RESUMEN DEL PROYECTO

FIN DE CARRERA

Arquitectura software y hardware para la automatización de una carretilla industrial

Departamento de Informática y Automática

Facultad de Ciencias

Universidad de Salamanca

Titulación: Ingeniería en Informática (2º ciclo)

Autora: Raquel Sánchez Díaz

Tutores: Belén Curto Diego y Francisco Javier Blanco Rodríguez

Fecha: Junio 2007

1. Introducción

En la actualidad, la automatización de los sistemas logísticos del almacenaje constituye un desafío fundamental que busca optimizar aún más el proceso productivo. Este trabajo se enmarca dentro de un proyecto de investigación más general cuyo principal objetivo es la automatización de una carretilla industrial, con el fin de obtener un sistema autónomo de transporte para realizar tareas de logística y almacenaje. La carretilla ha sido recientemente adquirida por el Departamento de Informática y Automática, y servirá como plataforma tanto para la prueba de resultados teóricos obtenidos previamente por el grupo de investigación del departamento, como para el desarrollo de posteriores labores de investigación sobre vehículos autónomos y la automatización de una gestión óptima del almacenaje.

El proceso de automatización de la carretilla es necesario plantearlo desde dos aspectos: la arquitectura hardware necesaria para la automatización de los movimientos y la arquitectura software que permita el control de los mismos.

Se deben integrar en la carretilla los elementos físicos necesarios para su movimiento controlado por ordenador, junto con los demás elementos de percepción del entorno por el que se mueve. Además se integrará un ordenador que recogerá las medidas de los sensores y ordenará las acciones a ejecutar. Para esta tarea se desarrollará una arquitectura que permita la interconexión de los diferentes elementos de tal modo que sea fácilmente escalable según aparezcan necesidades futuras referentes al sistema sensorial y de actuación.

Este proyecto se centra fundamentalmente en la parte relativa a los sensores y actuadores del sistema. Se plantea desarrollar y llevar a la práctica soluciones arquitectónicas software y hardware para la recogida de los datos sensoriales y el control de los actuadores.

2. Objetivos

2.1. Objetivos del proyecto

Los principales objetivos que se deben cumplir son:

1. Diseñar y desarrollar una arquitectura software y hardware para el control de los sensores y actuadores de la carretilla.
2. Utilizar el bus I2C para comunicar estos dispositivos con los microcontroladores que se encarguen de su control.
3. Utilizar el bus CAN para comunicar todos los microcontroladores entre ellos. Para ello será necesario estudiar sus características y funcionamiento, y realizar una documentación técnica sobre él para su uso en desarrollos posteriores.
4. Desarrollar software para que los microcontroladores que gestionen sensores y actuadores cumplan los siguientes requisitos:
    - ✓ Sean capaces de adaptar su comportamiento, es decir, que se puedan configurar (las direcciones de los dispositivos, los temporizadores...) desde una aplicación de control en el PC.
    - ✓ Informen a esa aplicación sobre sus funciones, su estado y el estado de sus dispositivos.
    - ✓ Si controlan sensores:
        - Lean continuamente los datos que éstos recogen mediante el bus I2C.

        - Realicen una lectura eficiente de los dispositivos.
        - Envíen las mediciones por el bus CAN de forma periódica, o bajo demanda.
        - Envíen alertas si las mediciones obtenidas no se encuentran dentro de los límites establecidos.
    - ✓ Si controlan actuadores:
        - Envíen a sus dispositivos las órdenes que les lleguen por el bus CAN.
5. Desarrollar una interfaz gráfica que permita al usuario manejar y controlar todo el sistema desde el PC de manera que:
    - ✓ Detecte automáticamente los microcontroladores conectados al bus CAN.
    - ✓ Reciba los datos tomados por cada sensor.
    - ✓ Envíe órdenes o instrucciones a los actuadores.
    - ✓ Conozca en todo momento el estado de los elementos del sistema: si están funcionando correctamente o no.
    - ✓ Sea capaz de configurar los parámetros de los microcontroladores que determinen características de su funcionamiento: las direcciones de los dispositivos, cada cuánto tiempo transmiten los datos de los sensores, cuándo deben enviar alertas, etc.
6. Implementar algún protocolo que especifique cómo se va a realizar la comunicación entre los distintos elementos conectados al bus CAN.
7. Desarrollar todos los programas necesarios para la correcta comunicación entre las diferentes partes del sistema.
8. Documentar adecuadamente los conceptos teóricos relacionados con el sistema, las bibliotecas de funciones y los programas que se desarrollen.

2.2. Objetivos personales

El principal objetivo personal es el de superar el reto que supone el aprender, comprender y poner en práctica todos los conocimientos nuevos necesarios para realizar este proyecto, ya que nunca antes había trabajado con este tipo de sistemas. Me resulta muy interesante la experiencia de programar un PIC por primera vez, y hacer que se comunique mediante los buses I2C y CAN, los cuales nunca había utilizado.

También el hecho de programar a tan bajo nivel, por las dificultades tanto de implementación como de depuración que conlleva, además de tener que aprender las peculiaridades del compilador que voy a utilizar para ello. Espero ser capaz de idear soluciones para resolver todos los problemas que vayan surgiendo durante el desarrollo.

3. Técnicas y herramientas

3.1. Técnicas metodológicas

En el subsistema de los microcontroladores se ha utilizado la programación estructurada, ya que es la forma de implementar este tipo de programas. En cambio para realizar la interfaz gráfica de usuario nos hemos decantado por el paradigma objetual, ya que creemos, se ajusta mejor a nuestras necesidades.

3.2. Lenguajes de programación

3.2.1 C

Se ha elegido el lenguaje C para desarrollar los programas de los microcontroladores porque además de ser un lenguaje indicado para la programación a más bajo nivel, ya que permite realizar fácilmente este tipo de operaciones, existen multitud de compiladores que generan el código en ensamblador, lo cual hace mucho más cómoda y fácil la programación.

3.2.2 Java

Tras barajar varias opciones, Java (versión 1.5) fue el lenguaje de programación elegido para desarrollar la parte de interfaz gráfica del proyecto, ya que cumple dos requisitos fundamentales: se basa en la metodología orientada objetos y es multiplataforma.

También ha sido necesario el uso del **JFC** y **Swing** para construir la interfaz gráfica. Para poder acceder al puerto serie nos hemos decantado por utilizar la **RXTX**, que es una biblioteca de funciones nativa que proporciona acceso a los puertos serie y paralelo para el JDK. Es compatible con la API Comm de Sun. Está disponible para una amplia gama de plataformas y es gratuita. Nosotros hemos utilizado la RXTX 2.1 para usar sin la API Comm (espacio de nombres *gnu.io*).

3.3. Entorno de desarrollo

Durante el proceso de desarrollo se han utilizado varias herramientas, en función de las características de cada una. Para realizar los programas en C para los PICs se ha utilizado el entorno de desarrollo **SourceBoost IDE** con su compilador **BoostC**. Para la interfaz en Java se ha optado por **Eclipse**.

Para cargar los programas en los microcontroladores se ha utilizado la programadora **PICkit 2** y el adaptador **PGM2KIT**, ya que nos permitían programar al PIC en el circuito definitivo a usar.

Ha sido necesario el uso de otros programas como el **232Analyzer** para la depuración del software de los PICs, **Doxygen** para realizar la documentación, y **Visual Paradigm**, **COCOMO II** y **Microsoft Proyect 2003** para las tareas de ingeniería del software.

3.4. Hardware y Sistema Operativo

3.4.1 Puesta en funcionamiento del sistema

En el sistema se pueden diferenciar dos partes: la programación de los PICs y la interfaz gráfica de usuario, ejecutada en el PC. La parte de interfaz de usuario se distribuye en forma de un único fichero *PicController.jar*. En ese archivo *jar* está contenido todo lo necesario para que la aplicación funcione. También está disponible un manual de usuario que explica las partes principales de la herramienta y sus principales características funcionales, además de cómo utilizarla correctamente.

Los únicos requisitos de software necesarios en el PC para que la interfaz funcione son el tener instalado un JRE 1.5 o superior y la biblioteca RXTX para comunicarse por el puerto serie. La aplicación funciona tanto en Windows como en Linux, ya que Java y RXTX son multiplataforma. Las pruebas del sistema completo sólo se han realizado en Windows debido a

que ciertas partes del software de programación para PICs sólo están disponibles para esa plataforma.

Para el subsistema de los microcontroladores se necesitan: los PIC18F258 (en las placas SBC28PC) conectados por el bus CAN, los sensores y actuadores (en nuestro caso los SRF08) comunicados con su PIC mediante el bus I2C, el PIC que realiza las conversiones entre serie y CAN, el conversor VScom USB-COM-I conectado tanto al PIC anterior por la interfaz RS485 como al PC por el USB, los programas para los PICs, y una fuente que los alimente.

En la siguiente figura podemos ver un ejemplo de la interfaz cuando el sistema está en ejecución. En este caso está conectado tan sólo el nodo que controla los sónares. Este nodo solamente tiene un dispositivo SRF08 conectado con la dirección 0xE0, y está recogiendo las medidas que va tomando:

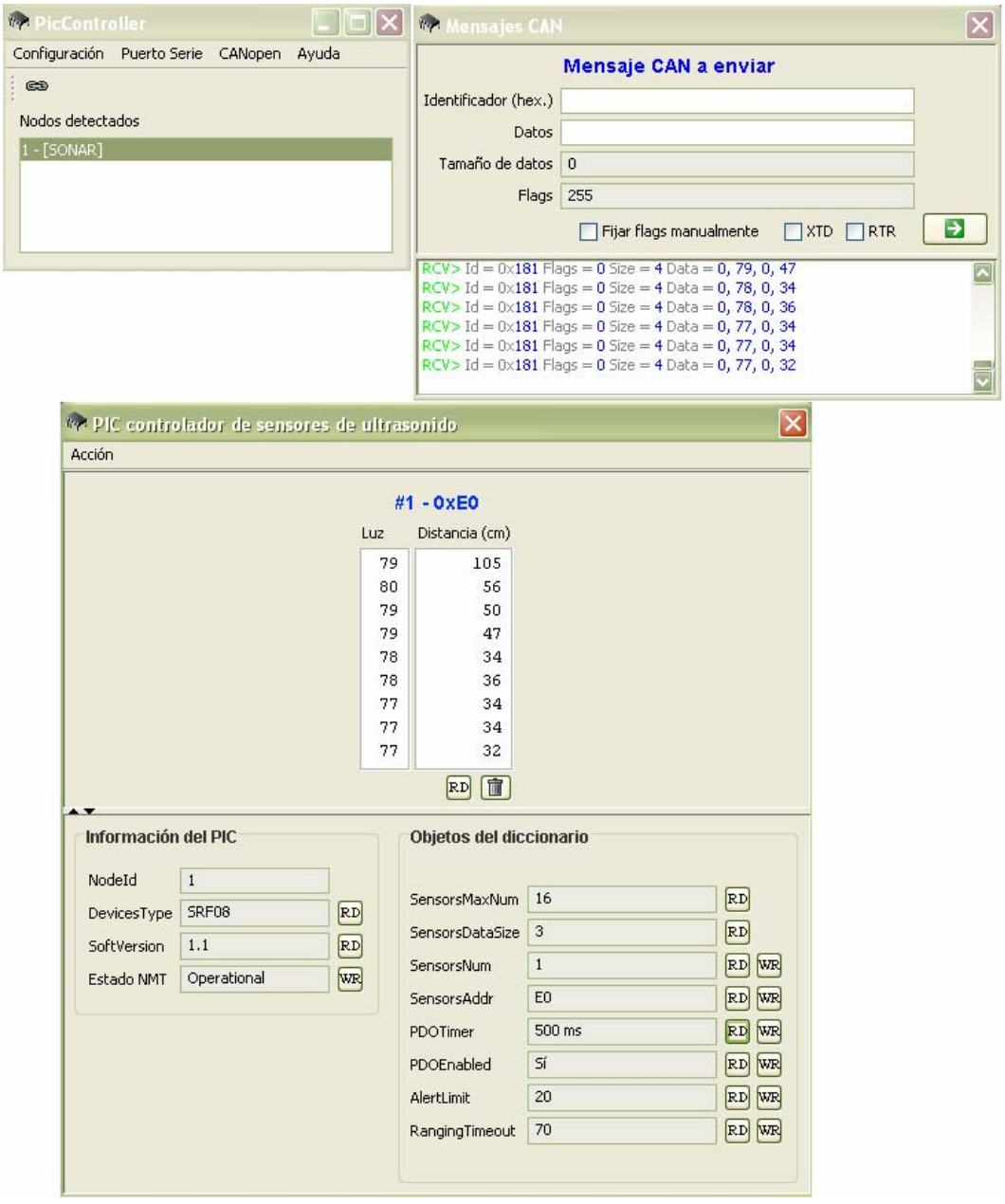

Figura 1: Ejemplo del sistema en ejecución

4. Aspectos relevantes del desarrollo

4.1. Descripción general sistema

Se ha diseñado una arquitectura en capas para que se pueda utilizar cualquier tipo de sensores y actuadores, sin tener que realizar cambios relevantes. El protocolo de comportamiento del sistema no se verá en absoluto modificado, ya que las interfaces para acceder a los servicios de cada capa están bien definidas y son independientes de los dispositivos.

Los sensores y actuadores están controlados por microcontroladores PIC, que se comunican con ellos mediante el bus I2C. A su vez los microcontroladores están comunicados entre sí mediante el bus de campo CAN.

Debido a que el bus CAN sólo define las capas física y de enlace, es necesario utilizar algún protocolo que implemente las capas superiores y especifique cómo se va a realizar la comunicación entre los distintos elementos conectados al bus. Para este fin se utiliza el protocolo **CANopen**, por estar ampliamente extendido y haber sido adoptado como un estándar internacional.

CANopen se ajusta perfectamente a nuestras necesidades ya que nos proporciona un conjunto de protocolos de comunicación y servicios necesarios para implementar un sistema de control distribuido. Se pueden diferenciar dos papeles principales: el de maestro y el de esclavo. En cada sistema habrá un único maestro y uno o más esclavos.

El maestro tiene un comportamiento diferente porque es el que se encarga de toda la gestión y el único que conoce a todos los nodos que están conectados al sistema. En nuestro caso este papel lo realiza la aplicación de control que se ejecuta en el PC.

Los esclavos (representados por los PICs) no se conocen entre ellos, ya que siempre se van a comunicar con el maestro. Podemos distinguir dos grandes grupos que son los que van a marcar la principal diferencia en el comportamiento: los esclavos que controlen sensores y los que controlen actuadores. Todos los nodos esclavos van a actuar de una manera similar dependiendo de a cuál de estos dos grupos pertenezcan.

Todo el sistema se controla desde un PC, por lo que es necesario que de alguna manera éste tenga acceso al bus CAN. Para ello hay que introducir un elemento intermedio entre el bus y el ordenador, que se encargue de realizar las transformaciones necesarias. Aunque en la arquitectura final se incluirá un adaptador de USB a CAN que realice estas tareas, provisionalmente se ha optado por resolver el problema de la siguiente manera. Hay un PIC adicional conectado al bus CAN cuya única finalidad es la de enviar por el puerto serie, en un formato determinado, los mensajes que le lleguen por el bus CAN, y al revés, es decir, transformar en mensajes CAN la información que le llegue por el puerto serie desde el PC. De esta forma se introduce una capa de abstracción para que el programa de control que se está ejecutando en el PC, y que actúa como maestro CANopen, pueda comportarse como un elemento más conectado al bus, cuando en realidad su comunicación se realiza por el puerto serie.

4.2. Descripción de los elementos del sistema

En el sistema hemos utilizado los siguientes buses, protocolos y dispositivos:

- Microcontroladores **PIC18F258**: Se han utilizado estos dispositivos para controlar los sensores y actuadores. Adicionalmente uno de ellos se encarga de realizar las conversiones entre bus CAN y serie, para que sea la posible la conexión con el PC.

- Placa **SBC28PC**: Sobre este tipo de placa se montan los PICs.
- Sensores de ultrasonidos **SRF08**: Con este tipo de sensores se han realizado todas las pruebas del sistema.
- Bus **I2C**: Se utiliza este tipo de bus para realizar las conexiones entre los sensores y actuadores y el PIC que los controla.
- Bus **CAN**: Todos los microcontroladores están conectados entre sí mediante este bus de campo. Debido a que el ordenador no está conectado directamente al bus CAN se utiliza un microcontrolador que actúa como mediador entre el PC y el bus, de manera que aunque el ordenador se comunica por el puerto serie, se comporta como si fuera un dispositivo más del bus.
- **CANopen**: Es el protocolo elegido para las comunicaciones entre los elementos del bus CAN. Los microcontroladores implementan los esclavos y en el PC se encuentra el maestro.
- Conversor **VScom USB-COM-I**: Es un conversor de USB a serie utilizado para la comunicación serie entre el PC y el PIC que realiza la conversión a CAN.
- **PC**: En él se ejecuta la aplicación que controla todo el sistema y actúa como maestro CANopen. Se comunica con el PIC conversor mediante el puerto serie, utilizando el conversor anterior de USB a serie.

4.3. Arquitectura del sistema

Se ha diseñado una arquitectura en capas para organizar los elementos descritos anteriormente según sus responsabilidades. Esto nos permite tener cierta abstracción y más independencia. De esta manera podemos realizar cambios en las capas sin que afecten a las demás, a excepción de si se modifica la interfaz que ofrece para acceder a sus servicios.

La arquitectura se organiza en cuatro niveles:

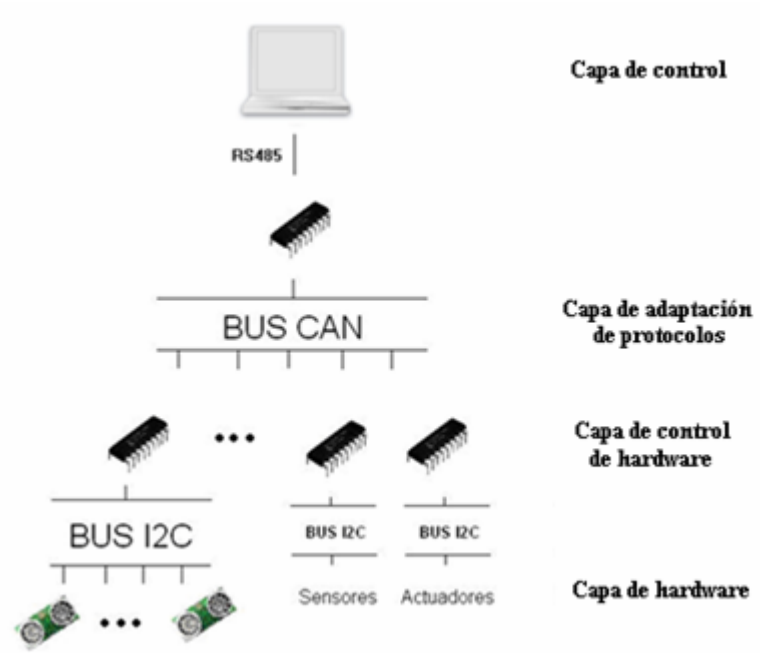

Figura 2: Arquitectura en capas del sistema

4.3.1 Capa de control

Es la raíz de la jerarquía y se corresponde con un PC. En él se ejecuta una interfaz gráfica desarrollada en Java para que el usuario pueda manejar y controlar todo el sistema. Esta aplicación actúa de maestro CANopen. Desde ella se puede acceder fácilmente y de una manera intuitiva a los servicios que este protocolo nos proporciona. Está internacionalizada y disponible en inglés y en español. Es configurable por el usuario y permite almacenar y modificar las opciones de configuración de los nodos del sistema.

Posee una biblioteca de clases que implementa a un maestro CANopen. Contiene módulos que gestionan cada una de sus partes y protocolos: NMT, *nodeguading*, PDO, SDO, mensajes de emergencia y alertas, el diccionario de objetos y los mensajes CAN. Para un funcionamiento más eficiente se han utilizado distintos hilos y por lo tanto mecanismos de sincronización para el control de accesos concurrentes. También existe un semáforo para poder realizar espera bloqueante sin consumir ciclos de CPU.

La aplicación se comunica con el resto del sistema mediante mensajes CAN pero como el PC no está directamente conectado al bus, contiene un módulo que realiza la conversión de estos mensajes a datos por el puerto serie, con un determinado formato. Por lo tanto, la comunicación con la capa inmediatamente inferior se realiza mediante la interfaz RS485.

Los tres módulos fundamentales: biblioteca CANopen, puerto serie e interfaz gráfica, son independientes unos de otros, ya que se han introducido capas de abstracción para lograr el desacoplo entre las distintas partes. Para ello se utiliza el patrón Modelo-Vista-Controlador.

4.3.2 Capa de adaptación de protocolos

Está formada por un PIC que realiza la conversión entre datos serie y mensajes CAN. Se comunica con la capa de control mediante la interfaz RS485 y con la capa inmediatamente inferior por el bus CAN. Realiza las conversiones de información en ambos sentidos. Los datos serie se transmiten en un determinado formato para poder extraer las distintas partes del mensaje CAN que contienen. Un autómata finito, situado en cada extremo (PIC y PC), se encarga de reconocer las tramas completas y correctas, y reconstruir los mensajes CAN originales.

4.3.3 Capa de control de hardware

En ella están todos los PIC del sistema que controlan los sensores y actuadores. Realizan el trabajo de un esclavo CANopen. Se comunican con los dispositivos de la capa inmediatamente inferior mediante el bus I2C, en el que el PIC actúa de maestro.

Se ha diseñado e implementado de manera general una biblioteca CANopen para nodos esclavos, siendo válida para todo tipo de PICs, e independiente de los dispositivos que se conecten al nodo. Tan sólo será necesario realizar pequeños cambios, que están debidamente documentados, y que no afectan en absoluto al protocolo ni al comportamiento general, para poder manejar características específicas de cada dispositivo. La biblioteca contiene diferentes módulos que gestionan: los mensajes de NMT, de *nodeguarding*, de emergencia y alerta, la transferencia de PDOs y SDOs, el diccionario de objetos y una capa de abstracción que la separe de las peculiaridades del PIC para la transmisión y recepción de mensajes CAN. También se ha desarrollado un controlador, ya dependiente del PIC, que se encarga de la comunicación por el bus CAN.

4.3.4 Capa de hardware

Es la de más bajo nivel. En ella se encuentran los sensores y actuadores. Estos dispositivos se encargan de la interacción con el exterior. Los habrá de diversos tipos, según el propósito que deban cumplir: sónares, cámaras, motores, etc. Todos los elementos de esta capa son esclavos

de un bus I2C, por el que se comunicarán con el PIC de la capa superior que se encargue de su control.

5. Conclusiones

Tras finalizar este proyecto hemos comprobado que se han cumplido satisfactoriamente todos y cada uno de los objetivos que teníamos. Se ha desarrollado una arquitectura software y hardware para la recogida de los datos sensoriales y el control de los actuadores.

Durante el desarrollo se han ido adquiriendo los conocimientos y experiencias necesarios para crear cada parte del sistema, dando como resultado una arquitectura en capas flexible y bien definida, así como un software robusto y de calidad.

Como protocolo de nivel superior que especifique la manera de realizar la comunicación entre los distintos elementos conectados al bus se ha elegido e implementado CANopen, ya que está ampliamente extendido y ha sido adoptado como un estándar internacional. Debido a que la documentación que existía estaba en inglés y no resultaba muy clara, se ha realizado una en castellano que será utilizada en desarrollos posteriores.

Se ha desarrollado una biblioteca de funciones que implementa en C un esclavo CANopen. Esta biblioteca puede reutilizarse en cuantos esclavos CANopen sean necesarios, independientemente del tipo de dispositivos que controlen. Tan sólo habrá que hacer pequeñas modificaciones para añadir características específicas de cada modelo de sensor o actuador. Cómo y dónde realizar estos cambios está explicado con detalle tanto en el código fuente directamente, como en la documentación de la API entregada en formato digital, y en el anexo de “Documentación técnica de programación” de la memoria.

Adicionalmente se ha implementado en Java el maestro CANopen, el cual se encarga del control del sistema. Desde la interfaz gráfica el usuario puede monitorizar e interaccionar con el sistema. Puede comprobar qué nodos están conectados al bus CAN, qué tipo de dispositivos controlan, cuál es su estado, si están funcionando correctamente tanto ellos como sus dispositivos, puede enviar órdenes y modificar parámetros del diccionario de objetos.

La interfaz abstrae al usuario de las peculiaridades del bus CAN y CANopen, siendo sencilla e intuitiva, pudiendo ser utilizada por cualquier persona que no conozca los protocolos. Pero a su vez permite a los usuarios con un cierto grado de conocimientos saber qué es lo que está pasando en el sistema en cada momento, ya que toda la comunicación tanto por el bus CAN como por el puerto serie se encuentra monitorizada, y existe la posibilidad de enviar directamente mensajes CAN, u otros datos por el puerto serie.

También se ha implementado la capa intermediaria entre los dispositivos del bus CAN, y el PC por el puerto serie. De esta manera el sistema se comporta como si la aplicación de control que está en el PC y realiza las labores de maestro CANopen fuera un nodo más del bus CAN.

Se ha realizado una documentación completa tanto de los conceptos teóricos relacionados con el desarrollo, como de las bibliotecas de funciones y programas implementados, para su reutilización o ampliación en el futuro.

En lo que se refiere al ámbito personal también se han cumplido mis objetivos, ya que he sido capaz de asimilar en poco tiempo muchos conceptos, y llevarlos a la práctica con éxito. Además he logrado superar las dificultades que entrañan la programación a tan bajo nivel, la depuración de los programas de los PICs, y el comprender e implementar un protocolo tan amplio y peculiar como CANopen a partir de la documentación existente. Finalmente ha sido una experiencia muy gratificante y que me ha aportado muchos nuevos conocimientos.