarrollo del proyecto:

- La creación de eventos ante el desconocimiento de éstos. No se tiene en cuenta que fueran necesarios para el desarrollo de la aplicación.
- Incompatibilidad entre el protocolo *modbus* y la biblioteca *NModbus4*. Al momento de almacenar la información, la biblioteca no almacena la información en las mismas direcciones.
- Mala configuración de los dominios de escucha al poner en marcha el esclavo *modbus*. La aplicación no se encuentra a la espera de escuchar en cualquier dirección, sólo en una en concreta.
- Mala implementación al desarrollar la aplicación. La aplicación no funciona de forma fluida cuando recibe muchas peticiones al estar mal planteada la implementación.
- Mala implementación del uso de la biblioteca de la empresa. No se tiene en cuenta el detalle que la biblioteca genera un desplazamiento en base 10.
- Fallos al detener las comunicaciones bruscamente. Esta prueba se hace para comprobar como actúa la aplicación cuando una comunicación no se cierra correctamente.

Capítulo 4

Análisis y diseño del sistema

4.1. Análisis del sistema

El requisito principal de la aplicación es suplir el funcionamiento del *PLC* de la impresora, para que no sea necesario depender de dicho componente al momento de realizar pruebas. Al estar destinado al usuario técnico, se da total libertad para el diseño de la interfaz de usuario. La comunicación con el dispositivo se hace a través de cable de par trenzado empleando el protocolo *mobdusTCP* mediante una red *ethernet*. Además, el código se ha de realizar en *C#*, junto con las bibliotecas *NModbus4*, *WPF*, *mvvmlight* y el protocolo interno propio que funciona sobre el protocolo *modbus*.

A partir de la lista de las metas que ha de cumplir el programa a desarrollar, se muestra la siguiente lista de las acciones que se han de llevar a cabo:

- Comunicación del dispositivo mediante el protocolo *modbus*
- Responder a las solicitudes, según dicta el protocolo interno de la empresa.
- Generar posibles resultados que se asemejen lo más posible a un *PLC* real.
- Generar una interfaz para el uso del programa
- Comprender el programa cedido por la empresa del cual se va a partir.
- No depender de herramientas físicas externas.

4.2. Diseño de la arquitectura del sistema

El sistema se ha diseñado en base a 4 elementos, mostrando la relación en la Figura 4.1:

- **Núcleo**: Es la parte que ayuda a comunicar las distintas partes del sistema.

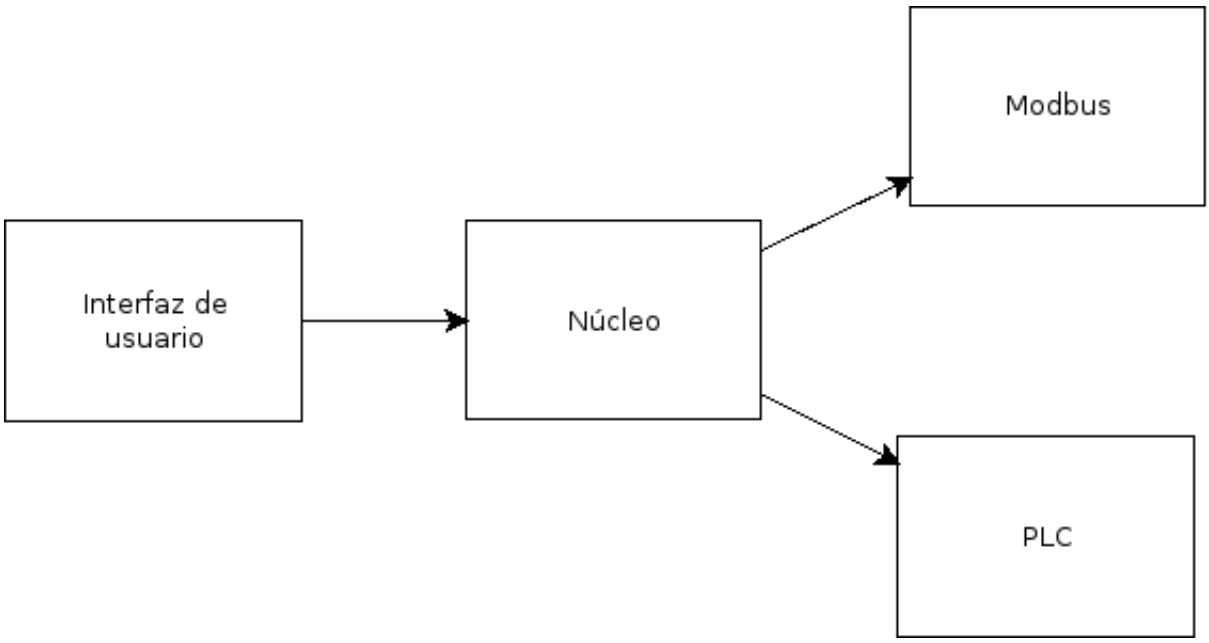

Figura 4.1: Dependencias entre los elementos de la aplicación.

- **Modbus**: Es la parte que se encarga de la comunicación con otras aplicaciones.
- **PLC**: Hereda del código de la empresa, adaptándose a las necesidades de nuestra aplicación.
- **Interfaz de usuario**: Es la parte de la comunicación con el usuario.

4.3. Diseño de la interfaz

Para realizar el diseño de la interfaz se buscó que cumpliera una serie de requisitos que se vieron importantes para una aplicación enfocada a un usuario técnico. Estos requisitos para la interfaz de usuario son:

- Buscar por identificador o nombre los campos que se desean visualizar.
- Separar las búsquedas en nuevas ventanas.
- Mostrar siempre la información actualizada en la pantalla.
- Hacer uso directo del teclado para la interacción con la aplicación.
- Modificar los datos rápidamente.

En el proceso del diseño de la interfaz se tuvo una primera idea de que el programa tuviese una interfaz visual mediante consola de comandos para el cumplimiento de los requisitos. El

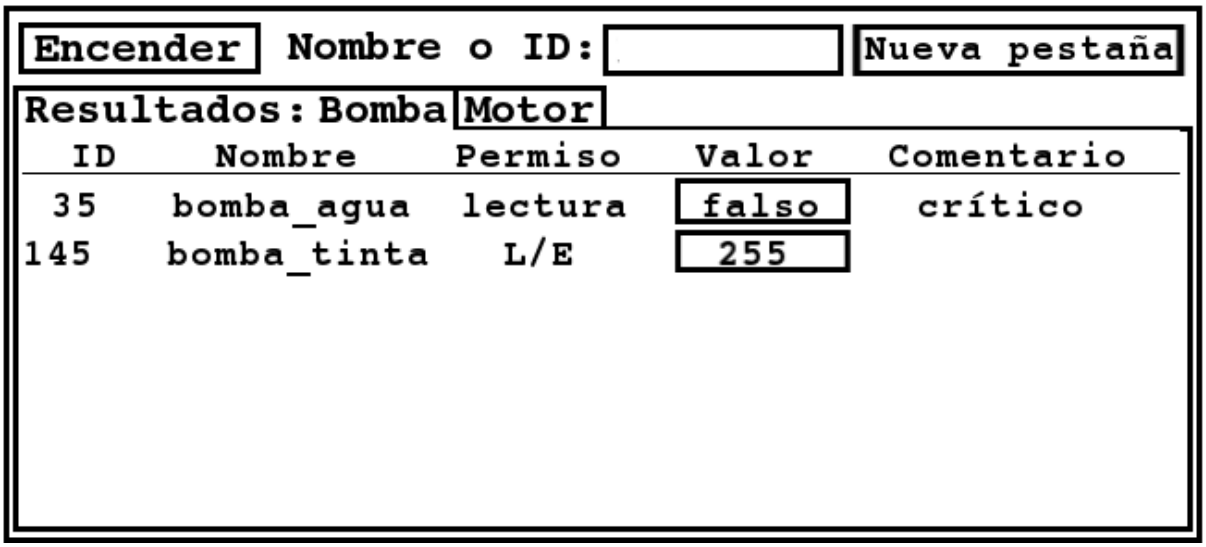

Figura 4.2: Edición después de comprobar que la complejidad de desarrollar la primera idea.

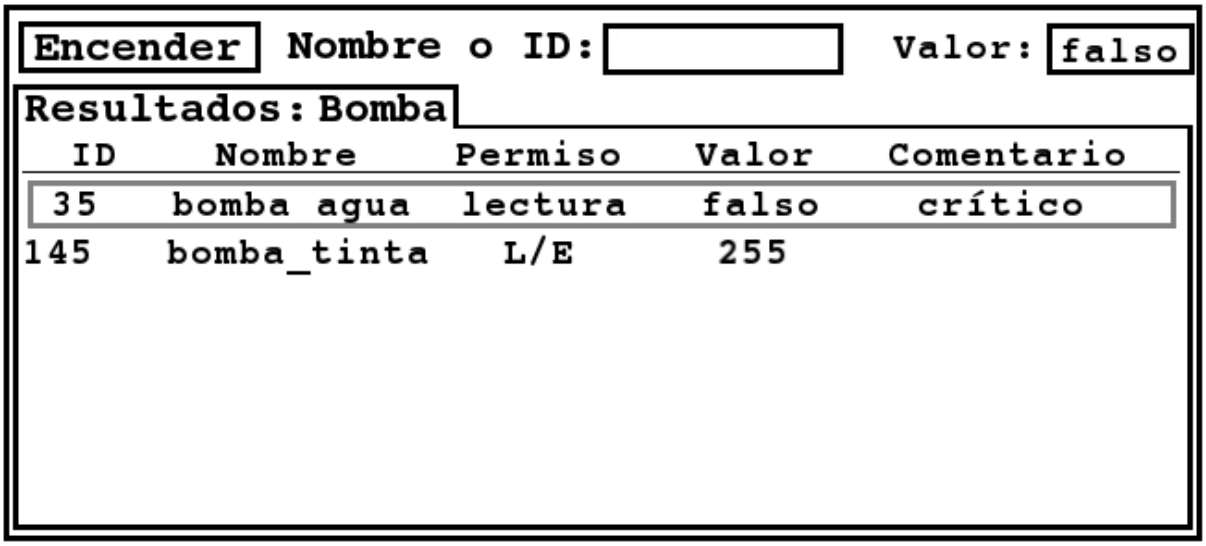

Figura 4.3: Edición después de comprobar que la complejidad de desarrollar la primera idea.

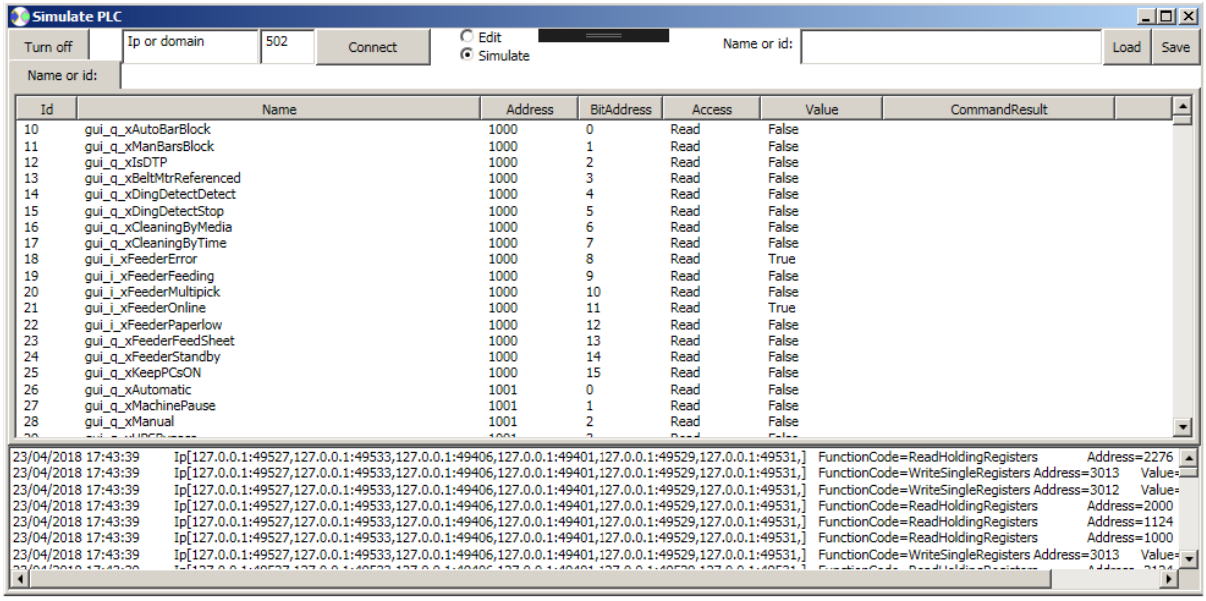

Figura 4.4: Interfaz final de la aplicación.

problema de este tipo de interfaz es la incapacidad de actualizar los cambios que ocurran durante la ejecución del programa con una sencilla implementación. Por ello, se tomó la decisión de emplear una interfaz gráfica intuitiva y sencilla, como aparece en la Figura 4.2 que cumpliese con los requisitos propuestos. La interfaz permite interactuar con ella mediante: un botón para activar/desactivar la escucha del simulador mediante *modbus*, una caja de búsqueda para filtrar los resultados que se muestran más abajo, un botón que genera una pestaña con los elementos filtrados de la caja de búsqueda.

Ante la incapacidad de poder realizar la funcionalidad que permite modificar los valores de los datos que se encuentran dentro de la lista filtrada en el tiempo establecido, se decidió hacer un cambio de la interfaz. Esto obligó a rediseñar la interfaz para que fuese posible desarrollarla en el menor tiempo posible, cambiando la opción de edición de cada línea de la lista a un campo nuevo estático en la parte superior de la aplicación, haciendo un poco más incomprensible y poco intuitiva la interfaz como se puede ver en la Figura 4.3.

El diseño presentado continuó dando problemas, aparte que se veía feo, sumado a que en la ejecución no realizaba un buen funcionamiento el programa, haciendo imposible editar los elementos por un mal diseño visual desarrollado. Este acontecimiento obligó a cambiar la forma en la que se visualiza el campo que modifica los datos de cada elemento. Esta vez se optó por separar la acción de edición de elementos en una ventana nueva, para así que no tengan relación las acciones de edición con el resto de la aplicación. Cabe destacar que en el momento que aparece esta nueva ventana, inhabilita toda acción que se pueda hacer en la ventana principal, para así evitar posibles errores de interacción.

Aparte de los cambios producidos por el problema de la edición de los datos, también se añadieron nuevas opciones de uso a la aplicación, así como un campo para indicar a la aplicación a qué *PLC* conectarse o dos botones donde que permiten almacenar y carga la información que

se encuentra en ese instante en la aplicación. También se añadió una zona donde se muestra las conexiones que recibe la aplicación, con la dirección y el motivo de la conexión, para así tener un mayor control de lo que está ocurriendo con el programa. Todos estos cambios se pueden observar en la Figura 4.4 con la interfaz ya terminada.

Capítulo 5

Implementación y pruebas

5.1. Detalles de implementación

En este apartado se va a conocer cuál ha sido el paso a paso del desarrollo de la aplicación.

5.1.1. Conocimiento de las tecnologías

La primera actuación en este proyecto ha sido documentarse acerca de las tecnologías que la empresa, así como conocer las capacidades, funcionalidades, la utilidad y cómo emplearlas. El primer movimiento llevado a cabo ha sido aprender el patrón de diseño *Modelo-Vista-Modelo de Vista* a través de la propia web de *Microsoft* [9]. Una particularidad que tiene este patrón es que es parecido al patrón *Modelo-Vista-Controlador* que enseña la universidad en los estudios del Grado. La siguiente tecnología en conocer ha sido el protocolo *modbus* empleado para la comunicarse con los *PLCs* por distintos medios de transmisión de datos. En concreto se va a emplear *modbus/TCP*, que es el más utilizado hoy en día por la gran cantidad de redes *ethernet* que existen actualmente consiguiendo un bajo coste de materiales, pudiéndose encontrar toda la información necesaria sobre dicho protocolo en su página web principal [13].

5.1.2. Desarrollo de un cliente *modbus/TCP*

Para la primera toma de contacto con dichas tecnologías, la empresa pidió que se hiciera un maestro *modbus* empleando la biblioteca *NModBus4*, que después se emplearía para la realización de pruebas con la aplicación final. La aplicación de prueba cuenta de una interfaz sencilla vía consola, la cual permite recibir la información del esclavo *modbus*, así como también modificar las posiciones indicadas. Podemos ver las opciones de funcionamiento que da la aplicación en la Figura 5.1.

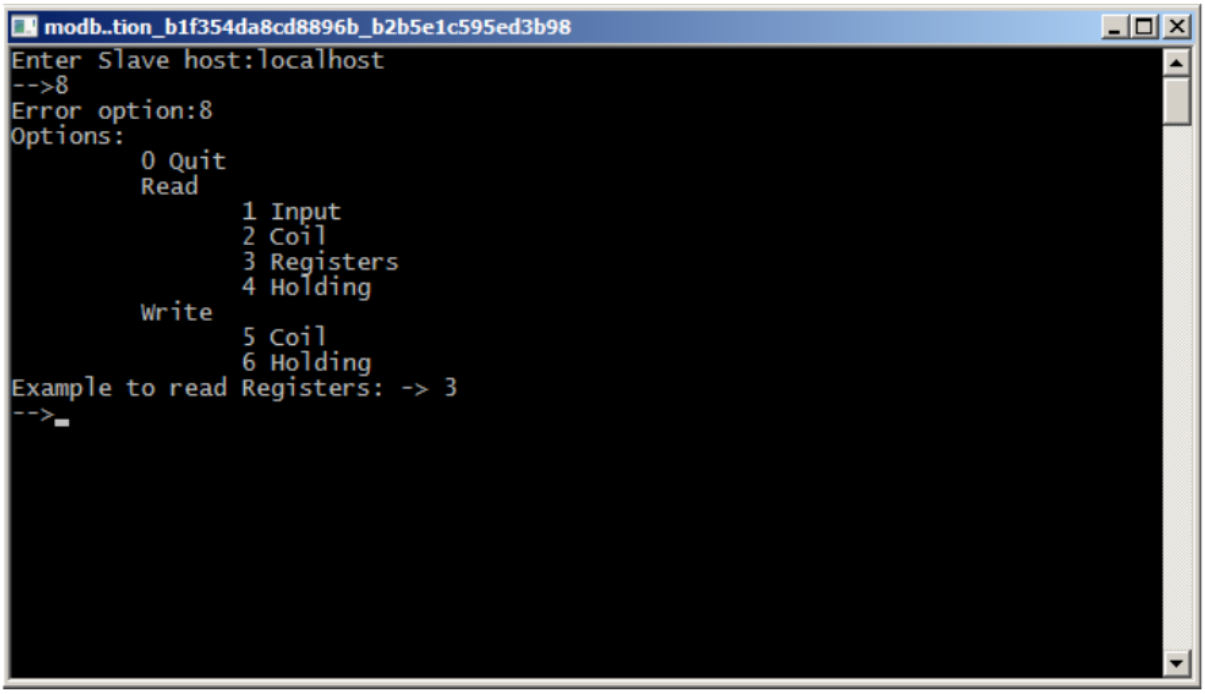

Figura 5.1: Opciones funcionamiento de la aplicación de prueba que hace la función de cliente *modbus*, abarcando todas las características del protocolo *modbus*.

5.1.3. Aprendizaje de las herramientas de la empresa

Llegado a cierto punto, la empresa decidió que sería conveniente ceder la biblioteca que se emplea para el protocolo interno de comunicación con la impresora, junto el código, brindando la oportunidad de conocer el funcionamiento del protocolo interno que emplea la empresa, siendo de gran ayuda en el futuro para el desarrollo del proyecto. Este código también dispone de su propia documentación donde explica el funcionamiento del *PLC*, ayudando a comprender cómo funciona la aplicación interna, junto con los campos y los requisitos que requiere para su correcto funcionamiento. Además la empresa realizó una exposición de los componentes más útiles de la biblioteca para el desarrollo del proyecto, ayudando a la comprensión.

5.1.4. Inicio del desarrollo del proyecto

En este punto de estancia en la empresa ya es posible empezar con el verdadero desarrollo del proyecto, disponiendo de los requisitos del proyecto de forma muy breve, los cuales fueron:

- El uso del protocolo *modbus*.
- El uso de la biblioteca interna.
- Visualización de los resultados.

- Estado del *PLC*.

Una vez ya se tienen los requisitos del proyecto, se pudo empezar con el diseño de la aplicación, separándose en cuatro partes diferentes:

- La interfaz visual.
- El apartado de comunicación con el *PLC*.
- Almacenaje de la información *modbus*.
- El intérprete del protocolo de la empresa.

También se aportaron ideas de nuevas funcionalidades que podría implementar la aplicación sin suponer un tiempo extra en su desarrollo. Estas características nuevas son:

- Añadir la opción de conectar la aplicación a un *PLC* real, realizando así la función de un supervisor intermediario que sea capaz de visualizar la información que envía y recibe entre el maestro y el *PLC*.
- Tener un registro de eventos de las conexiones realizadas al programa.
- Guardar los datos de la aplicación para poder ser cargados en la aplicación en un futuro.

Todas estas extensiones fueron propuestas en la revisión del diseño, siendo aceptadas permitiendo su desarrollo para la aplicación.

5.1.5. Diseño de la interfaz visual

Para el diseño de visualización de la aplicación se plantea inicialmente hacer un uso mediante consola, rechazándose inmediatamente esta propuesta al ver que el uso de una interfaz mediante consola no permite disponer de los datos actualizados en pantalla de una forma dinámica. Además, al tratarse de una aplicación para *Windows*, la consola no ofrece la misma potencia que en otros sistemas operativos, como tampoco permite una interacción inmediata. Por tanto se decidió hacer una aplicación con interfaz gráfica requiriendo una variedad de posibles diseños que puede abarcar la aplicación, como se puede ver en el capítulo 4.3.

5.1.6. Desarrollo de la etapa de almacenaje

El desarrollo de la aplicación empezó en el apartado de almacenaje de la información, al tratarse del punto central del cual dependerán el resto de elementos, además de requerir un formato especial que se asemeje lo más posible al protocolo *modbus*. Al hacer uso de la biblioteca *NModBus* esta parte del desarrollo se simplificó muchísimo al ya estar implementada y solo tener que prepararla para su uso con el resto de componentes creando funciones de lectura y escritura que facilitaran el acceso a la biblioteca.

5.1.7. Desarrollo de la comunicación modbus

El siguiente elemento a desarrollar del esquema de diseño es la comunicación, al pensar que es el apartado más difícil al tener que comunicarse con programas externos, siendo este trabajo facilitado principalmente por la biblioteca *NModBus4*. Una particularidad al emplear tanto el almacenamiento como la comunicación la biblioteca *NModBus4* es que dicha biblioteca ya está preparada para estas características lo que ha facilitado en gran medida el trabajo.

En este apartado principalmente nos hemos tenido que encargar de dar la opción de permitir que el programa abra la comunicación con el resto de programas o rechazarla, aparte de generar un evento que sea capaz de avisar en cada conexión recibida para el registro de eventos u otras necesidades.

5.1.8. Desarrollo a partir de la biblioteca de la empresa

A continuación se desarrolló el apartado que te tiene que ver con protocolo de la empresa, ya que una vez recibido y adaptado al caso de la aplicación que se desarrolla, realizando una implementación sencilla, surgió la problemática de ser necesario un fichero de configuración de la empresa requerido por la biblioteca.

La empresa cede un fichero de ejemplo *json* con el cual se puede continuar trabajando. En el fichero *json* se encuentran las posiciones *modbus* de acceso importantes para poder visualizar las posiciones que accede y modifica el programa. También dicho fichero contiene los permisos de cada posición (lectura, escritura y lectura/escritura), así como información que se pueda tener en cuenta a la hora de continuar desarrollar esta parte o tener en cuenta posibles errores que puedan aparecer. Este fichero es importante para no romper el sistema del *PLC*, ya que el *PLC* tiene la configuración de ese fichero. Se desarrollan a partir de la biblioteca dos partes distintas, una para que simule el *PLC* y la otra que se conecta al *PLC* para gestionar correctamente las conexiones. De esta forma el programa que se comunique con la aplicación desarrollada deberá emplear el mismo protocolo, dependiente del fichero de configuración.

5.1.9. Desarrollo de la interfaz de la aplicación

Se empieza con la herramienta gráfica que dispone *Visual Studio* para el desarrollo. Esto facilita mucho el trabajo para plasmar la idea gráfica que se tiene inicialmente, como vemos en el capítulo 4.3. La aplicaciones tienen botones de función, campos para rellenar y la lista de las posiciones a las que ha de acceder el programa, del cual parte del fichero *json*, un fichero donde se encuentra la configuración del protocolo de la empresa. Se tuvo un problema cuando se quería interactuar con la lista, ya que no se disponía del suficiente conocimiento como para que al pulsar encima de una fila de la lista se pueda modificar un elemento de ésta.

Al explicar este problema a mi tutor, me explicó que para eso debería utilizar una biblioteca llamada *MVVM Light Toolkit*, ya que es como se trabaja en la empresa. Por tanto se tuvo que empezar de nuevo la parte de la interfaz, empleando esta vez la biblioteca *MVVM Light Toolkit*, la cual no era muy distinto de la forma sin la biblioteca, simplemente añade facilidades que

ayudan a ahorrar tiempo.

El uso de esta biblioteca no ayudó a resolver el problema en la forma en la que se quería hacer. Lo que sí dio fue una solución alternativa y resolutiva para resolverlo por otra vía a la diseñada inicialmente que es la de crear una ventana emergente. Cada vez que se pulsa la fila del elemento el cual se quiere modificar, muestra todos los campos de dicho elemento y permite, si se da el caso, poder modificar los valores de dicho elemento.

5.1.10. Unión de todas las piezas, desarrollo del núcleo

Para unir todas las partes de la aplicación se ha empleado una clase que comunicará el almacenamiento de la aplicación con las distintas partes. Inicialmente se han encontrado muchos errores con el entorno visual de la aplicación, como:

- Botones que no funcionan correctamente.
- Cambios de estado incorrectos.
- La aplicación se cerraba al saltar excepciones no contempladas.
- Modificaciones de valores en lugares no correspondientes.
- Aparece más de una ventana emergente.
- Fallos en el apartado de conectividad sin ni siquiera utilizarlo.

Todos y cada uno de estos fallos que no permitían el correcto funcionamiento básico de la aplicación fueron resueltos, consiguiendo que la aplicación resultara estable sin realizar ningún tipo de comunicación por red.

5.1.11. Detección de errores de conectividad

Errores de la aplicación como esclavo

Para poder comprobar posibles errores de conectividad, se pasó a probar la aplicación con un maestro sencillo que emplea el protocolo *modbus*. El maestro *modbus* a utilizar es la aplicación de prueba desarrollada anteriormente. Se comprobó tanto con opciones de lectura como de escritura desde el maestro.

Cuando se hicieron este tipo de pruebas, se vio la necesidad de crear eventos en la parte de comunicación para que activamente actualice automáticamente la información cada vez que recibe una solicitud de escritura, para que de esta forma aparezca la información del programa en la interfaz del usuario instantáneamente conforme se recibe, cosa que no se tuvo en cuenta en su momento.

También se pensó que este avance es de gran ayuda cuando queramos que el programa haga de intermediario entre *PrintNuc* y el *PLC*, añadiendo la funcionalidad de enviar la información al esclavo, cuando se pide una modificación, puesto que antes la aplicación no tenía esta opción de desarrollo.

Errores de la aplicación como maestro

En este caso la aplicación se ha de conectar a un esclavo. Para ello empleamos la aplicación *ModbusPLCSimulator* que es de fácil manejo se dispone de conocimiento del protocolo *modbus*. Se puede encontrar más información de la aplicación en el capítulo 2.11.

Cuando activamos esta opción, la aplicación se pone en marcha automáticamente a escuchar en un puerto el esclavo *modbus*, mostrando los valores que tiene en ese instante el esclavo, actualizándolos cada vez que se modifica un valor.

De esta forma empezamos a comprobar que las limitaciones de escritura impuestas que no están permitidas se cumplen, comprobando que cada escritura y lectura las realiza correctamente, exceptuando por un error que ocurre cuando intenta modificar un valor de escritura que es de lectura.

El error detectado es debido a que en la biblioteca *NMobdus4* la forma de almacenar no empieza en cero, como hace el estándar del protocolo *modbus*, con lo cual, para solucionar el error se incrementa en uno, cada vez que se accede al dato, para que de esta forma la posición coincida con a la posición a la que se quiere acceder. Por eso generaba el error de intentar modificar un valor que no se corresponde con los permisos de lectura/escritura estipulados.

Errores de la aplicación como esclavo y maestro

En este caso se comprueba el uso de estos tres programas, la propia aplicación, el maestro *modbus* y el *ModbusPLCSimulator*. Después de realizar un conjunto de pruebas, el uso es aceptable y sin errores. Esto hace tomar la decisión de mostrar la aplicación al supervisor para comprobar que el funcionamiento es igual de correcto que con la aplicación propia.

5.2. Verificación y validación

5.2.1. Puesta en marcha con caso real

En la puesta en marcha de la aplicación junto con la aplicación de la empresa, observamos que la aplicación no se conecta, cosa que sorprende, ya que la otra aplicación de prueba sí que funciona. La empresa cede su aplicación, que denominaremos *PrintNuc*, para hacer pruebas sin depender del supervisor, para que pueda comprobar el por qué del no funcionamiento.

5.2.2. Solución de la problemática de conexión

Para el descubriendo de qué ha sido lo que ha podido fallar, se emplea la aplicación *Wireshark*, que podemos ver en el capítulo 2.12, para analizar las conexiones que realizan entre los programas y poder tener una idea de cuál es el fallo. Se descubre que el fallo es de interpretación del nombre de la máquina por una mala configuración, no abre los puertos en todas las direcciones que tiene la máquina en ese instante, solucionando este problema cambiando esta configuración de aplicación que estamos desarrollando.

5.2.3. Comprobación del correcto funcionamiento

Conexión de la aplicación como esclavo

En este punto se consigue conectar *PrintNuc* correctamente a la aplicación, con una gran cantidad de peticiones por segundo que apenas se pueden de ver debido a la gran velocidad que se generan y se reciben, sin generar ningún error a simple vista y comprobando que los valores que solicita leer y escribir el *PrintNuc* en cada identificador corresponde correctamente con los que se muestran en la aplicación.

Conexión de la aplicación como esclavo y maestro

Además de conectar la aplicación a *PrintNuc*, conectamos nuestra aplicación al *ModbusPLCSimulator* para comprobar que los valores son los mismos que envía *PrintNuc* y recibe *ModbusPLCSimulator* a través de nuestra aplicación como intermediario. Aquí se ve que algo raro está ocurriendo, la aplicación que se está desarrollando no funciona con suficiente fluidez, funcionando a tirones, con problemas en la comunicación, ya que no llega toda la información porque los tirones generan que se pierdan paquetes de conexión con *PrintNuc*, además de que en la aplicación *PrintNuc* se muestra que la conexión está desconectada.

Solución de errores de rendimiento

Empezamos a investigar a qué se debe la pérdida de paquetes. Se encontró que el problema que generaba el fallo es por un problema de diseño de la aplicación, ya que por cada petición que recibe, manda una gran cantidad de peticiones al esclavo para leer la información, una por cada elemento que muestra por pantalla, generando un importante cuello de botella en la red.

Para solucionar este problema, se cambia la forma en la que se refresca la información en la pantalla, para que así no tenga que hacer una petición para actualizar la información con cada elemento que se cambie el valor. A partir de este instante, la información que se muestra por la pantalla, no solicita la actualización de todos los datos al instante.

5.2.4. Comprobación de fluidez y comprobación de correcto funcionamiento

En esta nueva prueba se comprueba que el funcionamiento de la aplicación mejora en todos los aspectos de fluidez en los distintos modos de funcionamiento que tiene. Se vuelven a comprobar todos los parámetros de funcionamiento de las tres aplicaciones, con las distintas variables, comprobando que todo funciona, según los requisitos.

Cuando se comparan los datos que envía *PrintNuc* y son recibidos por *ModbusPLCSimulator* se observa que algunos datos son correctos, en cambio otros se encuentran como desplazados en potencias base 10 a la derecha o a la izquierda.

5.2.5. Solución de error de datos

Se empieza a analizar el problema comprobando qué cálculo hace la aplicación para hacer estos desplazamientos. Con la lectura del código de la biblioteca de la empresa, se descubre que hay un apartado que se encarga de esa función.

Como anteriormente se ha hecho, se vuelve a poner en marcha *PrintNuc* y *ModbusPLCSimulator*, para comprobar cuál es la diferencia entre la información inicial emitida y la final recibida, observando que la tarea del desplazamiento en base 10 se hace dos veces, una vez cuando la aplicación recibe el valor, almacenándose en nuestro programa desplazado y este mismo valor nuestra aplicación lo vuelve a desplazar para enviar al esclavo de nuevo.

Para solventar este error, se realiza la conversión de nuevo en formato que la biblioteca de la empresa pueda enviar correctamente la información al esclavo. De esta forma conseguimos corregir el error que provocaba la aplicación, enviando, ahora sí, correctamente los datos del programa a *ModbusPLCSimulator*.

5.2.6. Comprobación final

Se vuelve a hacer una comprobación exhaustiva de la aplicación, añadiendo nuevas pruebas de estabilidad. Estas pruebas de estabilidad consisten en hacer frente a una detención inesperada del maestro y/o el esclavo, cerrando forzosamente del sistema *PrintNuc* y/o *ModbusPLCSimulator* con las que mantiene una conexión.

Esta última comprobación ha generando en ocasiones que la aplicación fallara y en otras ocasiones no, descubriendo que se trata de un problema de salto de excepción al interrumpir el hilo en el que se encuentra ese momento la conexión. La solución a este problema ha sido que cada vez que se detectaba que la conexión se interrumpía bruscamente, finalizar el hilo correctamente para evitar que se siga cometiendo el mismo, además también se ha capturado la excepción para que no afecte al resto del sistema, de esta forma pueda continuar el correcto funcionamiento del programa.

Capítulo 6

Funcionamiento de la aplicación

6.1. Requisitos

Para su funcionamiento, la aplicación requiere de un fichero «**.json*» específico, puesto que este fichero contiene los parámetros de funcionamiento de la aplicación simulada, los mismos que va a emplear el maestro. Este fichero es escogido por defecto en la carpeta donde se encuentra la aplicación y en caso de no encontrarse, se solicita el fichero al usuario. Si no se selecciona ningún fichero, la aplicación se cierra.

6.2. Interfaz de la aplicación

Como se puede observar en la Figura 6.1, la aplicación dispone de una interfaz dinámica en la que se observan todos los cambios instantáneamente.

Con este aspecto se ha intentado que la aplicación sea lo más autodescriptiva posible, para su mayor facilidad de uso. Se puede ver que dispone de tres zonas claramente diferentes:

- La **zona superior** donde se consideran los elementos de control de la aplicación, en la cual se permite cambiar el funcionamiento de la misma.
- La **zona central** donde se encuentran los datos del *PLC* simulado o real y se permite editar dichos datos.
- La **zona inferior** la cual es un registro de las conexiones que han realizado otros maestros a nuestra aplicación mediante el protocolo **modbus**, que también se almacenan en un fichero.

A continuación se procederá a explicar qué función realiza cada uno de los botones de la zona superior de la aplicación, las cuales son:

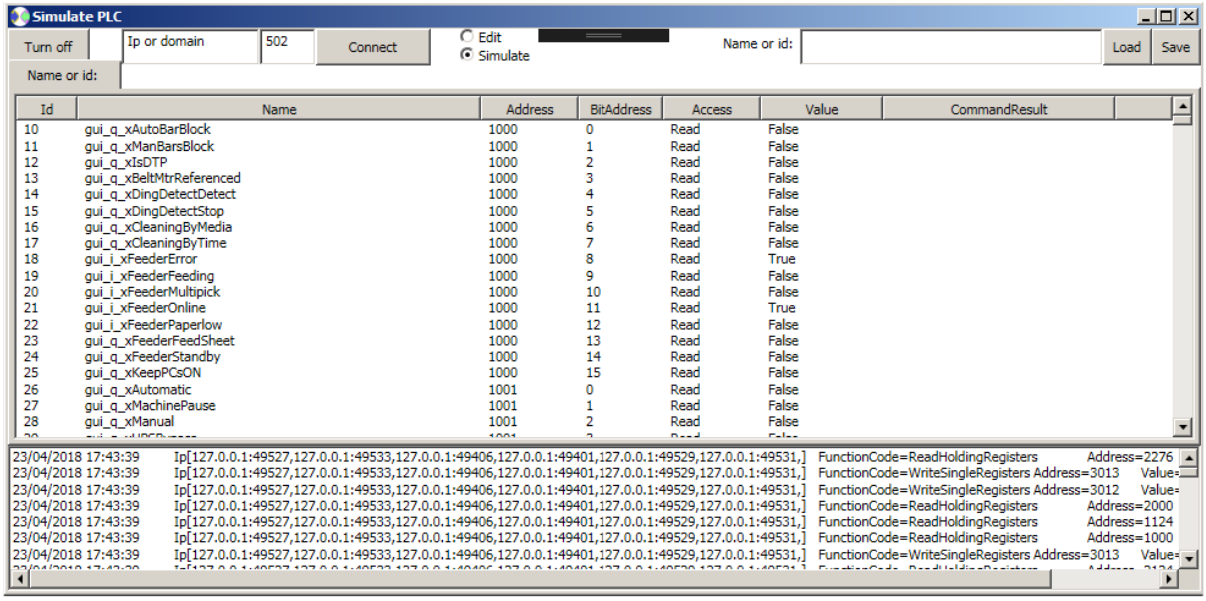

Figura 6.1: Aplicación en funcionamiento.

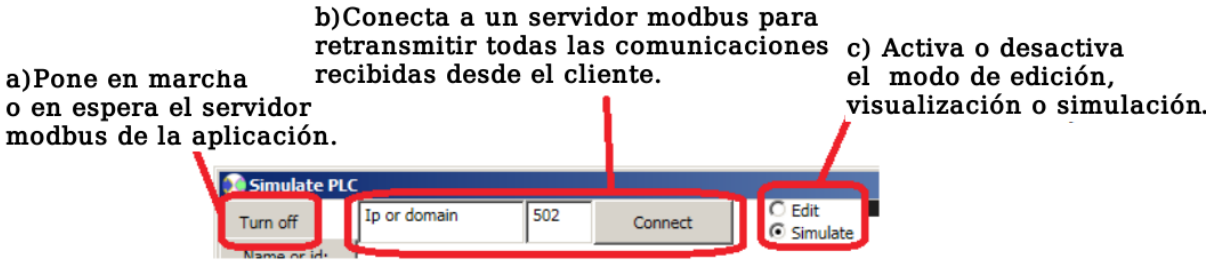

Figura 6.2: Funcionamiento de cada botón de la zona superior izquierda.

Identificar en las imagenes y en lalista

a) Poner en marcha o en espera el servidor *modbus* de la aplicación.

b) Conectar a un servidor **modbus** para retransmitir todas las comunicaciones recibidas desde el cliente.

c) Activar o desactivar el modo de edición, visualización o simulación de la aplicación.

d) Filtrar rápidamente por nombre o identificador.

e) Almacenar o cargar el estado de las variables de cada campo.

Estas explicaciones también se encuentran en la Figura 6.2 para la zona izquierda de control de la aplicación y en la Figura 6.3 para la parte derecha de la aplicación.

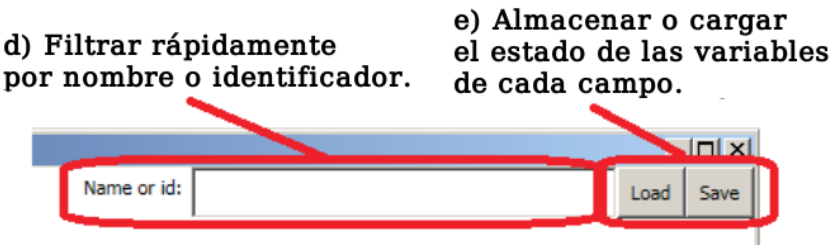

Figura 6.3: Funcionamiento de cada botón de la zona superior derecha.

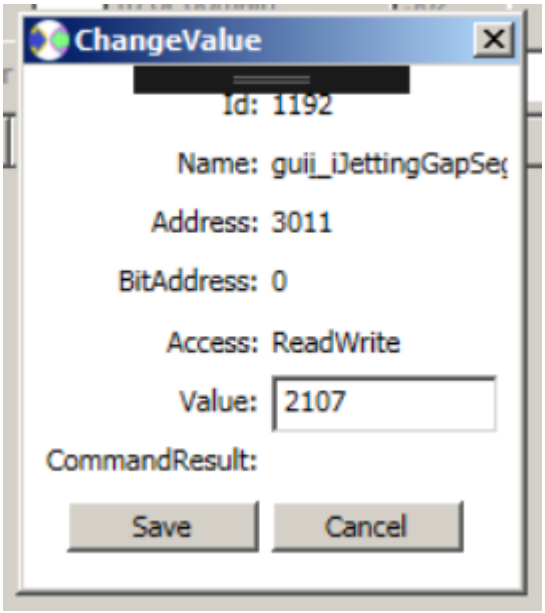

Figura 6.4: Esta ventana permite modificar los datos y ver las características del elemento.

Además cabe indicar que al pulsar cualquier dato en el apartado del centro, aparece una ventana que da la opción para la edición del valor si es posible en ese preciso instante. La ventana que se muestra es la de la Figura 6.4, y como podemos observar se da la opción de modificar el dato o de cancelar dicha modificación. Dicho valor cuando se trata de una variable booleana solo acepta «False» , «false» o «0» como valor negativo, considerando positivo cualquier otro valor.

6.3. Otras características a tener en cuenta

Hay que tener en cuenta que cada vez que se pone en ejecución el programa, la aplicación genera un fichero en la carpeta donde se encuentra el ejecutable. En este fichero se encuentra un registro de las conexiones que han realizado otros maestros a nuestra aplicación mediante el protocolo *modbus*.

Capítulo 7

Conclusiones

Durante el desarrollo de la aplicación, se han conseguido competencias que se desconocían, como bien puede ser la resolución rápida de conflictos, la documentación de la aplicación y las sinergias de trabajar con otros departamentos.

Además la aplicación ha conseguido evolucionar más sin la necesidad de emplear mucho tiempo en su desarrollo, añadiendo funcionalidades como puede ser realizar un volcado de datos a un fichero, tener un control de registros de eventos de las solicitudes recibidas y almacenarlas en un fichero, intermediario entre la aplicación y el *PLC*. No solo se han cumplido los objetivos básicos inicialmente establecidos, sino que además se han incluido otros que pueden ser de gran ayuda.

Sin embargo, en una futura revisión, se deberían corregir algunas carencias del sistema:

- Simulación del *PLC* más real.
- Añadir control de versión del *PLC*.
- Permitir almacenar los datos en la ubicación que el usuario desee.
- Permitir leer los datos en la ubicación que el usuario desee.
- Permitir cambiar el puerto de escucha por defecto.
- Permitir almacenar una copia del registro de eventos.
- Autoaprendizaje mediante inteligencia artificial para simular con la mayor precisión posible.

Bibliografía

[1] Comunidad wikipedia, *Controlador lógico programable*,

[2] Machine-Information-Systems.com, *What IS a PLC?*, http://www.machine-information-systems.com/PLC.html

[3] Ikujiro Nonaka y Hirotaka Takeuchi, *The New New Product Development Game*, 1986

[4] Atlasian, *página web*, https://es.atlassian.com/software/jira

[5] Douglas Crockford, *Internet JSON*, https://tools.ietf.org/html/rfc7493

[6] Microsoft, *página web*, https://visualstudio.microsoft.com/es/

[7] Microsoft, *página web*, https://docs.microsoft.com/dotnet/csharp/language-reference/

[8] Microsoft, *XAML Overview (WPF)*, http://msdn.microsoft.com/en-us/library/ms752059.aspx, 31 de marzo de 2018.

[9] Josh Smith, *WPF Apps with the Model-View-ViewModel Design Pattern*, https://docs.microsoft.com/dotnet/csharp/language-reference/, febrero 2009

[10] MVVM Light Toolkit group, *página web*, http://www.mvvmlight.net/doc

[11] Scott Chacon y Ben Straub, *Pro git*, https://git-scm.com/book/en/v2

[12] Modbus Organization, *MODBUS Protocol Specification*, http://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[13] Modbus Organization, *MODBUS TCP/IP*, http://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[14] Lin, Renee, *NModbus API Manual, versión 1.2* http://ftp.icpdas.com/pub/cd/8000cd/napdos/modbus/nmodbus/nmodbus_api_manual_v1.2_en.pdf

[15] zaphodikus, *About the Author* http://www.plcsimulator.org/Home/about

[16] The Wireshark team *Página web de Wireshark* https://www.wireshark.org/

[17] Ordenador HP Pavilion Power 570-P044NS https://www.pccomponentes.com/hp-pavilion-power-570-p044ns-amd-a10-9700-16gb-1tb-gtx-1050

[18] Monitor Samsung S24D330 https://www.coolmod.com/samsung-s24d330-24-led-monitor-precio

[19] Ratón L-Link LL-2080-R https://www.coolmod.com/l-link-ll-2080-r-negro-rojo-ratan-precio

[20] Teclado Kloner KTU20 https://www.coolmod.com/kloner-ktu20-usb-negro-teclado-precio

[21] Netllar Internet 30 https://www.netllar.es/internet-fibra-optica/

[22] Tusalario.es, *página web*, http://tusalario.es