UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA DE TELECOMUNICACION

PROYECTO FIN DE CARRERA

“Control a través del Universal Serial Bus de motores paso a paso”

ESPECIALIDAD: TELEMÁTICA

AUTOR: RUPERTO J. HERNÁNDEZ SÁIZ

TUTOR: DOMINGO MARRERO MARRERO

FECHA: OCTUBRE 2004

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA DE TELECOMUNICACION

PROYECTO FIN DE CARRERA

“Control a través del Universal Serial Bus de motores paso a paso”

Fdo. Presidente **Fdo. Secretario** **Fdo. Vocal**

Director/Tutor **Autor**

Calificación **Fecha**

Agradecimientos:

A mi familia, por darme todo su apoyo y ser tan pacientes conmigo.

A todos los amigos y compañeros que me han apoyado y animado para llevar a buen fin este Proyecto Fin de Carrera.

MUCHAS GRACIAS A TODOS

MEMORIA *

Pág.

1. Introducción

Como la mayoría ya conoce, Internet tuvo su origen en 1969 fruto de un proyecto de la Agencia de Proyectos de Investigación Avanzados (ARPA) perteneciente al ejército de los Estados Unidos, al abrir el primer nodo de la red ARPANET en la Universidad de California.

Tras sus inicios y posterior apertura de la red al gran público (no sólo militar), uno de los acontecimientos más importantes ocurrió a finales de 1989, cuando un científico del centro europeo CERN de Suiza diseñó un sistema para el intercambio global de información basado en hipertexto, denominado *"World Wide Web"*. La *WWW* (telaraña mundial) está formada por un conjunto de ordenadores conectados entre sí, denominados servidores Web, que albergan documentos escritos utilizando un formato especial (HTML), denominados páginas Web o documentos de hipertexto; a estos servidores se conectan los clientes, es decir aquellos usuarios que quieren acceder a la información almacenada en ellos.

Algo más tarde, en 1993, se diseña el primer navegador de páginas Web, con el que se afianza el uso de páginas de hipertexto como método para el intercambio de información, dadas sus características de sencillez y organización.

Con el gran entusiasmo generado por estas tecnologías se dio pie a la aparición de otras que añadieran valor a las ya creadas, como es el caso de Java; se trata de un lenguaje de programación orientado a objetos cuyos puntos fuertes se basan en ser Multiplataforma (independiente del S.O. de la máquina en que se ejecute) y el poder ser ejecutado dentro de una página Web. Cuando una aplicación Java se encuentra encapsulada en una de estas páginas se le denomina *"applet"*; la inclusión de este tipo de programas persigue crear páginas dinámicas, que permitan al usuario interactuar con la información que recibe.

La aparición de las infraestructuras (tanto hardware como software) asociadas a Internet, han propiciado el desarrollo de sistemas de telecontrol, con los que es posible gobernar cualquier aparato, o controlar sus parámetros, de manera remota. Su principal virtud reside en no requerir la presencia física del operario para llevar a cabo las acciones, facilitando la actuación en diferentes sistemas distantes desde un único punto. De esta manera es posible

acceder a instrumentos localizados en puntos de difícil acceso, o dar servicios sin requerir el desplazamiento de la mano de obra.

Es dentro de este entorno donde se desarrolla el presente PFC, basado en la Web y Java.

1.1. Peticionario

Actúa como peticionario del presente proyecto la Escuela Universitaria de Ingeniería Técnica de Telecomunicación (EUITT) de la Universidad de las Palmas de Gran Canaria (ULPGC), por medio del profesor D. Domingo Marrero Marrero, con el fin de cumplir la normativa vigente que impone como requisito indispensable la realización de un Proyecto Fin de Carrera, para la obtención del título de Ingeniero Técnico de Telecomunicación en Telemática.

1.2 Objetivos del Proyecto

El presente proyecto Fin de Carrera tiene como objetivo la creación de una herramienta genérica hardware/software, capaz de controlar un par de motores paso a paso de forma remota desde una página Web. La figura 1-1 muestra el diagrama global del sistema, desde los clientes hasta los motores, objetivos del proyecto.

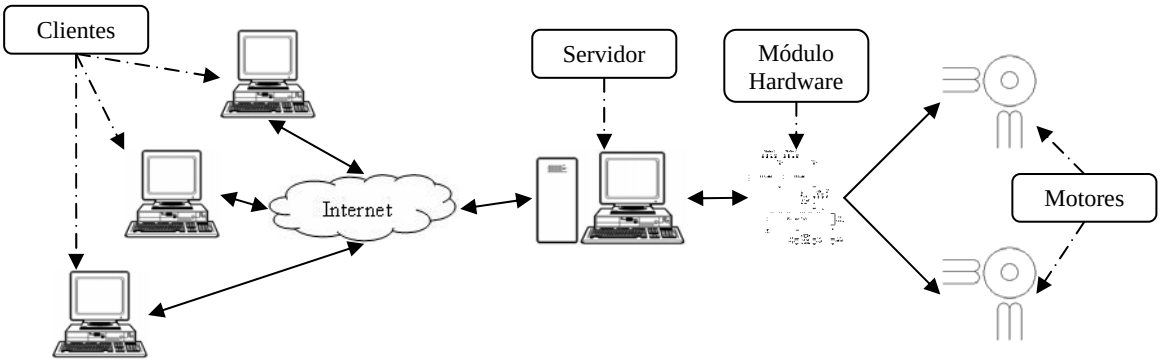


Figura 1-1 : Diagrama global del sistema

Para un uso correcto, la herramienta requiere de la gestión de usuarios, de tal forma que se permite el acceso ordenado (y permitido) a los mismos. De esta manera se evitan las posibles colisiones entre peticiones de movimiento y el acceso por parte de usuarios malintencionados.

El sistema debe seguir una estructura cliente/servidor en la que el servidor gestione los recursos hardware sobre los que el cliente hará las peticiones, de forma que sólo un cliente

puede actuar a la vez sobre estos recursos, dada la naturaleza de los mismos; en cualquier caso, todos los clientes registrados podrán conocer en cada momento el estado los motores.

La comunicación entre el servidor y el dispositivo encargado de controlar los motores se realizará utilizando el sistema USB, ya que con ello se incorpora un sistema de comunicación serie más moderno y eficaz que el ya tan utilizado RS-232. El uso de este sistema implica, por el momento, una dependencia con la plataforma utilizada, ya que en función del sistema operativo elegido, el acceso a los dispositivos varía ostensiblemente. En este caso, la plataforma escogida para el desarrollo y utilización de la herramienta es el sistema operativo Windows, lo que conlleva la necesidad de utilizarlo como soporte a las aplicaciones del sistema servidor.

El servidor debe ser una aplicación con interfaz gráfica, que permita a los usuarios permitidos (Administradores de la herramienta) configurar los parámetros oportunos de los motores y las conexiones con los clientes. Su función principal es la de hacer de puente entre el cliente y el dispositivo, al que le envía los datos de los movimientos a realizar y espera por la confirmación a los mismos. El servidor debe ser de tipo concurrente (de cara a la gestión de peticiones procedentes de la red) para poder atender a varios clientes de forma simultánea.

Por otro lado, el cliente deberá poder ejecutarse en cualquier máquina, permitiendo de esta manera el acceso desde cualquier sistema; es por ello que será implementado como un Applet Java, siendo accesible mediante un navegador Web. Una vez identificado y aceptado por el servidor, el cliente siempre tendrá acceso al estado actual de cada uno de los motores, siendo posible, si el servidor lo cree oportuno (según identificación y permisos), alterar la posición de cada uno de ellos.

El módulo hardware será controlado por microcontrolador encargado de traducir en secuencias “paso a paso”, a la velocidad correspondiente, las peticiones de movimiento que le llegan de la máquina que actúa como servidor y devolver una confirmación a las mismas. Deberá utilizarse el microcontrolador que mejor se adapte a nuestras necesidades, teniendo en cuenta su facilidad de uso, coste reducido, bajo consumo, facilidad de programación y ante todo disponibilidad de soporte USB. Este módulo deberá contar con el conector USB adecuado, así como aquellos que permitan la conexión de cada uno de los motores. Todo ello es implementará en una placa de circuito impreso.

Dado el hecho de que no todos los motores paso a paso requieren de la misma tensión de alimentación y que además pueden requerir potencias relativamente elevadas, la alimentación de los mismos se realizará empleando una fuente externa al módulo diseñado, de tal forma que se facilita el empleo de cualquier par de motores paso a paso. Por otro lado, la alimentación del microcontrolador se obtendrá del propio cable USB, el cual está preparado para ello.

1.3 Estructura de la memoria

- **Tema 1. Introducción :** En este tema se trata de acercar al lector las bases del PFC que en este documento se refleja.

- **Tema 2. Conceptos generales del software empleado en los sistemas finales :** En este capítulo se explicarán los conceptos fundamentales de los lenguajes de programación empleados para la construcción de la herramienta desarrollada en el presente PFC.

- **Tema 3. Conceptos generales de USB :** En donde se explicarán los fundamentos de este sistema empleado para comunicar un ordenador con sus periféricos, junto a aquellas características que por su importancia han hecho de USB un sistema ampliamente utilizado.

- **Tema 4. Conceptos generales del PIC16C745 :** En este capítulo se relatan las características más importantes del microcontrolador utilizado para gestionar el módulo hardware de este PFC.

- **Tema 5. Conceptos generales de Motores Paso a Paso :** En donde se explican los fundamentos de funcionamiento de estos motores, ya que conforman el objetivo final de este proyecto.

- **Tema 6. Descripción del proyecto :** En este capítulo se hace un estudio del *hardware* y *software* desarrollado en este PFC. En lo que al software de lo sistemas finales se refiere, el estudio se ha realizado separando en *análisis funcional* (estudio de los módulos en que se divide cada programa y sus usos) y *análisis estructural* (estudio de las clases y funciones diseñadas).

- **Tema 7. Manual de usuario :** Aquí se muestra la forma de utilizar la herramienta desarrollada, así como su funcionamiento, desde el punto de vista del usuario.

- **Tema 8. Conclusiones y líneas futuras :** En este capítulo se hace una reflexión sobre lo que se ha logrado hacer en función de los objetivos inicialmente propuestos

- **Bibliografía :** Donde se mostrarán las fuentes consultadas para disponer de los conocimientos necesarios con los que realizar este PFC.

- **Pliego de condiciones :** En este capítulo se expondrán tanto las condiciones técnicas (equipo) necesarias para poder utilizar la herramienta desarrollada, como la forma de instalar la misma, además del contenido del CD adjunto a este proyecto.

- **Presupuesto :** Donde se realizará un estudio pormenorizado de los costes derivados de los componentes, la mano de obra y la amortización de las herramientas empleadas para llevar a cabo este PFC.

- **Anexos :** Los anexos son empleados para añadir información complementaria sobre el PFC. Se encuentra dividido en:

    - **Anexo A. Herramientas empleadas.** Muestra las herramientas empleadas para realizar este proyecto, junto con una breve descripción de las mismas.

    - **Anexo B. Datasheets.** Incluye la información de los componentes utilizados en el módulo hardware, facilitada por sus fabricantes.

    - **Anexo C. Código de la aplicación :** Listado del código de los programas que conforman este PFC.

    - **Anexo D. Placa de circuito impreso:** Incluye la información de la placa de circuito impreso utilizada en el módulo hardware.

2. Conceptos generales del software empleado en los sistemas finales

Este capítulo se divide en tres grandes bloques. En primer lugar se aborda el lenguaje Java, utilizado en la aplicación principal de este proyecto. Inicialmente se tratan sus orígenes y características principales, para pasar a las herramientas necesarias en el desarrollo de aplicaciones. Finalmente se hace un estudio de sobre la inclusión de código nativo en Java, y las *Java Foundation Classes (JFC)*, que son un conjunto de librerías diseñadas para facilitar la elaboración de entornos gráficos en los programas.

En segundo lugar, se hace referencia a las características y facilidades aportadas por los lenguajes C y C++, teniendo en cuenta sus orígenes comunes y principales diferencias.

Para finalizar, en tercer lugar se explica el funcionamiento de básico de la API de Windows, utilizada para crear programas basados en este sistema operativo, en lenguajes C/C++.

2.1 Fundamentos del Lenguaje Java

Java [1][2] es un lenguaje de programación orientado a objetos, desarrollado por la compañía *Sun Microsystems*. Inspirado en C++, está diseñado para ser un lenguaje de programación independiente del equipo en el que se ejecute, lo suficientemente seguro como para moverse por las redes y lo bastante potente como para sustituir al código nativo ejecutable.

El creciente aumento de la importancia de las redes, y la necesidad de ofrecer contenidos dinámicos en las mismas, ha favorecido en gran medida la popularidad de Java como herramienta de desarrollo de este tipo de aplicaciones.

Java no sólo permite crear contenidos dinámicos para redes atractivos y funcionales, tanto en el lado del usuario (mediante los llamados *Applets*), como en el del servidor (Servlets), sino que también se ofrece como plataforma para el desarrollo de aplicaciones tradicionales, en la misma línea que otros lenguajes de programación como C/C++, Pascal, y en general todos los lenguajes de programación visuales.

2.1.1 Orígenes de Java

A principios de la década de los noventa, Sun Microsystems intentó abrirse camino en el mundo de la electrónica de consumo, y desarrollar programas para pequeños dispositivos electrónicos. Tras unos comienzos dudosos, Sun decidió crear una filial, denominada *FirstPerson Inc.*, para dar margen de maniobra al equipo responsable del proyecto.

Los objetivos iniciales de *FirstPerson* se centraron en aparatos microondas, tostadoras y, fundamentalmente, televisión interactiva. Este mercado, dada la falta de pericia de los usuarios en el manejo de estos dispositivos, requería unos interfaces mucho más cómodos e intuitivos que los sistemas de ventanas que proliferaban en aquel momento.

Otros requisitos importantes a tener en cuenta eran la fiabilidad del código y la facilidad de desarrollo. James Gosling, el miembro del equipo con más experiencia en lenguajes de programación, decidió que las ventajas aportadas por la eficiencia de C++ no compensaban el elevado coste que suponían las pruebas y la depuración de código ante cada cambio realizado. Gosling había estado trabajando en su tiempo libre en un lenguaje de programación que él había llamado *Oak*, el cual, aún partiendo de la sintaxis de C++, intentaba remediar las deficiencias que iba observando.

Entre los principales inconvenientes de los lenguajes tradicionales como C/C++ está el hecho de que son dependientes del sistema en que se utilizan; un cambio en el sistema, por tanto, supondría la recompilación de todo el software creado previamente.

Figura 2-1 : Duke, la mascota de Java

El primer proyecto en que se aplicó *Oak* recibió el nombre de *Proyecto Green* y consistía en un sistema de control completo de los aparatos electrónicos y el entorno de un hogar. El sistema presentaba una interfaz basada en la representación de la casa de forma animada, y el control se llevaba a cabo mediante una pantalla sensible al tacto, sobre un ordenador llamado *Star Seven*. Posteriormente se aplicó a otro proyecto denominado *VOD (Video On*

Demand) en el que se empleaba como interfaz para la televisión interactiva. Ninguno de estos proyectos se convirtió nunca en un sistema comercial.

Fue en 1993, con el boom por Internet, cuando la compañía decidió adoptar una nueva estrategia para su lenguaje, orientada a la realización de aplicaciones para el servicio de páginas Web, ya que *Oak* cumplía las premisas fundamentales de tamaño, robustez, orientación a objetos y de arquitectura independiente. Tras unos pequeños retoques y la incorporación de nuevas herramientas, *Oak* se convirtió en Java.

2.1.2 Características del lenguaje Java

Las características principales ofrecidas por Java respecto a otros lenguajes de programación se resumen en los siguientes apartados [2]:

- **Simple**

Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero al ser C y C++ lenguajes más difundidos, Java se diseñó para ser parecido a C++, y así dar pie a un rápido y fácil aprendizaje.

Java elimina muchas de las características de otros lenguajes como C++, para así mantener reducidas sus especificaciones, y añadir características muy útiles, como el recolector de basura *(garbage collector)*. No es necesario preocuparse de liberar memoria, el recolector se encarga de ello; se trata de un thread de baja prioridad que permite liberar bloques de memoria muy grandes, reduciendo la fragmentación de la memoria.

Java reduce en un 50% los errores más comunes producidos en lenguajes como C y C++, al eliminar algunas de sus características:

- Aritmética de punteros
- Referencias
- registros (struct)
- definición de tipos (typedef)
- macros (#define)
- necesidad de liberar memoria (*free*)

Además, el actual intérprete completo de Java muy pequeño, solamente ocupa 215 Kb de RAM.

- ***Orientado a objetos***

Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Las plantillas de objetos son llamadas *clases*, al igual que ocurre en C++, y sus copias, instancias.

- **Distribuido**

Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como *http (HyperText Protocol)* y *ftp (File Transfer Protocol)*. Esto permite a los programadores acceder a la información a través de la red, con tanta facilidad como a los ficheros locales.

Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se ejecuten en varias máquinas que interactúen entre sí.

- **Robusto**

Java realiza multitud de verificaciones en busca de problemas, tanto en tiempo de compilación como de ejecución (ver figura 2-2). La comprobación de tipos en Java, ayuda a detectar errores rápidamente en la etapa de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Se encarga de la gestión de la liberación y corrupción de memoria. También implementa auténticos *arrays*, en vez de listas enlazadas de punteros, con comprobación de límites, que evita la posibilidad de sobrescribir o corromper la memoria, al apuntar a zonas equivocadas.

Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los *byte-codes*, que son el resultado de la compilación de un programa Java; se trata de un código de máquina virtual, que es interpretado por Java. No se trata del código máquina directamente manejable por el hardware, pero ya ha pasado todas las fases del compilador: análisis de instrucciones, orden de operadores, ..., y ya tiene generada la pila de ejecución de órdenes.

Java proporciona:

- Comprobación de punteros
- Comprobación de límites de arrays
- Excepciones
- Verificación de byte-codes

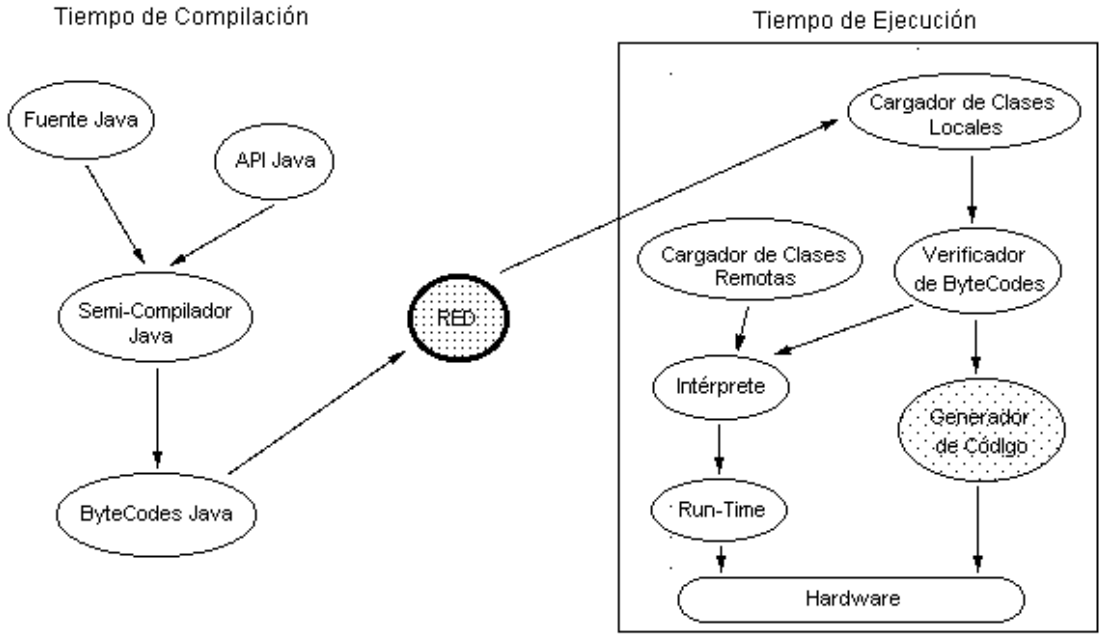


Figura 2-2 : Comprobaciones realizadas por Java

- **Neutral**

El compilador Java compila su código en un fichero objeto, cuyo formato es totalmente independiente del sistema en que será ejecutado (byte-codes). De esta manera, el código objeto podrá ser utilizado en cualquier máquina que disponga del sistema de ejecución (run-time), sin importar su arquitectura, ni la del sistema que lo creó.

Java pone una serie de APIs a disposición del programador, encargadas de entrar en contacto directo con el hardware, manteniendo una interfaz uniforme para todos los sistemas. Su implementación en cambio, será dependiente del sistema. Como ejemplos de este tipo de APIs podemos citar:

- Java 2D: gráficos 2D y manipulación de imágenes
- Java Media Framework : Elementos críticos en el tiempo: audio, video...
- Java Animation: Animación de objetos en 2D
- Java Telephony: Integración con telefonía

- Java Share: Interacción entre aplicaciones multiusuario
- Java 3D: Gráficos 3D y su manipulación

- **Multi-hilo:**

Java permite la ejecución de varias actividades simultáneas en un mismo programa. Los threads (o hilos), son básicamente pequeños procesos o piezas independientes de un gran proceso; al emplear recursos propios del lenguaje, son más fáciles de usar y más robustos que sus homólogos en C o C++, cuyo soporte viene dado por las APIs del sistema operativo correspondiente.

El beneficio de ser multi-hilo reside en la mejora sustancial en el rendimiento interactivo y en el comportamiento en tiempo real.

2.1.3 El entorno de desarrollo de Java

Existe multitud de entornos comerciales para el desarrollo de aplicaciones java. Entre ellos destaca el distribuido gratuitamente por Sun, creadora de Java: el Java 2 Software Development Kit (Java 2 SDK), que puede obtenerse gratuitamente en:

http://java.sun.com/j2se/index.jsp

Este entorno de programación utiliza la línea de comandos para realizar todas las operaciones (compilar, ejecutar...), lo cual puede resultar complicado y tedioso.

Por otro lado, existen *entornos de desarrollo integrado* (*IDEs*), que facilitan la programación, dotando al programador de un sistema en el que se puede editar, compilar y depurar el código, bajo una única interfaz, y en un entorno más amigable y visual, como puede ser un entorno de ventanas.

2.1.3.1 Java 2 SDK

La instalación completa del SDK está compuesta por las herramientas de desarrollo, y una abundante documentación.

La documentación contiene especificaciones de las APIs, páginas de referencia acerca de las herramientas proporcionadas, demostraciones y enlaces a información relacionada.

Las herramientas de desarrollo ofrecidas por el Java 2 SDK ofrecen multitud de aplicaciones, que soportan desde las básicas operaciones de compilación y depurado, hasta otras más especializadas que permiten crear aplicaciones que interactúan con la Web u otras redes. Entre las herramientas básicas encontramos:

javac: Compilador Java. Convierte el código fuente en byte-codes.

java: Intérprete que ejecuta las aplicaciones java.

javadoc: Genera documentos HTML a partir del código fuente Java.

appletviewer: Permite ejecutar y depurar applets, sin un navegador Web.

jar: Crea y manipula archivos comprimidos en el formato de ficheros Java Archive (JAR)

jdb: Depurador de Java (Java Debugger)

javah: Genera los ficheros de cabecera que establecen las reglas de implementación de código nativo, de tal forma que éste pueda ser invocado desde las clases de Java.

javap: Descompilador de Java. Convierte los byte-codes en código fuente Java.

2.1.3.2 El Java 2 Runtime Environment y el Java Plug-in

El Java 2 Runtime Environment, proporciona a cualquier sistema operativo (utilizando el paquete adecuado), un entorno en el que ejecutar aplicaciones Java. Está compuesto por la máquina virtual de Java (JVM), librerías de clases del API de Java, y archivos de apoyo; pero no incorpora ningún tipo de herramienta de desarrollo. Puede ser redistribuido junto con las aplicaciones desarrolladas en Java, de acuerdo con los términos de la licencia del Runtime Environment.

Pero si lo que se quiere ejecutar son applets en un browser, entonces requeriremos software adicional: el Java Plug-in. Este software permite que los applets sean ejecutados en un navegador Web, utilizando el Java 2 Runtime Environment de Sun, en vez del entorno de

ejecución que el navegador asigna por defecto. La ventaja de usar el Plug-in reside en que no todos los navegadores han implementado (o al menos correctamente) el Java 2; utilizando la versión de Sun nos aseguramos el soporte completo, y el acceso a las aplicaciones creadas con las versiones más recientes de Java (ya que en caso de utilizar un versión antigua, el propio Plug-in solicita permiso para actualizarse).

2.1.4 Una mínima aplicación Java

La aplicación más pequeña posible es aquella que imprime un mensaje en la pantalla. El mensaje típico suele ser "Hola Mundo", y es justamente el que presenta el siguiente fragmento de código:

```
1.  //Aplicación de ejemplo
2.  class HolaMundo {
3.      public static void main ( String args [] ) {
4.          System.out.println( "Hola Mundo" );
5.      }
6.  }
```

A continuación analizamos detalladamente la aplicación presentada:

- 1. //Aplicación de ejemplo

Esta primera línea se trata de un comentario. Los caracteres "//" hacen que todo lo que se escriba a su derecha sea interpretado como comentario, y por tanto, no es tenido en cuenta por el compilador.

- 2. class HolaMundo

En esta línea se declara la clase HolaMundo. Cada clase debe ser almacenada (si es pública) en un archivo con el nombre de la clase, y la extensión *.java*. Una vez compilada la clase, se creará un archivo (con los byte-codes) con el mismo nombre que la clase, y la extensión *.class*.

- 3. public static void main (String args [])

Esta línea especifica un método que el intérprete Java busca para ejecutar en primer lugar. Como sucede en otros lenguajes, Java también utiliza la palabra clave *main* para especificar la primera función a ejecutar.

-public: palabra reservada utilizada para indicar que la función miembro puede ser invocada por cualquiera, incluyendo el intérprete de Java.

-static: palabra reservada que indica que *main* se refiere a la propia clase en que reside, y no a ninguna instancia de la clase.

-void: indica que el método *main* no devuelve nada.

-String args[]: es la declaración de un array de *Strings*. Se utiliza para almacenar los parámetros escritos tras el nombre, en la línea de comandos.

- 4. System.out.println(“Hola Mundo”);

Esta línea es la encargada de mostrar el texto. Llama a la función println(), de la clase *out*, del paquete *System*. Esta función toma una cadena como parámetro, y lo envía a la salida estándar (normalmente, la pantalla).

- 5. }

6. }

Finalmente, se cierran las llaves que limitan la definición del método *main*, y de la clase *HolaMundo*.

2.1.4.1 Compilación y ejecución de HolaMundo

Para poder ver el resultado de esta mínima aplicación hay que seguir tres sencillos pasos:

- **Edición de los ficheros fuente**

En primer lugar se debe crear el código fuente; en nuestro caso, se trata de una única clase. Cada clase debe estar almacenada en un fichero independiente (sólo hay obligación si la clase es pública), es por ello que en nuestro caso deberemos almacenar el código en un fichero denominado HolaMundo.java.

- **Compilación**

Para este paso utilizaremos la herramienta incluida en el JDK, denominada *javac*. Esta aplicación se encarga de crear un nuevo archivo, a partir del fichero fuente *HolaMundo.java*, que contenga los byte-codes que maneja el intérprete de java. Este nuevo archivo conservará el nombre del fuente, pero utilizando la extensión .class.

Para utilizar *javac* basta con invocarlo, añadiendo como argumento el nombre de la clase a compilar. El hecho de que el directorio en que se encuentra *javac* pertenezca a la variable de entorno PATH del sistema, simplifica mucho el proceso.

```
%javac HolaMundo.java
```

- **Ejecución**

Para ejecutar la aplicación, debemos recurrir al intérprete de java. Para ello se debe invocar la aplicación *java* pasando como argumento el nombre de la clase (si previamente se compiló).

```
%javac HolaMundo
```

Tras esta línea debería aparecer en la pantalla:

```
Hola Mundo
```

2.1.5 Código Nativo

En ciertas ocasiones, aun a riesgo de perder el componente de portabilidad de las aplicaciones Java, se hace interesante hacer llamadas directas a funciones del sistema operativo, ya sea porque Java no dé soporte para la acción a realizar, o bien por simples motivos de eficiencia. Por esta razón, Java ofrece una interfaz para crear *código nativo*, denominada *JNI* (Java Native Interface).

Para escribir código nativo, e incluirlo en las clases de Java, se deben seguir una serie de pasos [3]:

- **Escribir la clase Java**

En primer lugar se debe escribir el código fuente de la clase que incluirá los métodos nativos. Estos métodos serán declarados de la forma:

native tipo_retorno nombre_metodo(argumentos);

La palabra reservada *native* se utiliza para indicar al compilador que la definición del método se encuentra en una librería. Para indicar el nombre de esta librería, se deben añadir al código de la clase las líneas:

```
static {
    System.loadLibrary("nombre_libreria");
}
```

La extensión de la librería será añadido por el compilador, pues ésta depende del sistema operativo; por ejemplo, en los sistemas win32, la extensión de las librerías de programación es *.dll*.

- **Compilar la clase**

Este paso es análogo al que se realizaría con cualquier otra clase de Java:

```
%javac nombre_clase
```

- **Generar el fichero de cabecera**

El siguiente paso consiste en generar un fichero de cabecera de C (.h). Esta cabecera definirá los tipos de datos a utilizar, y el nombre de las funciones. A partir de la cabecera se podrá implementar el código en C (el código nativo propiamente dicho).

Para crear la cabecera se utiliza la herramienta *javah*:

```
%javah –jni nombre_clase
```

El parámetro *–jni* se utiliza para indicar que el formato de la cabecera debe ser el de *Java Native Interface*, que es una actualización del formato utilizado en el SDK 1.0 (que además requería el empleo de un archivo adicional denominado *stubs*) al formato del SDK 2.

- **Implementar el código nativo**

Este paso consiste en definir las funciones declaradas en el archivo de cabecera, creado en el paso anterior, utilizando el lenguaje de programación *C/C++*. Este código es el que realizará las operaciones deseadas (el objetivo por el que nos hemos decidido a realizar métodos nativos).

- **Crear la librería compartida**

En este paso se crea la librería a la que llamará el código Java, para buscar la implementación del código nativo.

Crear la librería depende del compilador de C que utilicemos. Ejemplos de los casos más extendidos son las líneas de comandos:

- En Solaris, esta línea crearía la librería *libhello.so* [4]

```
cc -G -I/usr/local/java/include -I/usr/local/java/include/solaris \
      HelloWorldImp.c -o libhello.so
```

- En Microsoft Windows, esta línea generaría la librería dinámica *hello.dll* utilizando Microsoft Visual C++ 4.0 [4]:

```
cl -Ic:\java\include -Ic:\java\include\win32
      -LD HelloWorldImp.c -Fehello.dll
```

- **Ejecutar la aplicación**

Si la clase con método nativo incluye un método *main*, la ejecución del programa se realizará como siempre:

```
%java nombre_programa
```

En caso contrario, la clase con métodos nativos forma parte de un proyecto mayor, por lo que habrá que ejecutar la clase principal, que contiene el método *main*.

2.1.6 Los applets de Java

Los applets son aplicaciones hechas en Java que pueden “incrustarse” en páginas HTML, para ser ejecutadas en un navegador Web (con soporte Java).

Los applets permiten realizar páginas Web más atractivas, añadiendoles contenidos dinámicos e interactivos, frente a la simple combinación *textos–gráficos* de las páginas estándar.

2.1.6.1 El ciclo de vida de un applet

Los applets tienen como entorno de ejecución el navegador Web. Es por esta razón que su comportamiento varía sensiblemente con respecto al resto de aplicaciones. El applet debe ser iniciado por el browser, detenido cuando el navegador permanezca oculto (por no consumir recursos innecesarios), y eliminado cuando el navegador sea cerrado, ó el usuario cambie de página. Por todo ello, se establecen los siguientes estados:

- **Carga del applet**

Cuando es cargado por primera vez, se crea una instancia de la clase que controla el applet, se inicializa y comienza su ejecución.

- **Detención y reanudación**

Si el usuario oculta el browser en el que reside el applet, éste debería detener los procesos que consuman más recursos (no es una imposición, pero sí una recomendación), para así permitir al usuario realizar otras acciones. Una vez se retorna al applet, éste reanuda su ejecución.

- **Recarga**

Algunos navegadores permiten recargar el applet (descargar el applet y ejecutarlo otra vez). En este proceso, el applet libera sus recursos, detiene su ejecución, y ejecuta su finalizador para hacer una limpieza final de sus trazas. Acto seguido, se repite el proceso de carga del applet.

- **Descarga**

Una vez que concluye la ejecución en el navegador (por cierre de éste, ó cambio de página), se detiene la ejecución del applet, y se liberan todos los recursos utilizados por el mismo.

2.1.6.2 Métodos de control del applet

Existen cuatro métodos para controlar el applet. El programador deberá definir aquellos que desee utilizar, aunque no debe preocuparse de su llamada, ya que de esto se encarga el navegador:

- **init():** Este método es invocado en cuanto el browser carga el applet. Se ocupa de tareas de inicialización, previas al inicio de la ejecución del applet propiamente dicho.
- **start():** Invocado cuando el applet se hace visible, tras haber sido inicializado. También es llamado cuando el applet vuelve a estar visible, tras haber permanecido oculto.
- **stop():** Se llama automáticamente cuando el applet se oculta. Su finalidad es detener aquellos subprocesos que consumen recursos innecesariamente, ya que el usuario no está viendo el applet.
- **destroy():** Método llamado cuando se desea finalizar el applet, con el fin de liberar los recursos reservados durante la ejecución del mismo (excepto la memoria). Normalmente no se requiere su uso, pues la mayoría de los recursos se liberan en *stop()*; por tanto, *destroy()* se emplearía para liberar recursos adicionales.

2.1.6.3 Inclusión de un applet en una página HTML

Para que un applet pueda ser ejecutado (ya sea en un navegador Web ó en la utilidad *appletviewer*), es necesario incluirlo en el cuerpo de una página HTML. Esta operación se realiza añadiendo la etiqueta HTML <*APPLET*>, cuyo formato general es:

<APPLET CODE = miApplet.class [CODEBASE=] [NAME=]

WIDTH= HEIGHT=

[ALT=Texto alternativo]

[texto alternativo utilizado para indicar al usuario que su browser entiende la etiqueta <APPLET>, pero por alguna razón, no puede ejecutar el applet]

[ALIGN=] [VSPACE=] [HSPACE=] [ARCHIVE=archivo1, archivo2]

```
<PARAM NAME=          VALUE= >

…

</APPLET>
```

Los elementos entre corchetes indican que el campo es opcional. El significado de cada campo es el siguiente:

- CODE : Nombre de la clase principal a cargar.
- WIDTH : Ancho asignado al applet.
- HEIGHT : Alto asignado al applet.
- CODEBASE : URL base del applet.
- NAME : Permite asignar un nombre alternativo al applet. Se utiliza para permitir al applet comunicarse con otros elementos presentes en la misma página (como puede ser otro applet).
- ALIGN : Justificación del applet.
- VSPACE : Espacio vertical.
- HSPACE : Espacio Horizontal.
- ARCHIVE : Se utiliza para especificar los archivos comprimidos (ya sea en JAR o ZIP) en los que se deben buscar las clases utilizadas por el applet.
- PARAM : Utilizado para pasar parámetros al applet desde el fichero HTML. Cada parámetro está formado por el par:
  - NAME : Nombre del parámetro
  - VALUE : Valor del parámetro

  Ambos se dan en forma de String, aunque el valor sea numérico.

2.1.7 Las Java Foundation Classes (JFC)

Son un conjunto de componentes y características utilizadas para ayudar a construir los entornos gráficos de los programas ó GUIs (Graphic User Interface). Se definen como clases que contienen las siguientes características:

- **Componentes Swing**

  Incluyen la implementación de elementos gráficos como botones, paneles, menús y ventanas.

- **Look & Feel**

  Permite elegir el aspecto de cualquier aplicación que utilice componentes Swing. Actualmente existen tres tipos de entorno: Windows, Motif (sistemas UNIX) y Metal (aspecto propio de Java, asignado por defecto).

- **API de Accesibilidad**

  Permite el uso de tecnologías de asistencia para obtener información del interfaz de usuario, como pueden ser los lectores de pantalla o los displays de Braile.

- **Java 2D API**

  Permite incorporar gráficos 2D, texto e imágenes, en las aplicaciones y applets Java.

- **Soporte Drag & Drop**

  Dota a las aplicaciones de las capacidades “arrastrar y soltar” entre aplicaciones Java y aplicaciones nativas.

2.1.7.1 Características de Swing

Entre las características más importantes de Swing, cabe destacar:

- **Facilidades de la clase JComponent**

  Salvo los contenedores de alto nivel, todos los componentes Swing que empiezan por **J**, descienden de JComponent

- **Bordes :** Utilizando el método *setBorder*, seleccionar y asignar un borde para un componente gráfico, es realmente sencillo.

- **Doble Buffer** : Se trata de una técnica que mejora la apariencia de aquellos componentes que cambian su apariencia frecuentemente.

- **Tool Tips :** Se utilizan para dar ayuda instantánea al usuario, en forma de cadena de texto, acerca de un componente. Se activan cuando el cursor del ratón se detiene sobre un componente, mostrando la información pertinente a cerca del mismo.

- **Soporte de distribución :** Swing ofrece la posibilidad de encargarse de la distribución de los componentes dentro de sus contenedores, en base a unas restricciones dadas. Se trata de una característica muy útil, en favor de la portabilidad de la aplicación, ya que asignar ubicaciones absolutas para cada componente, no asegura la visibilidad de los mismos en los diferentes sistemas en que podrá ser ejecutada la aplicación.

▪ **Iconos**

Este tipo de objetos permite a los componentes Swing, mostrar imágenes.

▪ **Aspecto y comportamiento conectable (pluggable look & feel)**

Los componentes Swing pueden cambiar su aspecto y comportamiento en cualquier momento. Esta característica permite que la aplicación se parezca al entorno en el que se ejecuta (sistema operativo), facilitando de esta manera el acercamiento de los usuarios.

Modelos de datos y de estado separados

Esta característica permite trabajar, de forma independientemente, con el estado del componente y con la forma en que éste trata los datos. Puede ocurrir que un componente utilice varios modelos; de esta forma, podrá controlar por separado cada uno de los aspectos que utiliza el componente.

2.2 Conceptos generales de C/C++

2.2.1 C

El lenguaje *C* [6][7] nació en los Laboratorios Bell de AT&T, de la mano de Dennis Ritchie, en los años setenta. Se considera estrechamente relacionado con el Sistema Operativo *UNIX*, ya que su desarrollo se realizó en este sistema y debido a que tanto *UNIX* como el propio compilador de *C* y la casi totalidad de los programas y herramientas de *UNIX*, fueron escritos en *C*.

Este lenguaje se inspiró en el lenguaje *B*, con la intención de recodificar UNIX (que durante sus inicios estuvo escrito en ensamblador), en vistas a su gran portabilidad a otras máquinas. La principal ventaja de B sobre C es la capacidad de éste último para definir tipos y estructuras de datos.

Una de las peculiaridades de *C* es su riqueza de operadores. Prácticamente dispone de un operador para cada una de las posibles operaciones en código máquina.

A pesar de las buenas intenciones iniciales, y del potencial demostrado por el lenguaje, al igual que todos, presenta algunos inconvenientes:

- Carece de instrucciones de entrada/salida y de manejo de cadenas de caracteres. Este trabajo queda relegado a las librerías de rutinas, con la consiguiente pérdida de portabilidad que ello supone.
- La excesiva libertad en la escritura de los programas puede provocar errores de programación que, al ser correctos sintácticamente, no se detectan a simple vista.
- Las precedencias de los operadores convierten a veces las expresiones en pequeños rompecabezas.

A pesar de todo, C ha demostrado ser un lenguaje extremadamente eficaz y expresivo.

2.2.1.1 Elementos del lenguaje C

Las características del lenguaje C se deben a la unión de tres elementos:

- **Compilador**

Su misión consiste en traducir las sentencias escritas en el lenguaje C, al código de la máquina en el que se ejecutará la aplicación. El compilador es capaz de detectar ciertos errores durante el proceso de compilación, enviando el correspondiente aviso al usuario.

- **Preprocesador**

Se trata de un componente característico de C, que no existe en otros lenguajes de programación. Actúa sobre el código fuente, antes de que comience la compilación propiamente dicha, para realizar ciertas operaciones; por ejemplo, se encarga de la sustitución de las *constantes simbólicas* por su valor numérico.

- **Librería estándar**

Dado que uno de los objetivos de C es el de mantener un lenguaje lo más sencillo posible, las facilidades propias del lenguaje son reducidas. Es por esta razón que C no cuenta con sentencias para ciertas operaciones, como pueden ser las de manejo de E/S ó de cadena de caracteres, imprescindibles en casi cualquier tipo de programa. Estas carencias quedan cubiertas mediante el uso de librerías de funciones preprogramadas, que se distribuyen junto con el compilador; la agrupación de estas librerías de funciones básicas se conoce como *librería estándar*.

2.2.2 C++

El lenguaje C++ comenzó a desarrollarse en 1980 [7], de la mano de B. Stroustrup, dentro de la propia AT&T. En un principio era tratado como una extensión de C, denominada *C with classes*. En 1983 este lenguaje comenzó a ser utilizado fuera de AT&T, adquiriendo el nombre de C++ (haciendo clara referencia al operador de incremento de C "++"). En ese mismo año se formó el estándar ANSI C, el cual incorporó alguna de las mejoras aportadas por C++, como puede ser la forma de declarar las funciones, o los punteros a *void*. En 1989 fue convocado un comité ANSI, y poco después uno ISO, con el objetivo de estandarizarlo a nivel americano e internacional.

C++ conserva las ventajas de C en cuanto a la riqueza de operadores y expresiones, flexibilidad y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. Es por todos motivos que el C++ es actualmente un lenguaje versátil, potente y general, que cuenta con gran éxito entre los programadores profesionales.

2.2.2.1 Orientado a objetos

En cualquier caso, la característica que diferencia claramente a ambos lenguajes se centra en la programación orientada a objetos. Realmente, C++ es a la vez un lenguaje procedural (orientado a algoritmos, en la línea de C) y orientado a objetos.

Este punto es el que da verdadera relevancia a C++, ya que une la potencia de C con las ventajas de la orientación a objetos. Este paradigma permite la realización de grandes programas mediante la unión de elementos simples (*objetos*), que pueden ser diseñados y probados de manera independiente (y por supuesto, reutilizados en otras aplicaciones). Entre sus principales características encontramos:

- **Encapsulación**

  Es el mecanismo que agrupa el código y los datos que maneja. Los mantienen protegidos frente a cualquier interferencia o mal uso. Cuando el código y los datos están enlazados de esta manera, se ha creado un objeto. El código y los datos pueden ser privados para el propio objeto, o públicos para otras partes del programa.

- **Polimorfismo**

  Es la cualidad que permite que un nombre se utilice para dos o más propósitos relacionados, pero técnicamente diferentes. El propósito es poder usar un nombre para especificar una clase general de acciones, cuya implementación puede cambiar según sea el caso. De esta manera, se conserva el nombre, es decir, el identificador de la acción, mientras que la forma en que se realiza varía. Este concepto está estrechamente relacionado con la herencia.

- **Herencia**

  Proceso mediante el cual un objeto puede adquirir las propiedades de otro objeto. La información se hace manejable gracias a la clasificación jerárquica. De esta manera, es

posible crear clases complejas, a partir de clases simples; esta característica facilita la reutilización y la comprensión del funcionamiento de las clases.

- **Objeto**

Conjunto de variables y funciones encapsulados, pertenecientes a una clase. A este encapsulamiento es al que se denomina objeto. Por tanto, es la clase la que define las características y funcionamiento del objeto. Por así decirlo, los objetos son variables de una determinada clase.

2.2.3 Una mínima aplicación en C

Tal y como se hizo con el lenguaje Java, vamos a realizar el mismo proceso, y mostrar la aplicación más sencilla (aquella que muestra un mensaje en pantalla), pero esta vez en lenguaje C.

```
1.  //Programa de prueba en C
2.  #include <stdio.h>
3.  int main(int argc, char *argv[]) {
4.      printf("%s","Hola Mundo");
5.      return 0;
6.  }
```

A continuación se detalla el significado de cada línea:

- 1. //Programa de prueba en C

Esta primera línea es un comentario. Los caracteres "//" indican que todo lo que se escriba a su derecha no será tenido en cuenta por el compilador. Su uso se limita a informar al programador de algún aspecto del código.

- 2. #include <stdio.h>

Esta línea es una directiva del precompilador.

- #include: Esta palabra reservada indica al precompilador que se incluya al código, el fichero cuyo nombre se indica a continuación.

- <stdio.h>: Indica el nombre del fichero a incluir en el código de la aplicación. En este caso se trata de la librería stdio.h, que incluye las funciones necesarias para implementar la E/S estándar. Los caracteres "<>" indican que el compilador debe conocer la localización del archivo (normalmente se utiliza con la librería estándar).

- 3. int main(int argc, char *argv[]) {

En esta línea se declara la función principal, que será la primera a la que el sistema llame, una vez se ejecute la aplicación. Al igual que ocurre en muchos lenguajes, esta función se identifica por la palabra "main".

- int: indica el tipo de datos *entero*. En esta línea indica que la función main devuelve un entero al terminar su ejecución (es lo normal).

- int argc: este parámetro de la función *main* indica el número de parámetros introducidos en la línea de comandos, tras el nombre de la aplicación.

- char *argv[]: este parámetro es un vector de caracteres, que contiene los parámetros introducidos en la línea de comandos.

- 4. printf("%s","Hola Mundo");

Esta es la funcionalidad de la aplicación. Utiliza la función println, incluida en stdio, para enviar un dato con formato a la salida estándar (normalmente el monitor).

El primer parámetro de la función *"%s"*, es el indicador del formato. En este caso, indica que el dato a presentar es una cadena de caracteres; la indicada en el segundo parámetro.

- 5. return 0;

Esta instrucción indica la finalización de la función en la que se encuentra, devolviendo, además, un valor. En este caso, la función devolverá *0*, que normalmente indica que la aplicación finalizó correctamente.

- 6. }

Finalmente, se cierra la llave que limita la definición de la función *main*.

2.3 API de Windows

La API (Application Program Interface o Programación de la interfaz de una aplicación) de Windows [8] suministra un gran número de funciones y tipos de datos que permiten construir aplicaciones para Windows. Estos recursos se encuentran disponibles para crear aplicaciones en entornos C/C++.

Los recursos ofrecidos por la API aportan facilidades para la construcción de aplicaciones, desde crear la interfaz gráfica, hasta controlar cualquier aspecto del sistema en que se ejecute la aplicación, mediante llamadas directas a rutinas del sistema operativo (en este caso, Windows). Este aspecto se traduce en una mayor eficiencia a la hora de controlar ciertos aspectos de una aplicación, ya que al interactuar directamente con los sistemas de gestión del S.O., en vez de utilizar algún tipo de interfaz, ahorra intermediarios, recursos y tiempo.

En cualquier caso, hay ciertas ocasiones en las que la única forma de acceder a un recurso determinado es haciendo peticiones al Sistema Operativo; es en este aspecto en el que se hace esencial el uso de la API.

2.3.1 Recursos

Desde el punto de vista de Windows, un recursos es todo elemento susceptible de ser utilizado por las aplicaciones. Existen recursos físicos (memoria, ratón, impresora,...) y lógicos (gráficos, iconos, cadenas de caracteres,...).

El sistema operativo Windows se adueña de todos los recursos, por lo que si queremos acceder a alguno, tendremos que pedirle permiso. La gestión se realiza de esta manera dado que algunos recursos no pueden ser compartidos (por ejemplo el puerto serie), y se debe establecer algún tipo de control sobre los candidatos a poseer el recurso; de esta manera se intenta evitar que pueda producirse algún tipo de apropiación indebida. Aún en el caso de que el recurso sea compartible (como podría ser la memoria), se deben establecer mecanismos que gestionen su uso, con el objetivo de evitar colisiones entre aplicaciones, o malos usos.

Normalmente, el acceso a los recursos de realiza a través de un manejador o *Handle*. Un Handle es un tipo de datos específico de la API de Windows, que en caso de ser válido, da

acceso a un determinado recurso. Una vez terminada la operación sobre el recurso, debe cerrarse el manejador, con el objetivo de liberarlo, y permitir su uso a otra aplicación (en caso de que no pueda ser compartido).

2.3.1.1 Ventanas

Todas las aplicaciones interactivas ejecutadas en el sistema operativo Windows tienen, al menos, una ventana; se trata del interfaz a través del cual las aplicaciones se presentan al usuario, con el que intercambian información. La figura 2-3 muestra un ejemplo de ventana Windows.

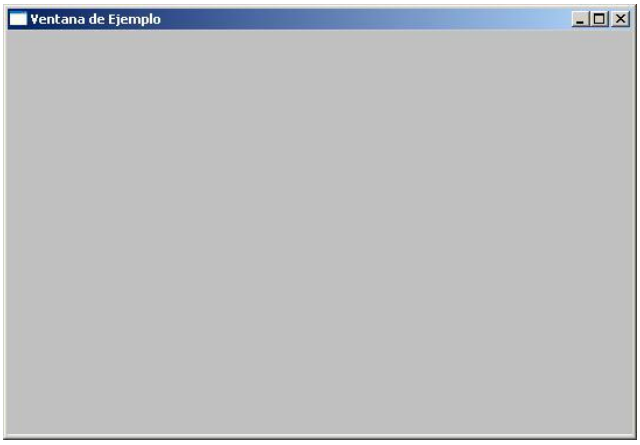


Figura 2-3 : Ejemplo de una ventana Windows

Cada ventana comparte el espacio de la pantalla con otras ventanas, incluso de otras aplicaciones, aunque sólo una puede estar activa; es decir, sólo una puede recibir información del usuario.

A continuación se muestra el código de ejemplo utilizado para crear la ventana de la figura 2-3, desarrollado en el lenguaje *C* utilizando la API de Windows:

```
#include <windows.h>

/*  Declaración del procedimiento de ventana  */
LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain (HINSTANCE hThisInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszArgument,
                    int nFunsterStil)
{
```

```
    HWND hwnd;              /* Manipulador de ventana */
    MSG mensaje;            /* Mensajes recibidos por la aplicación */
    WNDCLASSEX wincl;       /* Estructura de datos para la clase de ventana */

    /* Estructura de la ventana */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = "NUESTRA_CLASE";
    wincl.lpfnWndProc = WindowProcedure;        /* Esta función es invocada por
Windows */
    wincl.style = CS_DBLCLKS;               /* Captura los doble-clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Usar icono y puntero por defecto */
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;              /* Sin menú */
    wincl.cbClsExtra = 0;                   /* Sin información adicional para la */
    wincl.cbWndExtra = 0;                   /* clase o la ventana */
    /* Usar el color de fondo por defecto para la ventana */
    wincl.hbrBackground = GetSysColorBrush(COLOR_BACKGROUND);

    /* Registrar la clase de ventana, si falla, salir del programa */
    if(!RegisterClassEx(&wincl)) return 0;

    /* La clase está registrada, crear la ventana */
    hwnd = CreateWindowEx(
        0,              /* Posibilidades de variación */
        "NUESTRA_CLASE",    /* Nombre de la clase */
        "Ejemplo 001",      /* Texto del título */
        WS_OVERLAPPEDWINDOW, /* Tipo por defecto */
        CW_USEDEFAULT,      /* Windows decide la posición */
        CW_USEDEFAULT,      /* donde se coloca la ventana */
        544,            /* Ancho */
        375,            /* Alto en pixels */
        HWND_DESKTOP,       /* La ventana es hija del escritorio */
        NULL,           /* Sin menú */
        hThisInstance,      /* Manipulador de instancia */
        NULL            /* No hay datos de creación de ventana */
    );
```

```
   /* Mostrar la ventana */
   ShowWindow(hwnd, SW_SHOWDEFAULT);

   /* Bucle de mensajes, se ejecuta hasta que haya error o GetMessage devuelva
FALSE */
   while(TRUE == GetMessage(&mensaje, NULL, 0, 0))
   {
      /* Traducir mensajes de teclas virtuales a mensajes de caracteres */
      TranslateMessage(&mensaje);
      /* Enviar mensaje al procedimiento de ventana */
      DispatchMessage(&mensaje);
   }

   /* Salir con valor de retorno */
   return mensaje.wParam;
}

/*  Esta función es invocada por la función DispatchMessage()  */
LRESULT CALLBACK WindowProcedure (HWND hwnd, UINT mensaje,
WPARAM wParam, LPARAM lParam)
{
   switch (mensaje)              /* manipulador de mensaje */
   {
      case WM_DESTROY:
         PostQuitMessage (0);       /* Envía el mensaje WM_QUIT a la cola de
mensajes */
         break;
      default:                  /* Mensajes que no queremos manejar */
         return DefWindowProc (hwnd, mensaje, wParam, lParam);
   }

   return 0;
}
```

Dado que no es el objetivo de este apartado explicar con gran detalle este código, no se profundizará en su estudio. Cabe destacar la necesidad de dos funciones bien diferenciadas:

- **WinMain :** Función encargada de inicializar los recursos, prestando mayor atención a los componentes gráficos, y el registro de la ventana principal en el entorno Windows

- **WindowProcedure :** Función encargada de gestionar los mensajes que lleguen a la aplicación, manejando aquellos que interesen para la aplicación, o relegándolos a *DefWindowProc* para que sean tratados "por defecto".

2.3.1.2 Controles

Conforman el medio con el que la aplicación Windows interactúa con el usuario. La figura 2-4 muestra un ejemplo de controles Windows.

Figura 2-4 : Ejemplo de controles Windows (check box, list box y buttons)

Los más importantes y conocidos son:

- **control static:** etiquetas, marcos, iconos o dibujos.

- **control edit:** permiten que el usuario introduzca datos alfanuméricos en la aplicación.

- **control list box:** el usuario puede escoger entre varias opciones de una lista.

- **control combo box:** es una combinación entre un edit y un list box.

- **control scroll bar:** barras de desplazamiento.

- **control button:** realizan acciones o comandos; de button derivan otros dos controles muy comunes:

    - **control check box:** permite leer variables de dos estados "checked" o "unchecked"

- **control radio button:** se usa en grupos, dentro de cada grupo sólo uno puede ser activado.

2.3.2 Eventos

Los programas de Windows son orientados a eventos, es decir, normalmente se encuentran inactivos, hasta que sucede “algo” que les atañe y actúan como respuesta al evento.

Este es un aspecto muy importante a tener en cuenta, y se debe al hecho de que Windows es un sistema operativo multitarea, y debe repartir el tiempo del procesador entre todas las aplicaciones que lo solicitan. Si las aplicaciones fueran secuenciales puras, una aplicación no podría ejecutarse hasta que no haya finalizado la anterior.

3. Conceptos generales de USB

En este capítulo se hace un repaso general del sistema Universal Serial Bus (USB). En primer lugar se hace referencia a los principales problemas que presentaba la conexión de periféricos, y que provocaron la búsqueda de soluciones tales como el USB. A continuación se hace una breve referencia a los principales estándares generados para este sistema, para pasar a un estudio más extenso de cada uno de los elementos y capas que componen la arquitectura USB. Se hace especial hincapié en el apartado físico del sistema, entorno a conectores y señalización eléctrica. Finalmente se abordan los tipos de transferencias, las transacciones y los tipos paquetes involucrados en la comunicación.

3.1 Antecedentes

El estándar *USB ("Universal Serial Bus")* [9][11][12] fue introducido en el mercado en 1995, de la mano de empresas como *DEC*, *IBM*, *Intel*, *Microsoft* y *Compaq*. Su aparición y gran aceptación se deben a la intención de dar respuesta a las grandes dificultades asociadas a costes, configuración y conexión de los periféricos de los ordenadores personales, aportando una solución barata y simple (al menos desde el punto de vista del usuario).

Entre los grandes problemas existentes hasta la aparición de USB, cabe destacar:

- **Recursos limitados del sistema**

En el paradigma de *herencia de E/S* (periféricos estándar que derivan del IBM PC), los dispositivos son mapeados en el espacio de direcciones del procesador, y asignados a una línea de *IRQ* específica, o en algunos casos, a un canal *DMA*. Esto hace que los recursos del sistema sean escasos y limitados, a la vez que complican la configuración de los dispositivos.

- **Interrupciones**

  Debido al gran número de dispositivos diferentes que actualmente podemos encontrar en el mercado, uno de los principales problemas de la gestión de recursos se centra en la asignación de las *IRQ* (*Interruption Request*). Este problema se acentúa especialmente en los sistemas basados en el bus ISA, donde la mayoría de

las interrupciones están previamente asignadas a ciertos dispositivos, no permitiendo que, en su ausencia, éstas sean reutilizadas con otros fines.

- **Interfaces poco flexibles**

Los conectores de los dispositivos estándar (como puede ser el puerto serie ó el paralelo) sólo permiten la conexión de un único dispositivo al mismo tiempo. Esta limitación obliga frecuentemente al usuario a soportar el costo de una expansión del sistema, para obtener más conectores. Estos costos no son puramente económicos, pues también influyen factores de esfuerzo y tiempo dedicado, durante la conexión en el sistema y la posterior instalación del software controlador.

- **Conectar y reiniciar**

Como requisito para funcionar de una forma correcta, los periféricos estándar requieren que el sistema principal sea reiniciado, una vez han sido conectados. En este proceso, el software detecta el dispositivo, y se asegura de que funcione correctamente, además de comprobar que los recursos necesarios no están siendo utilizados por ningún otro dispositivo.

3.2 Aportaciones de USB

La aparición de un nuevo estándar para la conexión de periféricos, debe, al menos, superar las barreras impuestas por estándares anteriores, y USB no es una excepción. Las mejoras más destacadas, aportadas por este estándar, son:

- La gestión de los periféricos conectados al "bus" la lleva a cabo el propio subsistema USB, dejando al margen al sistema principal. Es decir, la conexión de dispositivos USB no implica el consumo de los recursos del sistema, tales como IRQs ó canales DMA, para cada dispositivo conectado. Los únicos recursos necesarios son aquellos que sustentan el controlador principal de USB (que dependerán del sistema en que esté alojado).

- USB soporta hasta 127 dispositivos diferentes en cada implementación, es decir, es capaz de controlar hasta 127 direcciones distintas. Cada dispositivo cuenta con una serie de registros que lo identifica, permitiendo que sea distinguido de entre todos los periféricos, y por tanto, reconocido por los controladores que lo gobiernan. Estos

registros se conocen como *"endpoints"*. Este método de direccionamiento es completamente independiente del utilizado por dispositivos no-USB, por lo que no es posible ningún tipo de interacción.

- Soporta tres velocidades de transmisión:

  - 1.5 Mb/s
  - 12 Mb/s
  - 480 Mb/s

En las versiones 1.x del estándar, sólo había soporte para las velocidades de 1.5 Mb/s y 12 Mb/s. Posteriormente, con la aparición de la versión 2.0, se añadió un nuevo nivel, 480 Mb/s, cuya única finalidad no es la de soportar dispositivos de muy alta velocidad, sino también ser capaz de albergar un mayor número de dispositivos de baja ó alta velocidad en un único bus (que de otra manera, debían ser incorporados en diferentes buses del mismo subsistema USB).

- USB es capaz de detectar la conexión/desconexión de un dispositivo, y actuar en consecuencia; es decir, ante la conexión de un dispositivo USB, sin necesidad de reiniciar el ordenador, es capaz de configurar el nuevo dispositivo, e instalar el software controlador correspondiente. Lo mismo ocurre cuando el dispositivo es desconectado, liberando los recursos utilizados de forma automática.

- Existen unos dispositivos denominados *"Hub"*, que permiten añadir puertos USB adicionales al sistema, sin necesidad de acceder al interior del mismo, o de instalar ningún tipo de software. Los *Hub* pueden encontrarse tanto como dispositivos independientes, como integrados en otros dispositivos, siendo el caso de teclados, impresoras ó monitores.

- Los periféricos pueden alimentarse directamente del bus, ya que este suministra 5v de tensión y una intensidad entre 100 y 500 mA, dependiendo del puerto *Hub*.

- USB ahorra energía, ya que los dispositivos conectados al mismo, pasan a un estado de inactividad tras 3ms sin actividad en el bus. En este estado, no pueden consumir más de 500μA del bus.

- Las transacciones USB incluyen un mecanismo de detección de errores, que vela por que la entrega de los datos se realice de forma correcta. Si se detecta información errónea, la transacción puede ser reiniciada.

- USB define cuatro tipos de transferencias de datos, con las que se pretende dar soporte a las diferentes exigencias que cada dispositivo pueda tener:
  - **Isócrona**
  - **Interrupción**
  - **Control**
  - **Bulk**

En el apartado 3.11.1 se exponen los detalles de cada tipo de transacción.

3.3 USB 1.x

Los dispositivos que siguen esta norma pueden alcanzar velocidades de hasta 1.5Mb/s (Low-Speed) ó 12Mb/s (Full-Speed).

3.3.1 Generación de transacciones

Las transacciones en USB 1.x se basan en la ejecución de una lista encadenada de estructuras de datos, llamadas "*descriptores de transferencia*". Cada descriptor especifica una operación solicitada por el software, para acceder al dispositivo. Esta operación se añade a la lista encadenada, para ser ejecutada en cuanto se tenga oportunidad, manteniendo el orden temporal en que deben ser lanzadas las acciones. Principalmente, la información contenido en un descriptor es:

- La dirección del dispositivo
- El tipo de transacción a realizar (lectura ó escritura)
- El tamaño de la transferencia
- Velocidad de la transacción
- Un puntero al buffer de datos (de donde se leerá ó donde se escribirán los datos)

La captura y ejecución de la lista para cada dispositivo, se produce durante un intervalo de 1ms, llamado "*frame*". En este periodo, son enviados tantos descriptores como sea posible, a la tasa de transferencia acordada, hasta que el sistema ceda otro *frame*, en el que se repetirá el proceso.

Algunos dispositivos requieren enviar información en cada *frame*; otros en cambio, sólo requieren de transferencias tras cierto tiempo. En cualquier caso, sólo se accede a los

dispositivos cuando el software del cliente hace peticiones de transferencia (ya sea de lectura ó escritura) al dispositivo, lo cual permite organizar, optimizar y compartir el bus con cierta eficiencia, ya que es el sistema principal el que gestiona el tráfico.

3.4 USB 2.0

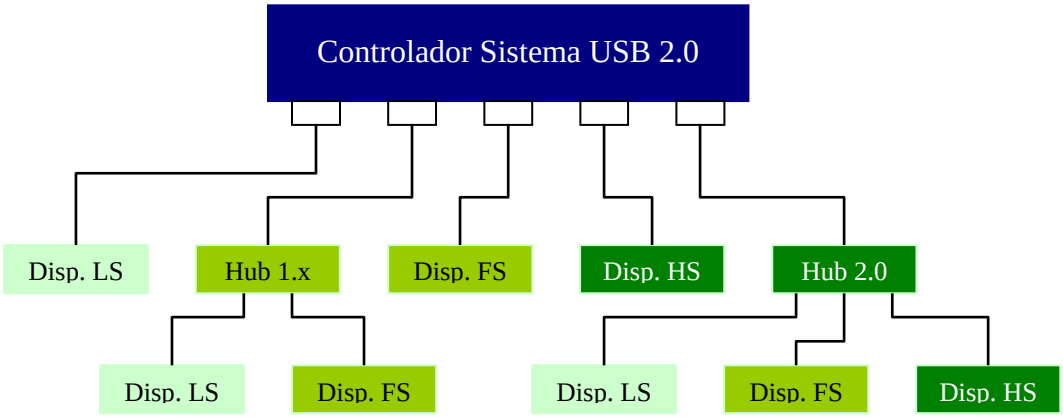


Figura 3-1 : Conexión de dispositivos USB 1.x a un sistema USB 2.0

El nuevo estándar, manteniendo soporte completo con las versiones anteriores (figura 3-1), aumenta la eficiencia del bus, y añade un nuevo nivel de transferencia de datos: 480Mb/s *(High-Speed)*. Esta nueva incorporación tiene como objetivo fundamental, aumentar el número de dispositivos que pueden acomodarse en el bus, ya que, con las versiones anteriores, todos los dispositivos conectados debían compartir un ancho de banda relativamente pequeño (12Mb/s), en comparación con el número de dispositivos que el sistema debía permitir. El nuevo nivel concede un bus 40 veces más rápido que el anterior.

3.4.1 Split Transactions

Debido a cambios en la eficiencia del bus, el hecho de emplear dispositivos LS *(Low-Speed)*, ó FS (Full-Speed), no supone un impacto importante en los dispositivos HS *(High-Speed)*. Para ello, en el caso de que un dispositivo LS/FS sea conectado a un puerto HS, se utiliza un nuevo tipo de transacción: “Split Transaction”. Esta transacción encapsula dentro de un paquete HS, la información FS/LS, para con ello mantener una uniformidad en el protocolo; el *Hub* al que están conectados dichos dispositivos, será el encargado de “traducir” a la velocidad correspondiente.

3.4.2 Dispositivos High-Speed

Tal y como ocurría en USB 1.x, las transacciones de velocidad superior no quedan reflejadas en los dispositivos de menor transferencia, es por ello que las transacciones generadas por el *host* HS, sólo son enviadas a los dispositivos HS, quedando bloqueadas para los FS y LS.

El método de acceso a los dispositivos es similar al utilizado en el estándar anterior con la salvedad de que en los dispositivos HS, los descriptores son enviados en intervalos de 125μs, llamados *"microframes"*.

3.5 Elementos primarios del sistema USB

3.5.1 Controladores del cliente USB

Elemento que, en el sistema principal, controla algún dispositivo USB. Todas las transacciones del sistema son generadas por el software USB. Normalmente, son efectuadas por el controlador de un dispositivo, que quiere acceder al mismo. Este controlador hace de intermediario entre el dispositivo y el controlador de USB, que gestiona el sistema principal.

Para la transferencia de datos, ya sea en una operación de lectura ó de escritura, el controlador del dispositivo habilita un buffer al controlador principal, en el que se almacena la información a enviar al dispositivo, ó la leída desde el mismo.

Para inicializar el diálogo con el dispositivo USB al que representan, realizan peticiones al controlador del bus USB, a través de los *paquetes de petición de E/S ("I/O Request Packets", IRPs).*

3.5.2 Controlador del bus USB

Es el elemento software que conoce las características de cada uno de los elementos conectados al sistema USB, por lo que se encarga de crear las transacciones individuales a partir de las peticiones realizadas por el controlador del dispositivo (IRPs), de acuerdo al modo de funcionamiento del dispositivo.

Las características de los dispositivos se conocen durante la configuración del mismo, mediante el envío de los *descriptores del dispositivo*.

3.5.3 Controlador del Host USB

Este elemento software se ocupa de programar el orden en el que los elementos de información, previamente asignados a las listas de transacciones de cada dispositivo, serán enviados al bus.

La forma en que se organizan estas transacciones dependerá, entre otros factores, de la velocidad de los dispositivos conectados, el tipo de transacción, y la prioridad de los mismos.

3.5.4 Hub Principal (Root Hub)

Es el elemento que da, en el sistema principal, el soporte físico a la comunicación del sistema USB, aportando los puertos (conectores) para sus dispositivos.

Entre sus funciones encontramos las de controlar la energía suministrada a sus puertos (de donde se alimentarán los dispositivos), habilitar o deshabilitar los puertos, avisar cuándo un dispositivo ha sido conectado, llevar un control sobre el estado de las conexiones, y aportar el camino físico por el que se encauzan los datos.

3.5.5 Controlador Principal (Host Controller)

Elemento responsable de generar la lista encadenada de *descriptores de transferencia* y de albergarla en memoria, con el objetivo de que, en cada *frame* (ó *microframe*), sean enviadas al bus.

También se ocupa de las transacciones; para ello, ante una petición del software de gestión, crea un transacción, se ocupa de su serialización, y de su paso al bus, a través del Hub Principal. Cuando el dispositivo devuelve datos, los conforma (paralelo) y los entrega al software controlador del dispositivo, para su procesamiento.

3.6 Descriptores de dispositivo

USB implementa *clases de controladores de dispositivo*, que agrupan a los dispositivos con características comunes, que pertenecen a la misma clase. De esta manera, cada grupo tiene un controlador común *de clase*, que es capaz de manejar a todos sus dispositivos.

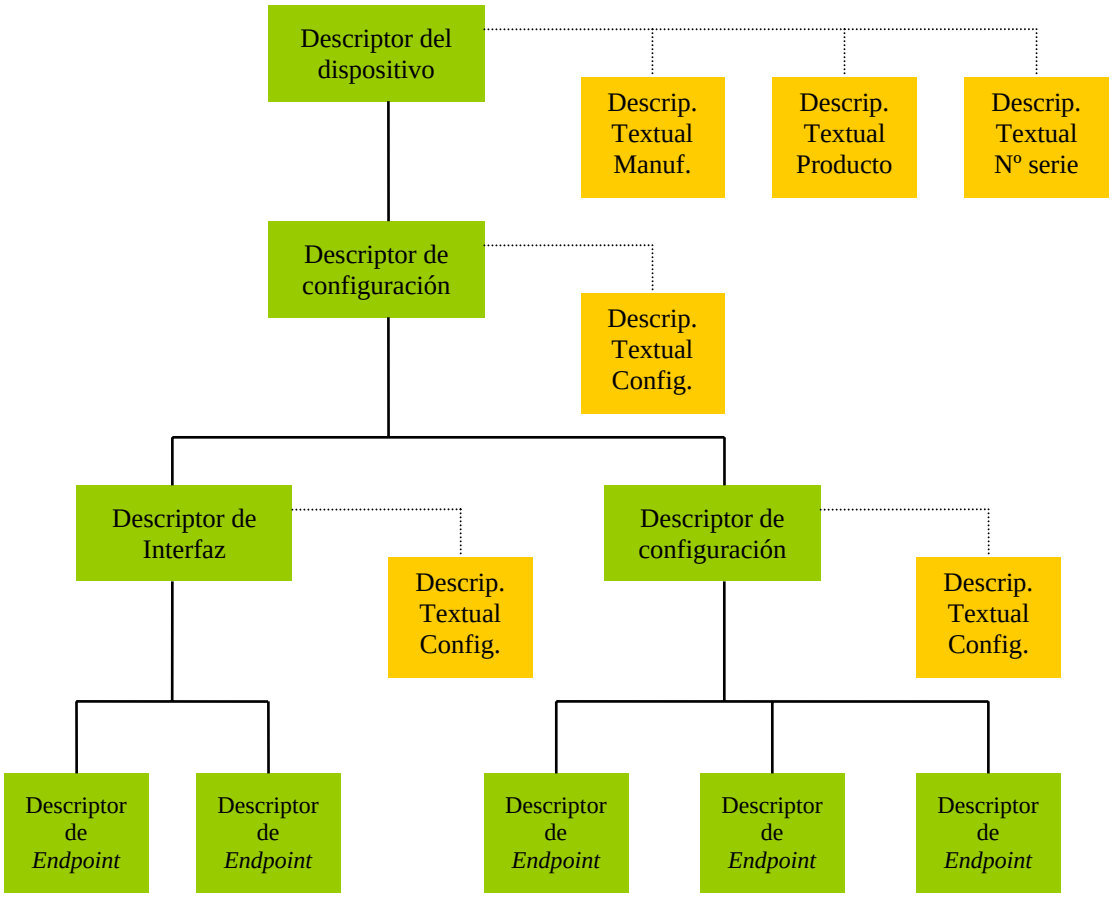


Figura 3-2 : Descriptores Estándar

A través de estos descriptores estándar, el dispositivo es capaz de identificarse a sí mismo. Incluyen:

- **Descriptor del dispositivo:**

  Contiene información general del dispositivo, tal como el pipe utilizado por defecto para configurar el dispositivo, ó el número de posibles configuraciones que admite el dispositivo.

- **Descriptor de configuración:**

  Existe uno por cada configuración soportada por el dispositivo. Contiene información general de cada configuración, y del número de interfaces que utiliza en dicha configuración.

- **Descriptor de Interfaz:**

Utilizado para definir las interfaces utilizadas por el dispositivo. Un dispositivo tiene varios interfaces cuando cuenta con más de una funcionalidad, y cada una de ellas requiere de un controlador diferente.

Su función principal es la de indicar la *clase* del dispositivo de cada interfaz, y el número de descriptores para registros "*endpoint"* que utiliza.

- **Descriptores de registros e*ndpoint* :**

Cada uno de estos registros define un punto de comunicación. Este descriptor indica el tipo de transferencia soportado por el *endpoint*, y la máxima transferencia soportada.

- **Descriptor textual :**

Contiene información en forma de cadenas de texto *(unicode)*, utilizadas para dar información al usuario del sistema al que se conecta el dispositivo.

- **Descriptor de clase específica :**

Contienen información adicional, que no es facilitada por los descriptores estándar, si el dispositivo así lo requiere.

La figura 3-2, hace referencia a la organización de los descriptores estándar.

3.7 Sistema de capas

El marco de trabajo de USB, dispone de tres capas lógicas, cuya finalidad es ayudar a entender el modelo de comunicación del sistema; para ello, describe las interacciones entre los elementos software y hardware presentes en el sistema principal. La figura 3-3 representa las relaciones entre las capas USB.

3.7.1 Capa de Interfaz del bus

Es la capa de más bajo nivel de la arquitectura. Su propósito se centra en la conexión física, la señalización eléctrica y la transferencia de paquetes.

Esta capa representa el flujo de bits a través del cable, que circulan entre el sistema central y el dispositivo USB,

3.7.2 Capa del dispositivo USB

Esta capa la componen el software, en el sistema principal, y la visión lógica del dispositivo. La visión lógica del dispositivo es el conjunto de elementos que conforman la interfaz a través de la que interactúa con el exterior, es decir, el conjunto de sus registros *endpoint*.

Entre sus funciones principales, encontramos:

- Detección de la conexión/desconexión de los dispositivos
- Configuración del dispositivo
- Asignación de ancho de banda (en base a la visión lógica del dispositivo)
- Gestión del flujo de control y de datos
- Creación de informes de estado
- Programación de las transacciones
- Controlar el interfaz eléctrico (capa inferior)

3.7.3 Capa de función

Esta capa representa la relación entre el software cliente, y el interfaz funcional del dispositivo. Cada una de estas interfaces representa un caso particular dentro de una clase de dispositivo, cuyo controlador es el encargado de manejar al periférico.

El software cliente no puede acceder directamente a los dispositivos USB, pues no hay ningún tipo de acceso directo (ya no se encuentran mapeados en memoria, ni en el espacio de E/S); es por ello que debe hacerlo a través del controlador USB que reside en la capa inferior, indicando el interfaz al que se pretende acceder, así como el modo y los parámetros a utilizar.

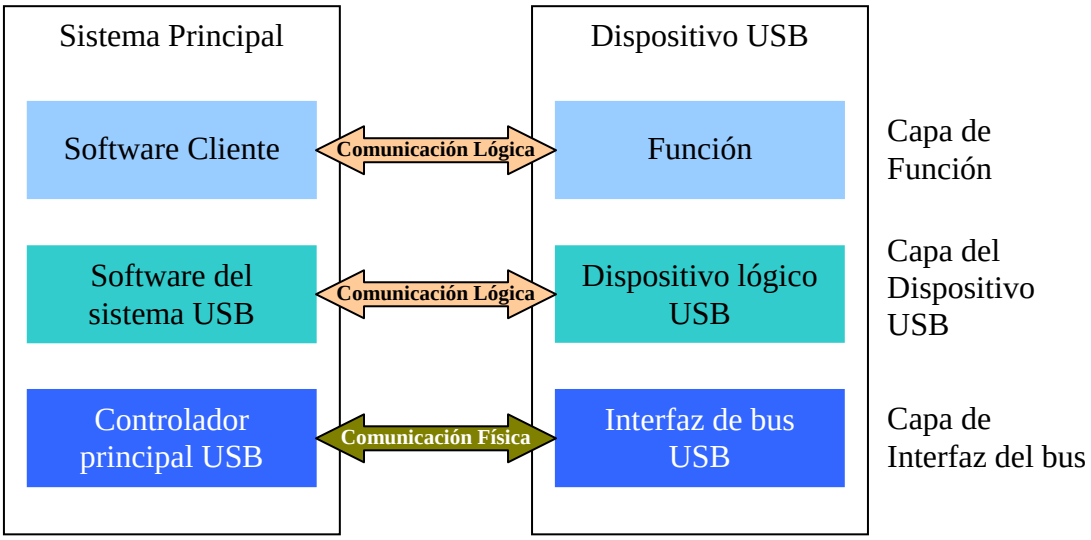


Figura 3-3 : Sistema de capas USB

3.8 Conectores

Con la intención de evitar confusiones a la hora de conectar los dispositivos, como puede ser el caso de interconectar dos puertos del mismo sistema USB, han sido diseñados dos tipos de conectores:

- Tipo A : Provee de un puerto de conexión a los *Hubs* (ya sea del sistema principal, ó como periférico). El conector hembra reside en el *Hub*, mientras que el macho lo encontramos en el cable. La figura 3-4 muestra un ejemplo del conector macho.

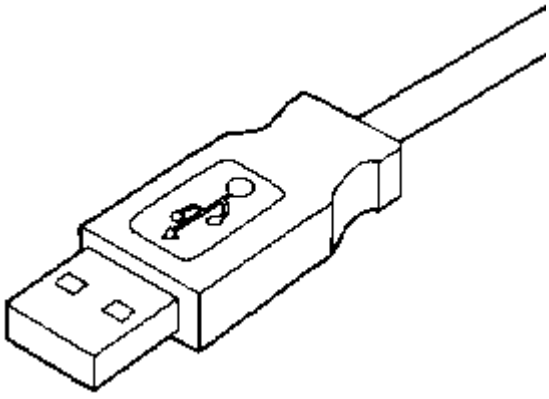


Figura 3-4 : Conector USB Tipo A macho

- Tipo B : Provee de un puerto de conexión al periférico, en los casos en que el cable no es parte del propio dispositivo. Siguiendo la misma idea empleada en el *tipo A*,

el conector hembra se encuentra en el dispositivo, y el macho en el cable. La figura 3-5 muestra un ejemplo del conector macho.

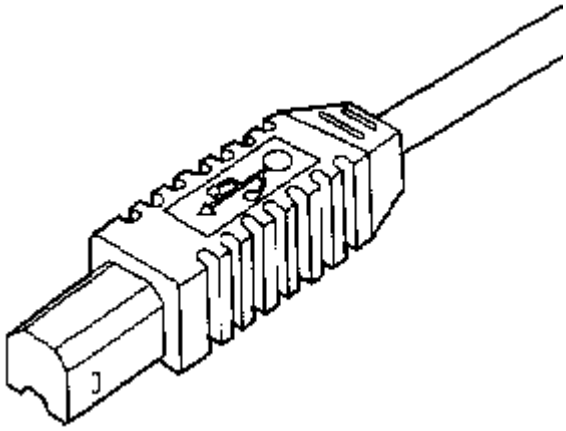


Figura 3-5 : Conector USB Tipo B macho

Cada conector cuenta con cuatro contactos. El par más corto es el utilizado para transmitir los datos en modo *diferencial*; el par restante (cuyos contactos son más largos) es utilizado para alimentar al dispositivo.

El uso del conector *B* es obligatorio en los dispositivos *FS* y *HS* que no tengan un cable incorporado (lo cual simplifica el cambio de un cable defectuoso) [12]. En los dispositivos *LS* es posible utilizar cualquiera de los dos conectores, aunque eligiendo el *tipo B*, es posible emplear un cable más largo que el específico para estos dispositivos.

Los contactos del conector están numerados, y a cada conductor del cable se le ha asignado un color para facilitar su identificación, tal como muestra la tabla 3-1.

Nº de contacto	Nombre	Color
1	Vcc	Rojo
2	-Datos	Blanco
3	+Datos	Verde
4	Gnd	Negro

Tabla 3-1 : Contactos de conectores USB [9]

3.9 Cables

La especificación USB define dos tipos de cable, uno para los dispositivos *LS* (más económico) y otro para los *FS* y *HS*.

3.9.1 Cables Low-Speed

Estos cables fueron diseñados para soportar velocidades de hasta 1.5 Mb/s, y son empleados en entornos donde el ancho de banda no sea un parámetro crítico.

Los datos, enviados de forma diferencial, son transmitidos por un par no trenzado de 28 *AWG*. El cable debe contar con un apantallamiento interno, conectado a un conductor de drenaje, que une el conector USB con la carcasa del dispositivo. La figura 3-6 muestra la sección de un cable de estas características.

Los cables *LS* pueden alcanzar una longitud máxima de tres metros, y su retardo de propagación no debe ser superior a 18ns (en cada dirección).

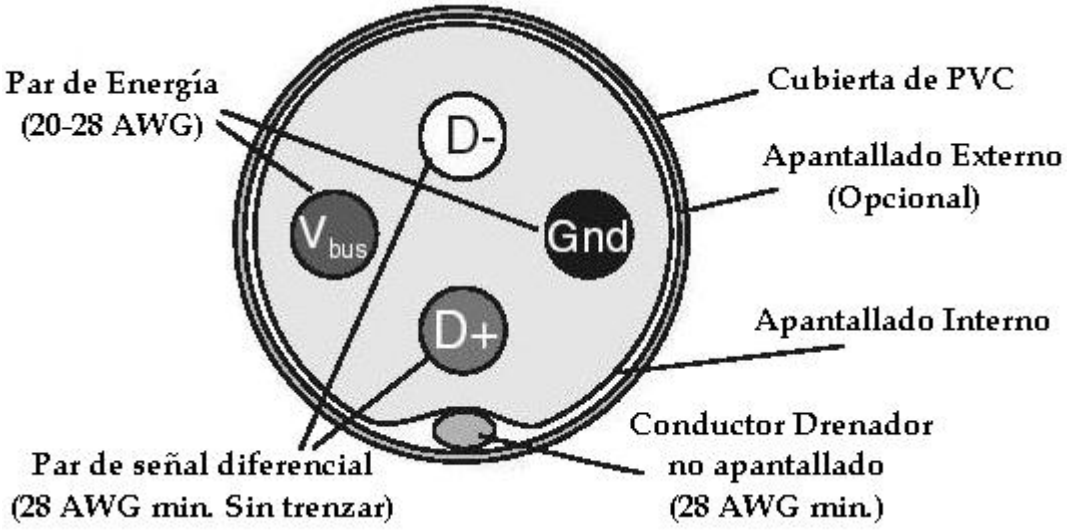


Figura 3-6 : Sección de un cable USB Low-Speed

3.9.2 Cables Full-Speed y High-Speed

En estos cables, el par de conductores que transmiten los datos, debe estar trenzado junto con el apantallamiento interno y el externo (que debe estar presente). La figura 3-7 muestra la sección de un cable de estas características.

El retardo de propagación máximo es de 26ns, cuando trabaja en el rango de 1 – 480 MHz; en caso de superar este valor, deberá emplearse una longitud menor del cable. Por lo tanto,

para calcular la longitud del cable, son parámetros decisivos el retardo de propagación, y la atenuación de los conductores. De cualquier modo, la especificación define que cinco metros es la longitud máxima aceptable.

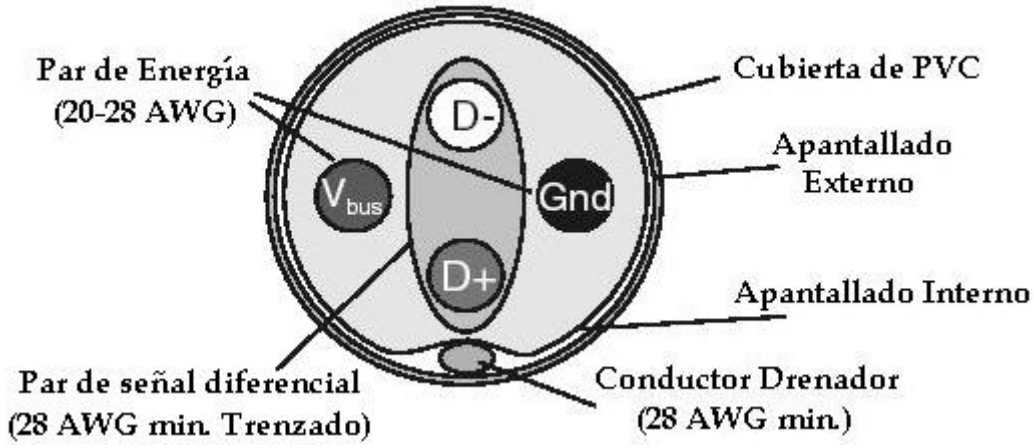


Figura 3-7 : Sección de un cable USB Full-Speed / High-Speed

3.9.3 Distribución de potencia

Los dispositivos USB y los Hubs USB pueden alimentarse de la potencia disponible en el cable, ó bien, utilizar un sistema de alimentación propio e independiente de USB.

3.9.3.1 Hubs

Una de las funciones más importantes relegadas a los *Hubs* es la del control de energía. Un *Hub* debe proveer energía a los dispositivos conectados a sus puertos; para ello, puede ceder potencia procedente del cable, ó bien, si cuenta con una fuente de alimentación, suministrar su propia energía. También es posible que el *Hub* obtenga la energía de ambas fuentes (previo aviso al sistema principal, a través del descriptor oportuno): una parte proviene del cable, y la restante de su propia alimentación. La figura 3-8 muestra un esquema genérico de este tipo de Hubs.

En cualquier caso, la corriente máxima suministrada no debe exceder los 500mA para cada dispositivo (incluso si lo que hay conectado es otro *Hub*, ya que éste último es tratado como un dispositivo más). Si el *Hub* tiene alimentación propia, podrá ceder el máximo de potencia a cada uno de sus dispositivos. En caso contrario, el único requisito que se le exige es que suministre un mínimo de 100mA, ya que es la corriente máxima que un puerto debe ceder a un dispositivo durante su configuración; si no le es posible, el último dispositivo que haya intentado conectarse será rechazado, y una alarma será enviada al

sistema principal. Puede ocurrir que el *Hub* sea capaz de ceder los 100mA mínimos exigidos, pero no la corriente necesaria para el funcionamiento normal del dispositivo; si este fuera el caso, durante el proceso de configuración, el dispositivo sería rechazado (en la negociación de requisitos).

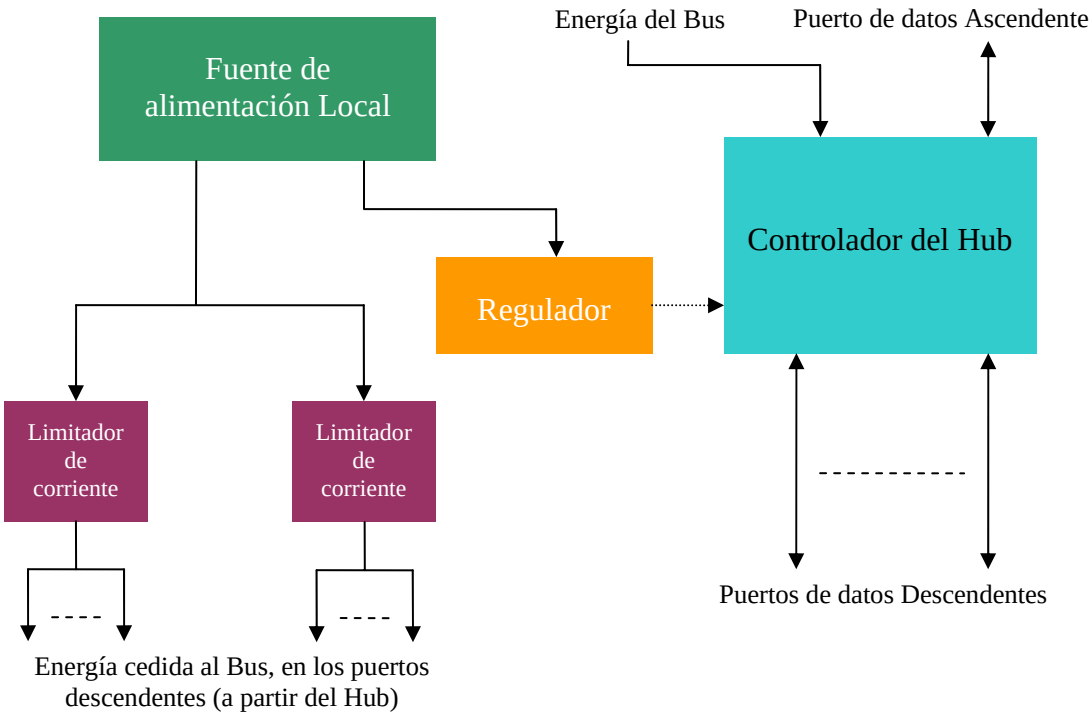


Figura 3-8 : *Hub* híbrido (alimentación propia y del bus)

Es por este hecho que si un *Hub* sólo va a ser alimentado por el cable USB, el número máximo de puertos de que puede disponer debe ser 4, ya que la corriente máxima que lo puede alimentar es de 500mA (pues será tratado como cualquier dispositivo), a los que habrá que restar los que necesita el propio *Hub* para realizar sus funciones.

Puede ocurrir que un *Hub* con una fuente de alimentación (si así lo ha indicado por medio de sus descriptores al sistema principal), pierda este recurso; en este caso, el *Hub* no puede utilizar potencia adicional del cable USB, por lo que si no le es posible continuar su función sin alimentación externa, éste dejará de actuar tras avisar al software USB. El sistema gestor de USB puede conocer si se encuentra alimentado o no, atendiendo al estado del dispositivo.

Por motivos evidentes de seguridad, existe una limitación que impide que los *Hub* suministren una corriente superior a 5A, al conjunto de sus puertos. Teniendo en cuenta

que cada puerto puede consumir como máximo 500mA, podemos afirmar que se trata de una limitación bastante holgada.

3.10 Interfaz de señalización en entornos LS – FS

Esta interfaz es utilizada para ciertas funciones primarias, tales como la detección de la conexión/desconexión de dispositivos (y su consiguiente identificación de tipo) ó el reinicio de los mismos.

3.10.1 Detección de conexión y velocidad de un dispositivo

Antes de proceder al intercambio de datos, el dispositivo debe ser detectado por parte del sistema principal. Para ello, USB está permanentemente a la escucha de lo que ocurre en sus puertos. Tras la detección, comenzará el intercambio de datos con objeto de identificar y configurar el dispositivo.

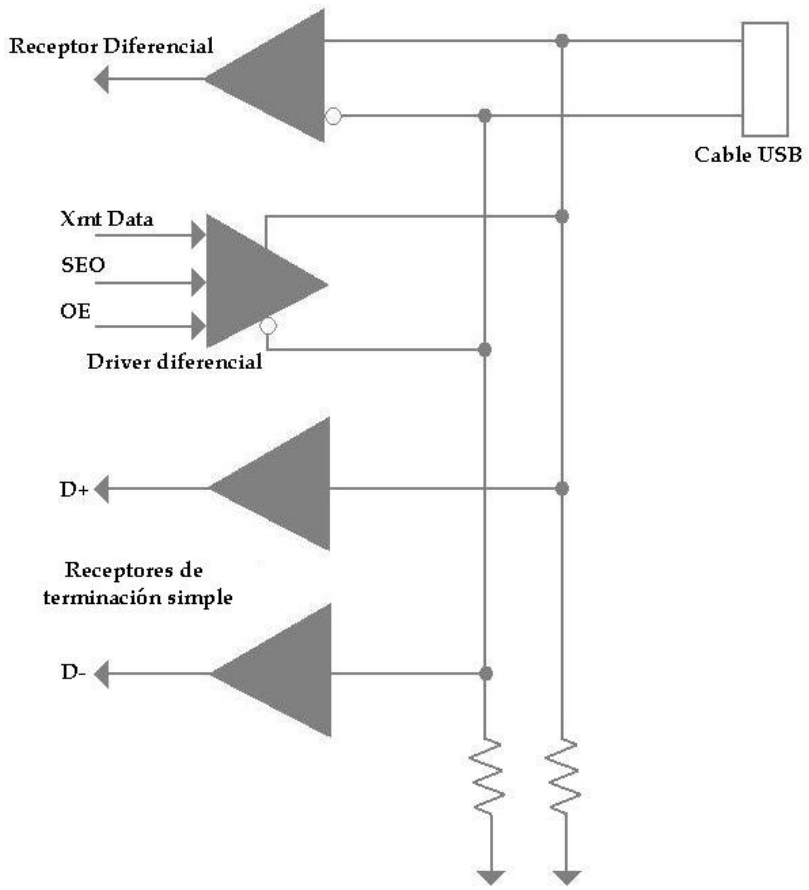


Figura 3-9 : Puerto *Hub* libre (sin conexión a dispositivo)

Tal y como se aprecia en la figura 3-9, que representa un puerto USB al que no hay conectado ningún dispositivo, las líneas D+ y D- (por las que se transfieren los datos),

están unidas a tierra a través de sendas resistencias *"pull-down"*. En esta situación, el receptor de terminación simple de cada línea, detectará un nivel bajo.

Para que un dispositivo sea detectado, deberá alterar esta situación, provocando un nivel alto en una de las dos líneas. Para ello, los dispositivos USB cuentan con una resistencia "pull-up" (conectadas a tensión positiva) en D+ ó D-, según la velocidad del dispositivo, para anunciar su presencia.

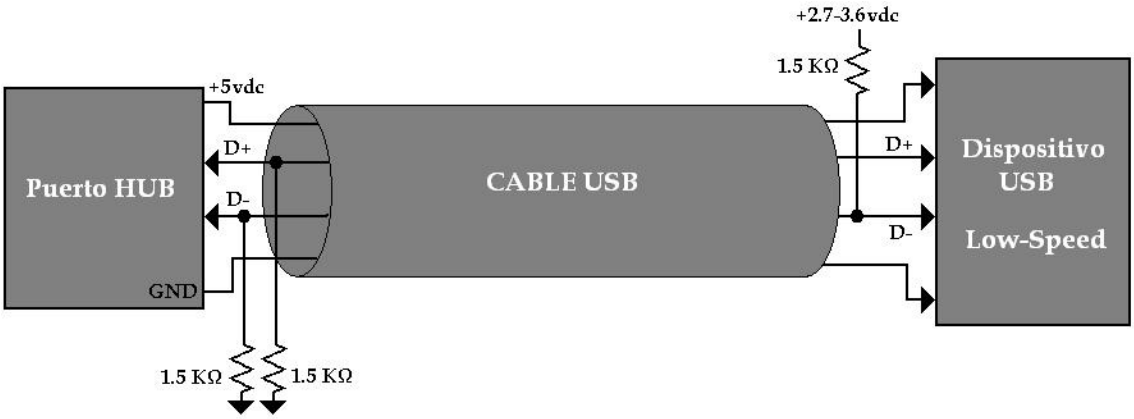


Figura 3-10 : Detección de un dispositivo Low-Speed

La línea a la que esté conectada la resistencia de *pull-up* en el dispositivo, es empleada para distinguir su velocidad de transferencia. Si esta resistencia está conectada a la línea D-, circulará corriente a través del divisor de tensión formado por la resistencia de *pull-down* (puerto *Hub*) y la de *pul-lup*, generando un nivel alto en el receptor de la línea D-, del puerto al que corresponda la conexión en el Hub; de esta manera, el dispositivo quedará identificado como *"Low-Speed"*. La figura 3-10 muestra un ejemplo de esta situación. Por otro lado, si la resistencia reside en la línea D+, el dispositivo será reconocido como *"Full-Speed"*.

El proceso de detección de un dispositivo puede durar entre 2.5µseg y 2ms, tras lo cual, el *Hub* colocará la palabra de estado apropiada, asociada al puerto, que indique el tipo de dispositivo conectado.

La figura 3-11 muestra la secuencia de pasos que se produce hasta lograr la detección del dispositivo. El estado inicial supone el puerto sin alimentación, es por ello que la secuencia arranca cediendo potencia al puerto.

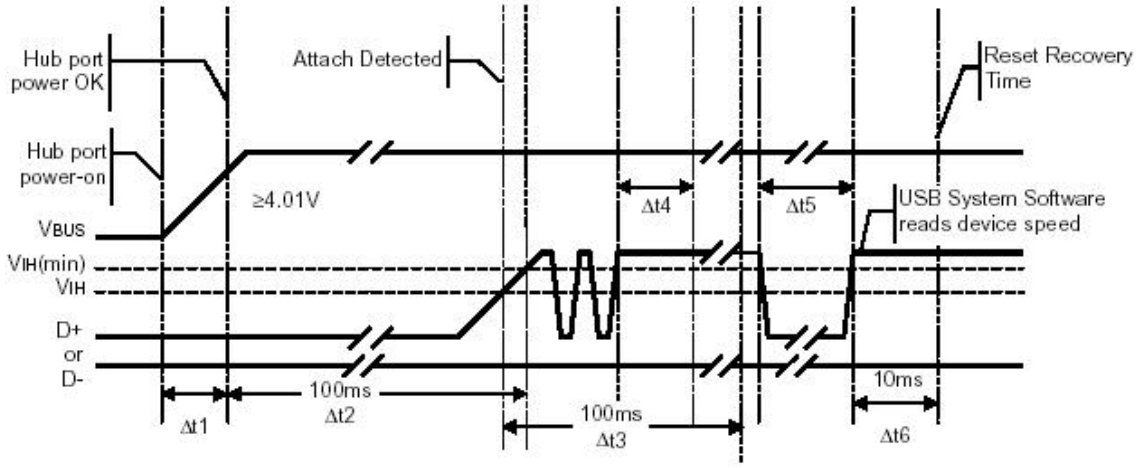


Figura 3-11 : Secuencia de detección de dispositivos LS/FS [12]

Cada uno de los pasos en que se divide el proceso, supone retardos que el software deberá respetar para asegurar el éxito de la operación. Estos retardos son:

- Δt1: Es el tiempo que necesita el *Hub* para alcanzar el valor de tensión apropiado en el puerto. Este valor puede variar entre *Hubs*, y es indicado al sistema que lo controla, mediante el *descriptor de la clase Hub*. Si el dispositivo se conecta a un puerto cuya tensión de alimentación ya ha alcanzado el nivel requerido, Δt1 vale cero.

- Δt2 : Tiempo establecido para que el dispositivo tenga oportunidad de fijar correctamente el nivel de tensión en D+ ó D- (según sea el dispositivo *FS* ó *LS*). Este tiempo debe ser inferior a 100ms, y sólo es necesario para aquellos dispositivos que obtienen la energía a través del bus.

- Δt3 : Tiempo que se debe esperar siempre (forzado por software). Se utiliza para asegurar que la conexión mecánica y eléctrica del dispositivo ha sido realizada correctamente, al menos 100ms antes de que comience el intercambio de datos.

- Δt4 : A este intervalo de tiempo se le conoce como *"bus idle"*. Una vez que el dispositivo ha fijado correctamente la tensión en D+ ó D- (según sea el caso) durante el intervalo Δt3, el dispositivo pasa a un estado de inactividad (que se produce siempre tras 3ms de inacción en el bus).

- Δt5 : Reinicio. El software de control USB envía el comando *PortReset* al Hub, el cual asignará un nivel bajo a las líneas D+ y D- (*Single-Ended Zero* : SE0) durante 10-20 ms. De esta manera, el dispositivo se ve forzado a reiniciarse, es decir, a

pasar a su estado por defecto. Este estado es requerido antes de proceder a la configuración del dispositivo.

- Δt6 : Este intervalo se conoce como el tiempo de recuperación del reset. Durante este tiempo, el dispositivo pasa a modo *suspendido*.

Los dispositivos que soporten las capacidades *"High-Speed"* (*HS*) deben inicializarse como si fueran *"Full-Speed"*, y por tanto, soportar todos sus requisitos de conexión. La transición a *HS* se hace a través de un protocolo a bajo nivel, durante el intervalo de Reinicio. [12]

3.10.2 Detección de desconexión

Un dispositivo *LS* ó *FS* estará conectado mientras exista una señal diferencial entre D+ y D- (niveles invertidos); por lo tanto, en cuando vuelva a la situación original (es decir, cero lógico en ambas líneas) durante un tiempo establecido, el dispositivo será automáticamente dado por desconectado.

Una vez desconectado, el *Hub* reestablecerá la palabra de estado asociada al puerto, indicando su liberación.

3.10.3 Señalización diferencial

El uso de señalización diferencial se debe a que con su uso se reducen ciertas fuentes de ruido, como puede ser el que añaden los amplificadores cuando tratan la señal al enviar y recibir, o el ruido que el propio cable pueda introducir debido a los campos electromagnéticos generados por sus conductores.

El hecho de contar con un único par (diferencial) para la transmisión y recepción de datos, implica que este servicio deberá ser realizado en *"half-duplex"*; es decir, un dispositivo puede enviar y recibir datos, pero no podrá realizar ambas operaciones al mismo tiempo. Por esta razón, los dispositivos electrónicos encargados de controlar el bus deben ser *tri-estado*; de esta forma, cuando no estén transmitiendo, podrán presentar *alta-impedancia* al bus, posibilitando la recepción de señal.

La representación del "1" lógico se produce cuando D+ presenta un valor de tensión mayor que D-; el "0", cuando el valor de D- es mayor que el de D+. Los niveles que el receptor debe captar son:

- "1" : (D+) - (D-) > 200mV y D+ > V_{IH}(min)
- "0" : (D-) – (D+) > 200mV y D- > V_{IH}(min)

3.10.4 Inicio de paquete (SOP)

Cada paquete comienza con una secuencia de ocho bits, cuya utilidad principal es la de sincronizar al receptor ante el paquete que se le va a enviar. La figura 3-12 representa su cronograma.

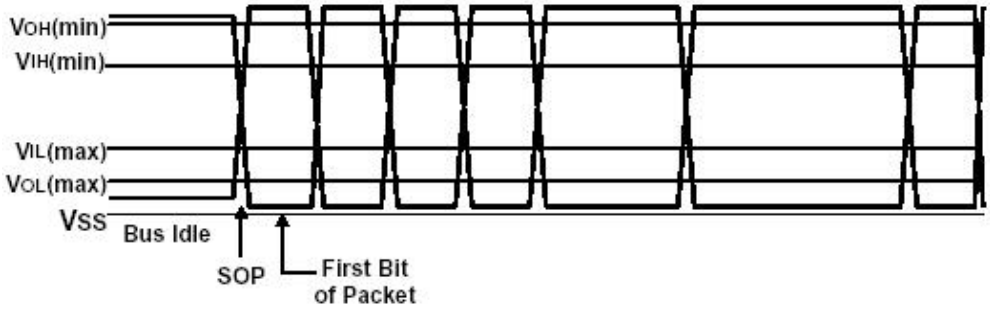


Figura 3-12 : Cronograma de la palabra de sincronización SOP[12]

3.10.5 Fin de Paquete (EOP)

La terminación de un paquete se representa mediante el envío de *SE0 (Single Ended Zero)* durante dos intervalos de bit, al final de cada paquete.

Esta situación será detectada en el receptor a través de los receptores de terminación simple, ya que el receptor diferencial no detectará señal alguna, al no haber diferencia de nivel entre las líneas de datos (tanto D+ como D- tienen asignado un nivel bajo).

3.10.6 Codificación NRZI

Los paquetes USB se codifican utilizando NRZI (No Retorno a Cero Invertido), con el objetivo de añadir un nivel más de seguridad frente al ruido (junto con la señalización diferencial), además de permitir la transferencia de datos sin la necesidad de adjuntar un reloj de sincronización.

En este esquema de codificación, el “0” es representado por una transición de señal, mientras que el “1”, por la ausencia de transición (manteniendo el nivel). Estos cambios en el nivel de la señal facilitan la sincronización con el receptor.

El problema aparece ante una larga secuencia de “1”, pues supone la ausencia de transiciones, y con ello, una posible pérdida de la sincronización. Ante esta situación, se emplea la técnica llamada “Bit Stuffing”, consistente en forzar una transición tras una secuencia de seis “1” consecutivos, que asegura transiciones ante largas secuencias.

Para mantener la integridad de los datos, el receptor siempre ignorará la transición que se produzca tras seis “1” consecutivos, ya que no se trata de un “0”, sino de un bit ficticio añadido.

3.11 Transferencias LS y FS

Durante el proceso de configuración, el software gestor de USB determina el tipo de transferencia soportada por cada uno de los registros *endpoint* del dispositivo (a través de sus descriptores). Con esta información, en caso de disponer del ancho de banda suficiente para satisfacer sus necesidades, el software del sistema principal creará un *pipe* (tubería) de comunicaciones con el dispositivo, reservando el ancho de banda solicitado.

Estos *pipes* permanecerán inactivos hasta que el software cliente (en el sistema principal) decida realizar algún tipo de transacción. Es decir, todas las transferencias realizadas por el dispositivo, son respuestas a requerimientos por parte del sistema principal.

Existen dos tipos de *pipes*: [12]

- ***Streaming Pipes*:**

  Las estructuras utilizadas para dar formato a este tipo de *pipe* no son impuestas por el sistema USB, sino que utilizan estructuras específicas de la clase, ó incluso propias del dispositivo (vendor-specific).

  Este tipo de pipe es el empleado en los registros *endpoint* configurados como *Isócrono*, *Interrupción* y *Bulk*.

- ***Message Pipes*:**

La estructura empleada queda impuesta por el sistema USB. Su uso se centra en las comunicaciones de control (como en el *endpoint* cero, que deben implementar todos los dispositivos), ya que de esta manera se crea un método universal con el que comunicarse con el dispositivo durante su configuración, cuando aún no se conoce el controlador a utilizar.

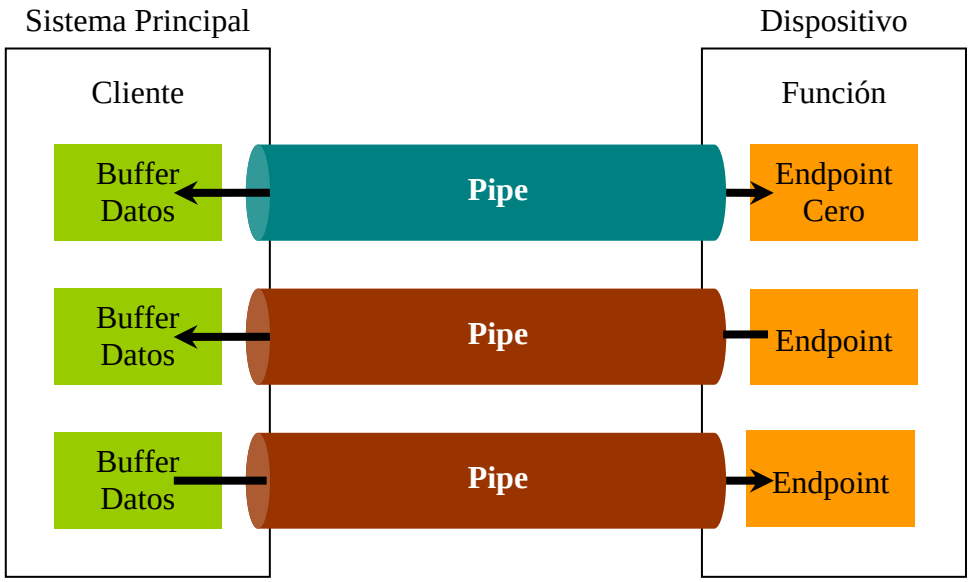


Figura 3-13 : Pipes de comunicaciones entre un dispositivo y el sistema principal

3.11.1 Tipos de transferencias

USB soporta cuatro tipos de transferencias asociadas a los *endpoint*:

3.11.1.1 Isócrona (Isochronous)

Este tipo de transferencia es el utilizado por dispositivos que requieran establecer una conexión síncrona con otro dispositivo, es decir, necesitan transmitir cierta cantidad de datos, en intervalos determinados de tiempo (como pueden ser las aplicaciones de audio). Se toma además la premisa de que en este tipo de conexiones es más importante la entrega "a tiempo", que la integridad de los datos; por esta razón, no se garantiza la entrega correcta de los datos.

Las transferencias siempre deben ser en un solo sentido; es decir, si se requieren transferencias de este tipo en ambos sentidos, deberán implementarse dos registros *endpoint*: uno de entrada y otro de salida.

El tamaño útil de datos es de 1023 bytes por cada *frame*, y no podrá ser utilizado por dispositivos LS. Si un dispositivo necesitara un *payload* mayor (que se traduce en mayor ancho de banda) que el disponible en el sistema, el dispositivo no sería configurado, ya que este tipo de transferencias requieren la garantía del ancho de banda.

El hecho de no existir una implementación de un modo de transferencia síncrono, sino isócrono, obliga a los dispositivos que requieren transferir a intervalos precisos de tiempo, a utilizar estructuras de sincronización sobre las transferencias isócronas. Esto se debe a que el sistema isócrono enviaría de una sola vez (si le es posible) el bloque de datos generado por la fuente desde el *frame* anterior, sin ningún tipo de estructura, ni reloj que indique la relación de transferencia a la que deben ser interpretados los datos. Existen tres tipos de sincronización:

- **Asíncrono:**

  El dispositivo no puede sincronizar la transferencia con el reloj del sistema USB (basado en el primer instante del frame). Por tanto, los datos se sincronizan con un reloj interno del dispositivo, que se inicia con cada paquete.

- **Síncrono:**

  Los registros *endpoint* que soportan este tipo, podrán sincronizar las transferencias con el instante en el que se inicia el *frame* (es decir, con el reloj del sistema USB).

- **Adaptado:**

  En este tipo de sincronización, la tasa de transferencia puede variar dentro de unos márgenes establecidos en la configuración. Con el objetivo de adaptar la velocidad de transferencia a las exigencias del receptor, incorpora un mecanismo de retroalimentación (ó Feedback), con el que el sistema que envía información puede conocer los cambios producidos en la conexión, y “adaptarse” a ellos.

  El campo de *Feedback* está compuesto por un valor entero que especifica el número de muestras por *frame*, y un valor que representa la porción de muestra necesaria para establecer la sincronización con la fuente, a una frecuencia de 1 Hz (utilizada para determinar la cantidad de muestras por segundo). Este campo se compone de 3 bytes.

3.11.1.2 Interrupción (Interrupt)

Este tipo de transferencia es utilizada para preguntar a los dispositivos si tienen datos para enviar. Si no hay datos pendientes de envío, enviará un paquete *NACK (No ACKnowledge)*.

Un uso extendido de estas transferencias es el de preguntar a los dispositivos, a intervalos determinados de tiempo, si tienen algo que enviar. Estos intervalos son definidos en un campo destinado para ello en el descriptor del *endpoint*. En los dispositivos FS este intervalo puede variar entre 1 y 255 frames; en los LS, en cambio, el intervalo mínimo es de 10 *frames*.

En los dispositivos FS, el tamaño del campo de datos debe ser de 64 bytes, excepto en la última transacción, que deberá ser de un tamaño inferior (cualquier paquete de un tamaño inferior a 64 bytes será tomado como el último).

También se incluye soporte para la corrección de errores: si se detecta un error durante la transacción, el paquete será reenviado en el siguiente frame, una vez transcurrido el intervalo especificado.

3.11.1.3 Control (Control Transfers)

Proveen un mecanismo para configurar los dispositivos, y controlar ciertos aspectos de su modo de operación. Cada dispositivo debe incorporar un *endpoint* de control por defecto, llamado *endpoint zero* (EP0), a través del que se realiza la configuración inicial.

El *endpoint* de control está preparado para responder a ciertas peticiones específicas, como puede ser la entrega de los descriptores. También, según el contexto, es posible crear nuevas peticiones (con sus respuestas), dependiendo de la clase, ó incluso para un dispositivo concreto.

Una transferencia de control pasa por al menos dos de las etapas siguientes:

- **Etapa de configuración:**

  Las transferencias de control siempre empiezan en esta etapa, encargada de enviar información al dispositivo destino, en la que se define el tipo de petición realizada al mismo.

- **Etapa de datos:**

  Sólo se da para aquellas peticiones que requieran una transferencia de datos, como pueden ser las de petición de descriptores.

- **Etapa de estado:**

  Todas las transferencias han de pasar por esta etapa, ya que en ella se muestra el resultado de la operación solicitada.

En la etapa de configuración, las transferencias son de 8 bits, en los que, entre otras cosas, se determina la longitud de los datos que deberá recibir como respuesta (en caso de requerirlos), cuyo valor máximo es de 64 bytes.

Las transferencias de control tienen reservado un 10% del ancho de banda del sistema; en caso de mayor disponibilidad, ésta será utilizada.

3.11.1.4 Bulk

Son utilizadas por los dispositivos que no están sometidos a la exigencia de la transmisión de datos en intervalos precisos de tiempo. El ejemplo típico de esta transferencia es el de una impresora, ya que la operación se realizará correctamente, aunque la entrega de los datos sea lenta (siempre dentro de unos márgenes).

Debido al hecho de que no está sujeta al tiempo, esta transferencia es relegada a la prioridad más baja. Por así decirlo, se trata de la transferencia más solidaria en cuanto al ancho de banda, ya que no lo reserva ni lo ocupa siempre, sólo lo utiliza si hay disponibilidad.

Los tamaños establecidos para los datos son de 8, 16, 32 ó 64 bytes unicamente.

Dado que en estas transacciones, la integridad de los datos es mucho más importante que el momento en el que llegan, soportan sistemas de recuperación y corrección de errores.

3.12 Paquetes USB

Normalmente, las transacciones están formadas por tres paquetes básicos (figura 3-14); el hecho de ser la misma transacción no implica que todos sus paquetes vayan en el mismo

sentido, ya que un paquete puede ser la respuesta al anterior, tal y como la transacción lo solicita:

- **Paquete Testigo:**

Toda transacción comienza con este paquete, el cual define el dispositivo destino, su identificador de *endopoint*, y la dirección del buffer de datos. Se distinguen cuatro tipos de paquetes testigo:

  - SOF (Start Of Frame) : indica el inicio de un *frame*. Es el único paquete de testigo que es enviado a todos los dispositivos (broadcast).
  - IN : Indica que el sistema principal desea leer datos del dispositivo.
  - OUT : Indica que el sistema principal desea enviar datos al dispositivo.
  - SETUP : Indica el comienzo de una transferencia de control, y es utilizado para enviar una petición al dispositivo

- **Paquete de datos:**

Consiste en un paquete cuya carga útil es información de usuario. Su tamaño depende del tipo de transacción que se lleve a cabo, pero la longitud máxima establecida es de 1023 bytes (transferencias isócronas).

- **Paquete de confirmación:**

Todas las transferencias, excepto las isócronas, utilizan este tipo de paquetes para asegurar la correcta transferencia de los datos. Para indicar un error, el receptor debe omitir el envío del paquete de confirmación, que es argumento suficiente para detectar alguna irregularidad. Ante esta situación, cada dispositivo tiene su forma de recuperarse. Existen tres tipos de paquetes:

  - ACK : Reconocimiento del paquete sin error.
  - NACK : Indica al sistema principal que el dispositivo temporalmente no puede aceptar o enviar datos. En las transferencias de *interrupción*, indica que no hay datos para enviar.

- STALL : Indica al sistema principal que el dispositivo no es capaz de completar la transferencia. El software del sistema deberá tomar parte para sacar al dispositivo del estado STALL (se encuentra en un estado de excepción).

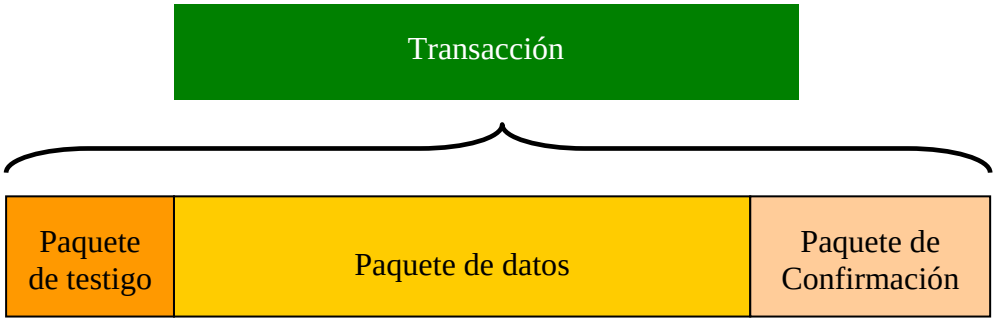


Figura 3-14 : Transferencia típica de tres paquetes

Existe un tipo de paquete especial, denominado *"Preamble"*, cuya finalidad es la de avisar al *Hub* (por parte del sistema principal) de que la transacción a realizar es de *LS*. La necesidad de este paquete reside en el hecho de que un *Hub* debe impedir que las transferencias *FS* puedan llegar a los dispositivos *LS* (para evitar problemas); es decir, que mientras no se indique lo contrario (con *Preamble*), todas las transferencias son FS, y por tanto, no son vistas por los dispositivos LS.

3.12.1 Formato de los paquetes

Para conformar un paquete, además de la carga útil del mismo, debe añadirse ciertos campos que aseguren su cometido, tal y como muestra la figura 3-15:

- **Secuencia de sincronización:**

  Consiste en una serie de 8 bits, con 7 "0" lógicos consecutivos, y un "1" lógico al final. Se realiza de esta manera debido a que una vez se codifique en NRZI, los ceros provocan transiciones en la señal, facilitando la sincronización del reloj del receptor, a la tasa de transferencia del paquete que viene a continuación.

- **Identificador de paquete (PID):**

  Indica el tipo de paquete que se envía, es decir, su propósito. Los paquetes pueden ser de testigo, de datos, de confirmación, ó paquetes especiales. Se trata de un campo de 8 bits, donde los 4 primeros representan el PID, y los cuatro siguientes se

utilizan como comprobación, transmitiendo el PID, pero complementado (invirtiendo los bits)

- **Información específica:**

Campo que contiene la información objeto del paquete (carga útil con información del cliente), además de ciertos datos relativos al mismo, como la dirección del dispositivo a quien va dirigido ó el número de *frame* en el que fue transmitido.

- **CRC (Código de Redundancia Cíclica):**

Campo utilizado para comprobar la integridad del campo de *información específica*, y detectar errores en el mismo. Dependiendo del tipo de paquete puede ser de 5 ó 16 bits.

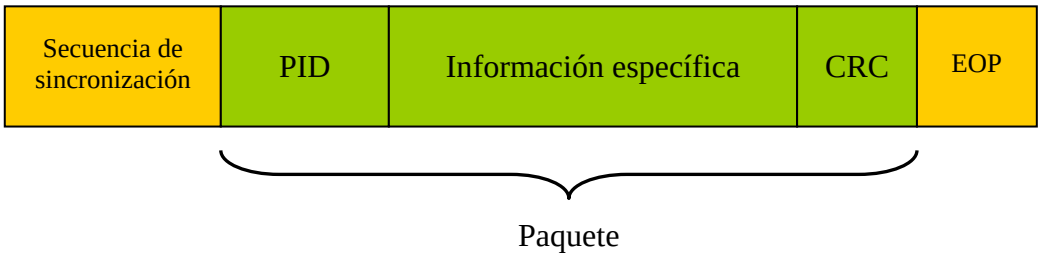


Figura 3-16 : Formato de los paquetes USB

3.13 Transacciones

Las transacciones representan la ejecución de las operaciones, solicitadas por el software cliente del sistema principal, mediante el envío y recepción de los paquetes oportunos. Existen tres tipos de transacciones.

3.13.1 Transacciones de entrada (IN Transactions)

Con estas transacciones, el sistema principal intenta leer datos del dispositivo. Normalmente requiere que se cumplan las tres fases básicas (*testigo*, *datos* y *confirmación*), aunque pueden darse varios casos:

- **Transacción sin errores:**

Cumple las tres fases: envía un testigo, recibe el paquete de datos, y se envía un *ACK*. Ver figura 3-17.

- **Transacción con errores:**

En este caso, la transacción sólo emplea dos paquetes, el de testigo, y el de datos, pues el de confirmación no se envía. El dispositivo tiene un mecanismo de time-out para los paquetes, de esta forma, si el de confirmación no ha llegado en el tiempo establecido, se toma como un error.

- **Transacción sin interrupción pendiente o con destino ocupado:**

Si el dispositivo actualmente no puede enviar información, enviará un *NACK* en la fase de datos, avisando de esta manera al sistema principal de tal incidencia. También puede ocurrir en transferencias por *interrupción*, si no hay interrupciones pendientes con el dispositivo, ó durante las *bulk* y de *control*, si el dispositivo está ocupado.

- **Transacción con destino en estado de excepción (STALL):**

Cuando el destino de la transacción se encuentra en la condición *STALL*, informará al origen de esta situación, mediante un paquete de confirmación de tipo *STALL*. De esta manera, el sistema principal podrá actuar sobre el dispositivo para sacarlo del estado STALL, y proseguir con la transferencia de datos.

- **Transacción en transferencias Isócronas:**

En este caso la transacción consiste en una fase de testigo y una de datos, pues en principio no puede enviar paquete de confirmación (las transferencias isócronas son en un solo sentido).

Figura 3-17 : Transacción de entrada sin error

3.13.2 Transacciones de salida (OUT Transactions)

Con estas transacciones, el sistema principal intenta enviar datos al dispositivo. Normalmente cumplen las tres fases básicas (*testigo*, *datos*, *confirmación)*, aunque pueden darse varios casos:

- **Transacción sin errores:**

Se dan las tres fases: se envía un testigo, luego los datos, y finalmente el dispositivo devuelve un ACK. Ver figura 3-18.

- **Transacción con errores:**

Para indicar que se ha producido un error, el dispositivo se limita a omitir el paquete de ACK (confirmación). A partir del time-out de los paquetes, el sistema principal podrá advertir la situación.

- **Transacción en la que el destino no puede recibir datos:**

Debido a que el dispositivo se encuentra ocupado, no puede aceptar los datos de entrada, por lo que en el paquete de confirmación envía un NACK; esto indicará al sistema principal que el dispositivo no aceptó los datos, por lo que la transacción deberá ser repetida más tarde.

- **Transacción con destino en estado de excepción (STALL):**

Cuando el destino de la transacción se encuentra en la condición *STALL* no podrá aceptar los datos, por lo que informará al origen de esta situación, mediante un paquete de confirmación *STALL*. De esta manera, el sistema principal podrá actuar sobre el dispositivo para sacarlo del estado STALL, y proseguir con la transferencia de datos.

- **Transacción en transferencias Isócronas:**

En este caso la transacción consiste en una fase de testigo y una de datos, pues en principio no puede enviar paquete de confirmación (las transferencias isócronas son en un solo sentido).

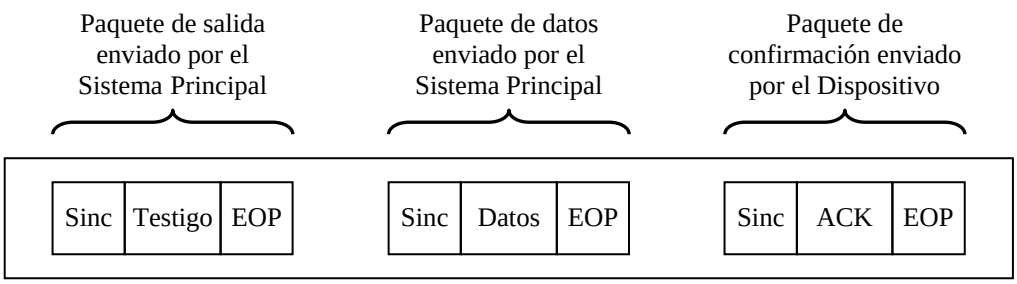


Figura 3-18 : Transacción de salida sin errores

3.13.3 Transacciones de configuración y transferencias de control

Las transferencias de control siempre comienzan con una transacción de configuración (8 bytes), que define el objetivo de la transferencia de control a la que se va a proceder. A continuación pueden haber una o varias transacciones de entrada o de salida. El último paso es el de *estado*, utilizado como confirmación de toda la operación. En general, existen dos tipos de transacciones de control, en dos pasos ó en tres:

- **Transferencia de control en dos pasos:**

  En este caso, los 8 bytes de la transacción de configuración son empleados para enviar toda la información necesaria para realizar la operación. El paso de *estado* consistirá en una transacción de entrada que verificará la correcta realización del proceso.

- **Transferencia de control en tres pasos con fase de entrada de datos:**

  Se trata de una transferencia en la que se requiere una lectura de datos por parte del sistema principal. Comienza con una transacción de configuración, seguida por una o varias de entrada, para finalmente requerir una transacción de salida a modo de confirmación; en esta última transacción, el paquete de datos tiene una longitud de cero bytes, pues el único paquete importante es el de confirmación. Ver figura 3-19.

- **Transferencia de control en tres pasos con fase de salida de datos**:

  Comienza con una transacción de configuración, seguida de una o varias de salida, para finalizar en una transacción de entrada que hace las veces de confirmación (cuyo campo de datos es de cero bytes).

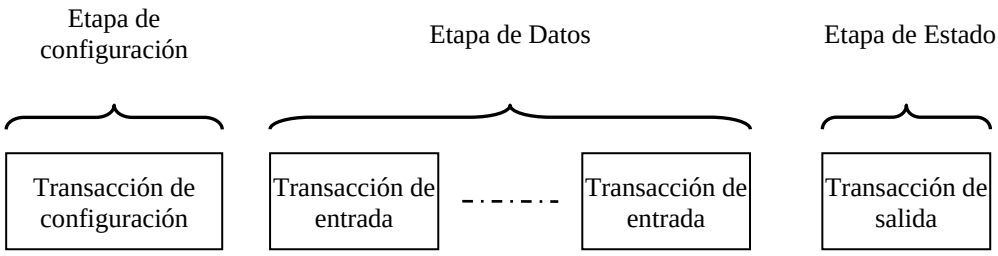


Figura 3-19 : Transferencia de control en tres pasos con entrada de datos

4. Conceptos generales del PIC 16C745

En este capítulo se realiza una descripción general de los microcontroladores PIC. En primer lugar, se argumenta la evolución, necesidad y usos de los microcontroladores. Posteriormente se presentan las características y familias de los PIC, haciendo especial hincapié en los recursos que ofrece el 16C745. Finalmente, se explica el proceso de programación y borrado, a llevar a cabo en el proceso de desarrollo.

4.1 Los Microcontroladores

La aparición de la electrónica, y ante todo la electrónica digital, ha provisto de un método de control automatizado, y en ciertos campos, hasta inteligente, a la gestión de procesos. Hoy en día son pocas las cosas que pueden escapar al gobierno de un controlador.

La implementación física de los controladores electrónicos, a raíz de los nuevos avances tecnológicos, ha variado de forma considerable a lo largo del tiempo. Lo que en un principio quedaba cubierto con el empleo de lógica discreta, poco a poco ha ido dando paso a sistemas más complejos basados en el *microprocesador*.

Un microprocesador es básicamente una unidad de gestión de recursos, pero que por sí misma no es capaz de hacer gran cosa. Esto se debe a que no cuenta con un interfaz directo, a través del que interactuar con el exterior, ni ningún tipo de soporte en el que almacenar las instrucciones a llevar a cabo. Para cubrir estas carencias, los microprocesadores se rodean de otros elementos, como *chips* de memoria y de gestión de E/S.

Según ha ido avanzando la tecnología, y el empeño por reducir el tamaño de los dispositivos electrónicos, se ha logrado incluir en un único *chip*, el microprocesador y todos aquellos elementos necesarios para su funcionamiento, dando paso al concepto de *microcontrolador*. Además, los microcontroladores pueden llegar a especializarse en campos determinados, incluyendo para ello, periféricos afines al área de actuación.

El extenso uso de los microcontroladores, ante todo en sistemas *empotrados* (o *embebidos*), se debe en gran medida a las ventajas que ofrecen frente a los microprocesadores, como son:

- **Menor coste :**

El gasto al que hay que hacer frente en un sistema basado en microcontroladores, es claramente inferior al desembolso que supone el microprocesador, junto con todos aquellos componentes que lo dotan de una funcionalidad mínima, además de los empleados para funciones específicas.

- **Simplicidad del diseño:**

El empleo de un único chip, que normalmente cubre las necesidades del proyecto en que se utiliza, supone un conexionado sencillo a los posibles dispositivos adicionales necesarios para su funcionamiento, y por supuesto, a los elementos que controla.

- **Tamaño reducido:**

La gran integración conseguida en los microcontroladores, incorporando gran cantidad de dispositivos, ofrece la posibilidad de que muchos diseños se limiten al uso de un único integrado, que en algunos casos se encuentra acompañado por elementos eléctricos pasivos, como resistencias y condensadores. Es decir, la superficie ocupada es mínima.

Este hecho facilita la portabilidad de los sistemas creados, permitiendo su uso e instalación en lugares pequeños, ó de difícil acceso.

- **Fiabilidad:**

La sustitución de un elevado número de elementos por un único dispositivo, disminuye el riesgo de averías, se precisan menos calibraciones y simplifica la búsqueda y reparación de errores.

4.2 Microcontroladores PIC

Los *PIC* (*Peripheral Interface Controller*), fabricados por la empresa *Microchip*, forman parte de una de las familias de microcontroladores más importantes del mercado actual.

Las claves de su éxito se centran en su utilización; una vez se conoce la arquitectura y el repertorio de instrucciones de uno de ellos, es realmente sencillo abordar otro modelo. Obviamente, también cobra gran importancia su utilidad y sencillez de uso. Además, existe

un gran número de aplicaciones, como compiladores y simuladores, que Microchip distribuye gratuitamente (y otras tantas específicas, previo pago); estos factores facilitan en gran medida el empleo a gran escala de estos dispositivos.

Figura 4-1 : Microcontrolador PIC, con encapsulado OTP

4.2.1 Características principales

4.2.1.1 Arquitectura Harvard

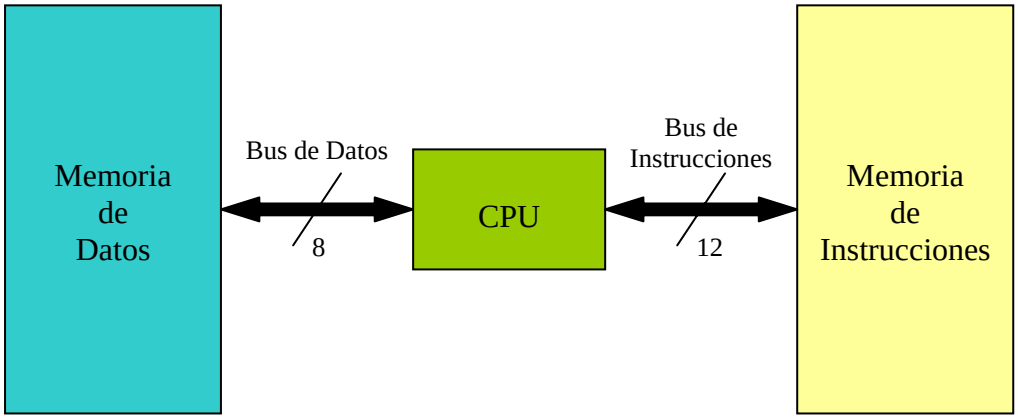


Figura 4-2 : Modelo de la arquitectura Hardvard en PICs de gama baja

Los microcontroladores PIC basan su arquitectura en el modelo *Harvard* (ver figura 4-2). Este modelo basa su funcionamiento en la separación de los datos y las instrucciones en memorias distintas; es decir, utiliza una memoria sólo para almacenar datos y otra para albergar las instrucciones del programa a ejecutar. Cada una de estas memorias se conectan a la CPU (*Central Process Unit*) de forma independiente, y mediante buses distintos. Este tipo de organización posibilita el acceso simultáneo a datos e instrucciones.

4.2.1.2 Segmentación

Los *PIC* utilizan la técnica denominada *"pipe-line"*, que permite la ejecución de una instrucción, mientras se busca el código de la siguiente. Esta característica permite ejecutar cada instrucción en un único ciclo (un ciclo de instrucción equivale a cuatro ciclos de reloj).

Como excepción, las instrucciones de salto requieren dos ciclos, ya que no es posible conocer la dirección de la siguiente instrucción a ejecutar, hasta que no se haya completado la bifurcación.

4.2.1.3 Procesador RISC

Este tipo de procesadores presentan como característica principal, *un juego de instrucciones reducido*. Los PIC de gama baja cuentan con un repertorio de 33 instrucciones, los de gama media 35, y casi 60 los de la alta.

4.2.1.4 Ortogonalidad de las instrucciones

Cualquier instrucción puede manejar cualquier elemento de la arquitectura, como fuente o destino [15].

4.2.1.5 Arquitectura basada en bancos de registros

Desde el punto de vista del usuario, todos los objetos (memoria, periféricos...) se encuentran situados en un mismo banco de registros. Para acceder a un objeto determinado, tan sólo se requiere apuntar a la dirección del registro adecuado.

4.2.2 Familias de PIC

No todos los problemas requieren el empleo de controladores de altas prestaciones, ni de aquellos que cuentan con gran número de periféricos, pues las soluciones a problemas sencillos se verían perjudicadas, sobre todo, en costes; por otro lado, los proyectos de gran envergadura requieren la existencia de microcontroladores con recursos numerosos y potentes. Por ello, con la finalidad de adaptarse lo mejor posible a los requisitos de cada aplicación, existen tres gamas de PIC:

- **Gama Baja**

  Agrupa a los microcontroladores más sencillos y baratos. La memoria de instrucciones puede contener 512, 1k ó 2k palabras de 12 bits, y puede ser ROM ó EPROM. La memoria de datos, por otro lado, puede albergar entre 25 y 73 bytes.

  Cuentan con un único *timer* y entre 12 y 20 pines de E/S.

- **Gama Media**

Añade abundantes prestaciones a las ya incluidas en la gama baja, lo que los hace adecuados para proyectos de mayor envergadura.

Entre estas mejoras encontramos PIC que admiten interrupciones, poseen comparadores de magnitudes analógicas, conversores A/D, puertos serie, diversos temporizadores, y últimamente, puertos USB.

Existen modelos cuya memoria de instrucciones es EEPROM o incluso Flash, lo cual facilita la programación y el ensayo, ya que el borrado y reprogramación es más rápido y simple que con las memorias EPROM (que requieren luz ultravioleta). También existen modelos cuya memoria de programa es OTP (*One Time Programmable*), que sólo permiten grabar una vez, pero que resultan mucho más económicos para prototipos y series pequeñas.

- **Gama Alta**

Las principales características que los distancian de la gama media, son las de contar con mayores capacidades de memoria y algunos recursos mejorados. Además, hay modelos de *"arquitectura abierta"*; es decir, permiten la expansión del microcontrolador, en la misma medida que lo haría un microprocesador. Para ello, utilizan las líneas de E/S para sacar al exterior los buses de datos, de dirección y de control, lo cual permite la expansión de la memoria y de la E/S, utilizando circuitos integrados externos.

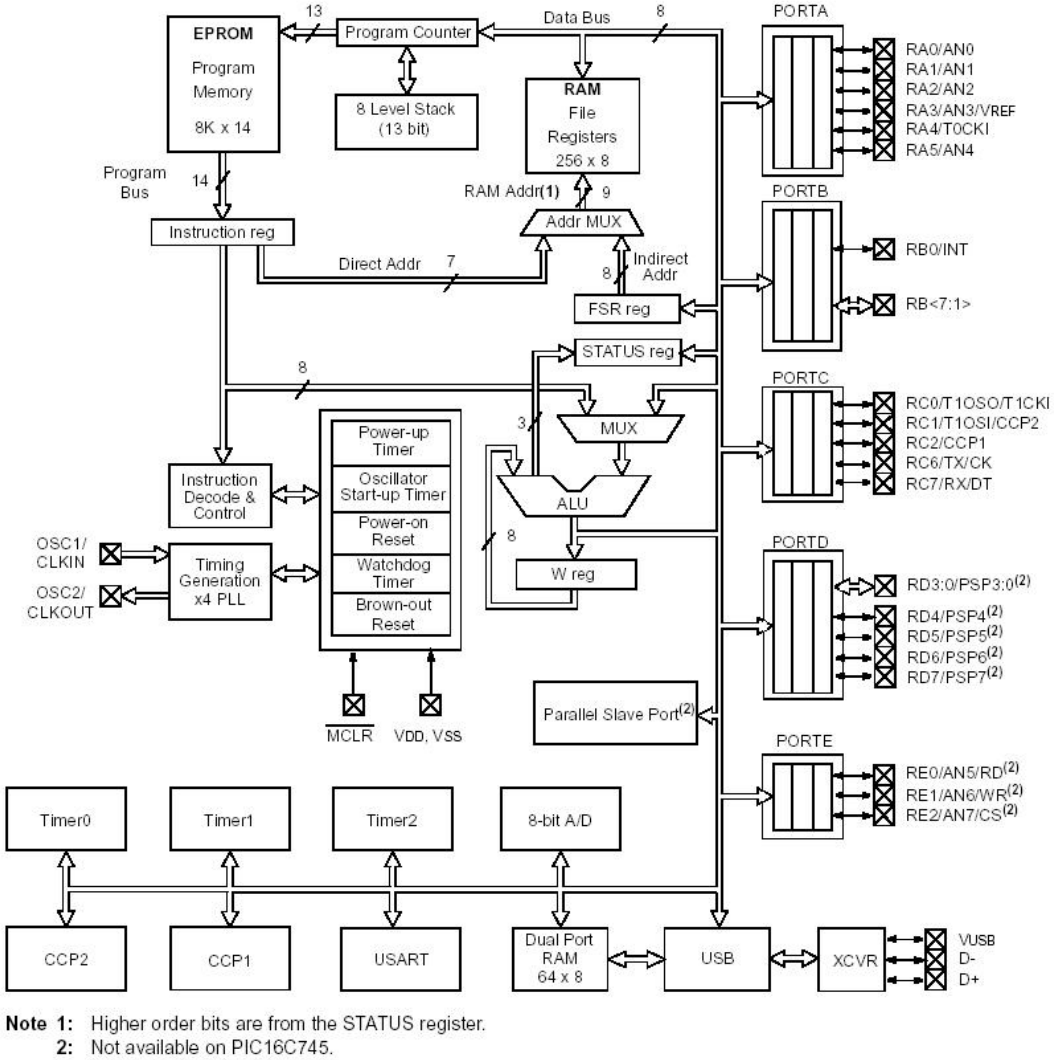
4.3 PIC 16C745

Se trata de un microcontrolador de la gama media, dentro de la clasificación de los PIC.

Cuenta con palabras de instrucción de 14 bits, y de datos de 8 bits. Dispone de un repertorio de 35 instrucciones.

Entre los recursos encontramos 22 pines de E/S, capacidad para 8k palabras de 14 bits para la memoria de instrucciones, 256 bytes de RAM, 3 temporizadores, 2 módulos de captura/comparación/PWM (Pulse-Width Modulation), 2 puertos serie, 1 conversor A/D de 5 bits y un interfaz USB 1.1. La figura 4-3 muestra la organización interna de los PICs

16C745 / 16C765, ya que ambos presentan características muy similares; la principal diferencia reside en que el 16C765 cuenta con un mayor número de pines (40, frente a los 28 del 16C745), por lo que algunos recursos se ven ampliados).



Note 1: Higher order bits are from the STATUS register.
2: Not available on PIC16C745.

Figura 4-3 : Organización interna de los PICs 16C745/16C765 [16]

Este microcontrolador ha sido especialmente diseñado para reducir el número de componentes externos necesarios para su funcionamiento; es por ello que cuenta con cuatro modos de oscilador: *EC*, para ser controlado por un reloj externo, *E4* que también permite un reloj externo, pero con el PLL activado, *HS* da soporte a cristales de alta velocidad, y *H4* que también utiliza cristales de cuarzo, pero con el PLL activado.

Actualmente existen dos tipos de encapsulado para este microcontrolador:

- **OTP**

El encapsulado *One Time Programmable* es el utilizado para la construcción de prototipos y la producción de pequeñas series, debido al hecho de que sólo se puede programar una vez, y que se trata de un encapsulado económico.

- **UV erasable CERDIP**

Este tipo de encapsulado cuenta con una pequeña ventana en la parte superior del integrado, cuya exposición a la radiación ultravioleta provoca el borrado de la memoria del microcontrolador. Por esta razón, este encapsulado es el utilizado durante la fase de desarrollo del prototipo, ya que permite borrar y reprogramar el microcontrolador.

4.3.1 Puertos de E/S

El PIC 16C745 cuenta con tres puertos de E/S. Cada uno de estos puertos cuenta con una dirección de registro donde se leen o escriben los datos, y otra dirección en la que se configura el funcionamiento del mismo. Los puertos son:

- **Puerto A**

Se trata de un latch de 6 bits (RA0 – RA5), que acepta y transmite niveles TTL. Este puerto comparte los pines con el conversor A/D

- **Puerto B**

Puerto bi-direccional de 8 bits (RB0 – RB7). Los pines 4-7 cuentan con posibilidades de interrupción; es decir, previa configuración, son capaces de iniciar una interrupción ante un cambio en el estado de alguno de estos pines.

- **Puerto C**

Puerto bi-direccional de 5 bits. Se encuentra multiplexado con otros recursos, como el puerto serie ó el reloj contador del Timer1.

El microcontrolador permite especificar, mediante el registro correspondiente, el sentido (entrada ó salida) de cada uno de los pines del puerto sobre el que se quiere actuar.

4.3.2 Temporizadores

Hay disponibles tres tipos de temporizadores:

- **Timer0**

Se trata de un temporizador/contador de 8 bits. Cuenta con un divisor de frecuencia de 8 bits, programable por software, que se utiliza para controlar tiempos mayores que los permitidos por los 8 bits del timer. Este divisor de frecuencia también es empleado por el WatchDog (timer interno que evita bucles infinitos, reiniciando el microcontrolador cuando su registro se desborda), pero no por ambos a la vez; es por ello que se debe asignar a uno u otro, según convenga.

El valor del registro que lleva la cuenta del timer es accesible en todo momento, ya sea para introducir un nuevo valor, como para leer la cuenta actual.

Básicamente existen dos modos de funcionamiento:

- Contador : Cuenta el número de impulsos que se producen en el pin T0CK1. Es decir, lleva el cómputo de las ocurrencias de un acontecimiento externo.

- Temporizador : Cuenta los impulsos del temporizador interno (resultante de dividir la frecuencia del oscilador entre 4), teniendo en cuenta la asignación del divisor de frecuencia; es decir, incrementa su cuenta cada (F_{osc} / 4) * Rango_Divisor. Se utiliza para calcular intervalos de tiempo fijos.

En cualquier caso, una vez se desborde el contador, indicará una interrupción. Es especialmente útil para controlar tiempos.

- **Timer1**

Temporizador/contador ascendente de 16 bits, consistente en dos registros de 8 bits (TMR1H y TMR1L). La fuente de impulsos se aplica a un divisor de frecuencia que los divide por 1, 2, 4 ú 8.

Al igual que el Timer0, funciona en los modos de contador o temporizador, lanzando una interrupción cuando pasa de FFFFh a 0000h, si así se establece.

- Contador : Incrementa el valor de sus registros conforme a un oscilador externo, conectado a los pines RC0/T1OS0/TICK1 y RC1/T1OS1/CCP2.

- Temporizador : Incrementa el valor de sus registros, conforme al oscilador interno, una vez tenido en cuenta el divisor de frecuencias.

Es posible configurarlo para sacar del estado de reposo al sistema.

- Timer2

Temporizador de 8 bits, diseñado para utilizarse conjuntamente con el circuito de Modulación de Anchura de Pulsos (PWM).

4.3.3 Módulo USB

A través de este periférico, el 16C745 es capaz de dar soporte USB 1.1. Debido a sus limitaciones, sólo es capaz de dar servicio como dispositivo *"Low-Speed"*. Las transacciones soportadas son las de control (obligatorias en todo dispositivo USB), y de interrupción (tanto de entrada como de salida). Para ello cuenta con 3 registros *endpoint* (0,1,2), configurables según necesidades.

Microchip provee al diseñador de un firmware [17] que realiza las operaciones básicas de todo sistema USB, como son la lectura y la escritura, o el proceso de configuración (enumeración). De esta manera, se simplifica el diseño e implementación de aplicaciones que utilicen USB, ya que el diseñador se "limita" a ejecutar funciones del firmware, que se ocupan de llevar a cabo todas las operaciones oportunas, una vez le sean indicados ciertos parámetros.

El firmware debe enviar los descriptores del dispositivo, cuando el sistema principal los solicite en el proceso de configuración. Estos descriptores deben ser creados por el programador durante la etapa de diseño, siguiendo el modelo dado por la especificación USB [12] para los descriptores estándar, y las especificaciones de clase, cuando el dispositivo pueda ser agrupado dentro de una clase USB.

4.4 Desarrollo del software

Para escribir el código ensamblador del PIC, puede utilizarse cualquier procesador de textos convencional. Una vez editado, será el ensamblador el que realice las operaciones

oportunas, teniendo como base este código, para dar lugar a un archivo binario, que contiene las instrucciones que deben ser enviadas a la memoria de programa del PIC.

El programa ensamblador lo suministra *Microchip* de forma gratuita, a través de su página Web. Su nombre es *"MPASM"*. El fabricante ofrece, también de forma gratuita, la *suite* de programación *"MPLAB"*, que además de contar con un más que aceptable procesador de textos (pues realiza de forma sencilla operaciones comunes en la programación de código ensamblador), permite simular y depurar el código creado; esta herramienta se hace indispensable en la etapa de diseño, ya que permite conocer los resultados y posibles errores, sin necesidad de programar el microcontrolador. La figura 4-4 muestra la ventana principal del *MPLAB*.

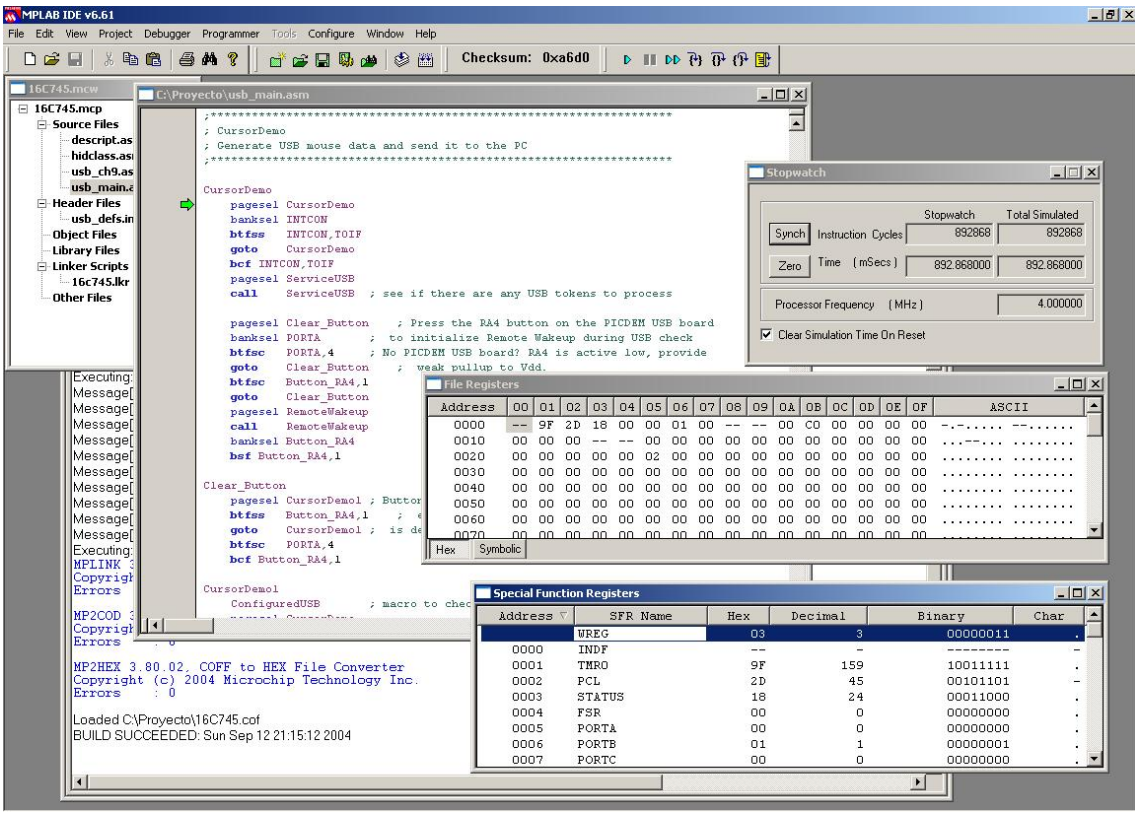


Figura 4-4 : Entorno de programación MPLAB

Una vez hemos obtenido el archivo binario, tan sólo queda un último paso: enviar el programa al microcontrolador. Para ello, se requiere de un *hardware* denominado *"programador"*, que conoce el procedimiento de inserción de las instrucciones en el microcontrolador (temporizaciones, patillaje, tensiones...), o al menos, la forma de obtenerlo de un ordenador al que debe estar conectado. Dispone de un zócalo, al que se conecta el dispositivo a programar, y de una interfaz para la conexión con un ordenador. La figura 4-5 muestra la fotografía de uno de estos programadores.

El envío del código al programador, se realiza a través de una aplicación capaz de interactuar con este tipo de hardware. Uno de los programas más utilizados para este fin es el *"IC-Prog"* (ver figura 4-6), debido a su fiabilidad, facilidad de uso y al elevado número de dispositivos soportados. Puede obtenerse de forma gratuita en:

http://www.ic-prog.com/

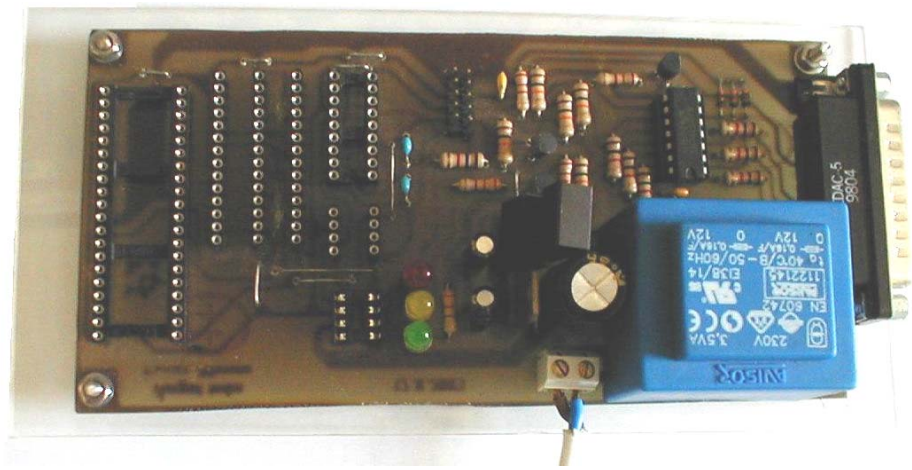


Figura 4-5 : Programador de Pics

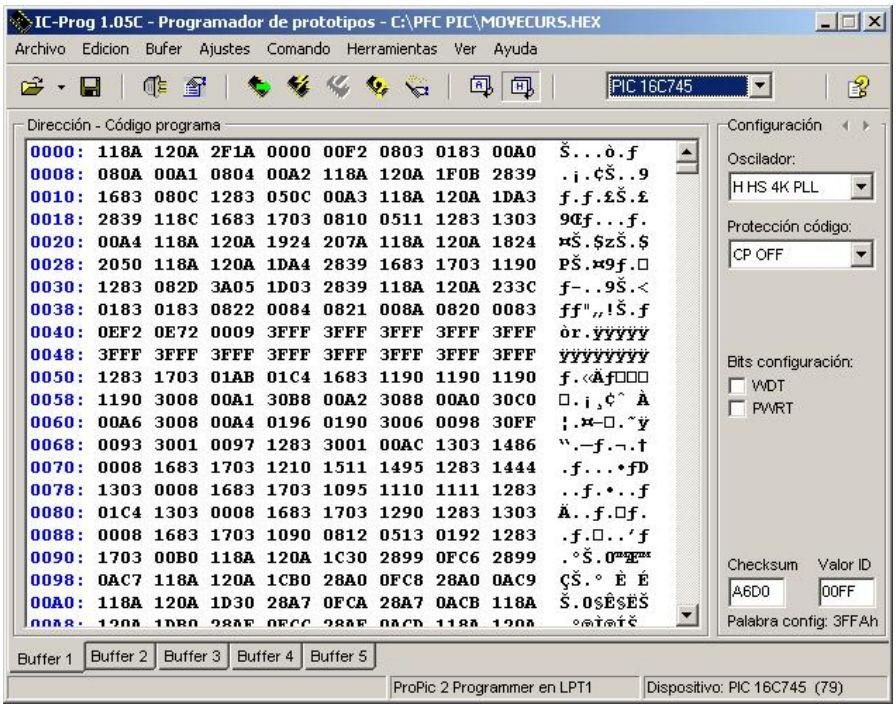


Figura 4-6 : IC-Prog

4.5 Borrar el programa

El PIC debe estar vacío antes de cargar un código nuevo. Si el microcontrolador no está en blanco, debemos borrarlo antes de proceder a una nueva programación.

El 16C745, sólo podrá ser borrado cuando el encapsulado sea CERDIP (tenga la ventana), ya que la memoria de este microcontrolador es EPROM, y por tanto, sólo puede ser borrada mediante radiación ultravioleta.

El proceso de borrado es bastante simple, basta con exponer el PIC a radiación ultravioleta (mediante un borrador de EPROMs), durante al menos 10 minutos. Tras este tiempo, el microcontrolador estará listo para someterlo a una nueva programación.

Figura 4-7 : Borrador de EPROMs

La figura 4-7 muestra la fotografía de uno de estos borradores; en ella se aprecian los elementos más importantes:

- Interruptor de encendido (izquierda)
- Selector de tiempo de exposición (derecha)
- Tubo de UV (centro)

Por motivos de seguridad, el aparato cuenta con una tapa que protege a los usuarios de la radiación ultravioleta, de manera que el tubo queda oculto al cerrarla.

5. Conceptos generales de Motores Paso a Paso

Un motor paso a paso [18][19] es un tipo de motor especial, cuya principal característica es que gira de forma incremental (paso a paso), donde cada paso representa un desplazamiento angular fijo del eje del motor. Es por este motivo que son ideales para la construcción de mecanismos que requieran movimientos muy precisos. La figura 5-1 muestra la fotografía de un motor paso a paso.

Figura 5-1 : Motor paso a paso RS 440-458, de 1.8º de paso

Estos motores están constituidos normalmente por un *rotor* (eje móvil, es la parte que se mueve) sobre el que van aplicados distintos imanes permanentes y por un cierto número de bobinas excitadoras colocadas en su *estator* (parte estática del motor, es decir, lo que permanece quieto). La figura 5-2 muestra un rotor de este tipo de motores.

Figura 5-2 : Rotor de un motor paso a paso

5.1 Funcionamiento

Al alimentar ciertas bobinas del estator (en la forma en que se indicará más adelante), éstas generarán un campo magnético de cierta polaridad, que atraerá el polo correspondiente del imán que se encuentra en el rotor, provocando el movimientote éste hasta la posición en

que los campos magnéticos se alineen (ver figura 5.3). Esta situación se mantendrá mientras circule corriente por las bobinas. De esta manera, hemos movido el motor una posición; para pasar a la siguiente, tendremos que alimentar aquellas bobinas cuya combinación corresponda al siguiente paso.

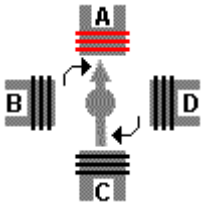


Figura 5-3 : Movimiento del imán (rotor), debido al campo producido por una bobina

Toda la conmutación (o excitación de las bobinas) deber ser manejada externamente por un controlador, según unas secuencias *cíclicas* que se detallan más adelante.

Para que un paso se haga efectivo, la polarización de las bobinas debe mantenerse durante un periodo mínimo de tiempo (que varía según el motor). Si el tiempo que se mantienen alimentadas las bobinas es inferior al mínimo, se producirán saltos en la secuencia, y por tanto, el motor no se moverá correctamente. En este caso puede ocurrir que el motor:

- No realice ningún movimiento.
- Comience a vibrar, pero sin llegar a girar.
- Gire erráticamente.
- Gire en sentido opuesto.

5.1.1 Motores paso a paso bipolares

Este tipo de motor utiliza bobinados independientes (ver figura 5-4). El control de este motor se lleva a cabo mediante la inversión de la polaridad de cada una de las bobinas, en la secuencia adecuada; para esto es necesario utilizar un puente en "H" o un driver específico para cada bobina. De esta manera se obtiene una tabla de secuencias de la forma:

Paso	A	B	C	D
1	+Vcc	Gnd	+Vcc	Gnd
2	+Vcc	Gnd	Gnd	+Vcc
3	Gnd	+Vcc	Gnd	+Vcc
4	Gnd	+Vcc	+Vcc	Gnd

Tabla 5-1 : Secuencia de un motor bipolar de dos bobinados

Cada inversión de la polaridad provoca el movimiento del eje, es decir, la ejecución de un paso. Dependiendo del orden en el que se ejecute la secuencia, el motor girará en un sentido o en otro.

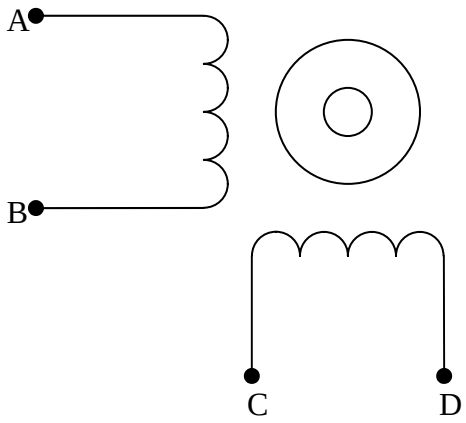


Figura 5-4 : Esquema de un motor paso a paso bipolar de dos bobinados

5.1.2 Motores Unipolares

La principal diferencia respecto a los motores bipolares reside en el hecho de que las bobinas tienen un punto en común (tal yo como muestra la figura 5-5). Los *comunes* de las bobinas pueden estar unidos o no. Esta característica permite desarrollar diferentes combinaciones, utilizando el controlador correcto, para enfatizar ciertas características de los motores, como fuerza o velocidad.

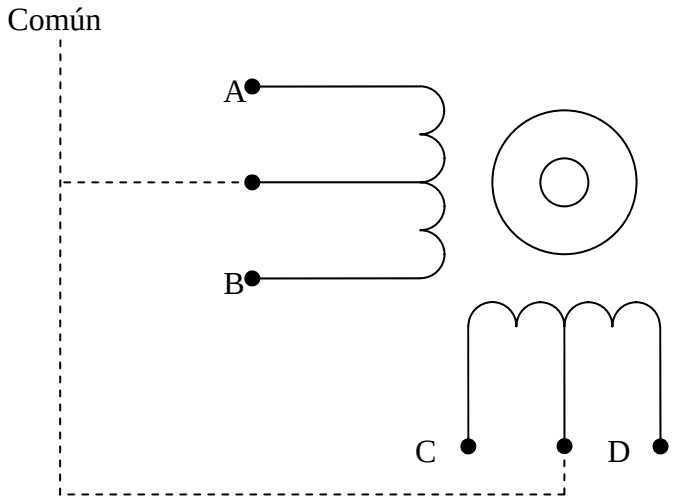


Figura 5-5 : Esquema de un motor unipolar con dos bobinados y un único común

El esquema de conexión se basa en conectar el común a +Vcc o bien a masa, de tal manera que para polarizar una bobina, tan sólo será necesario alimentarla o no. Es decir, si por ejemplo el común se conecta a +Vcc, para que circule corriente por la bobina (y por tanto

genere un campo electromagnético) tendremos que conectar el terminal de la bobina a tierra, generando así una diferencia de potencial. Obviamente, si conectamos un terminal a +Vcc y otro a masa, lo que obtenemos es un corto circuito; es por ello que ésta operación requiere del uso de un controlador que limite y administre la corriente suministrada a la bobina. En cualquier caso, el controlador es necesario para realizar la conmutación; por tanto, se trata de un elemento imprescindible.

Para controlar este tipo de motores existen tres secuencias de encendido de bobinas:

- **Normal**

  Las bobinas son activadas de dos en dos, lo que produce un campo magnético potente, atrayendo con bastante fuerza el rotor del motor; es por ello que con este método los pasos son bruscos. Debido a que siempre hay al menos dos bobinas activas, se obtiene un alto *torque* de paso y de retención.

PASO	A	B	C	D
1	**ON**	**ON**	OFF	OFF
2	OFF	**ON**	**ON**	OFF
3	OFF	OFF	**ON**	**ON**
4	**ON**	OFF	OFF	**ON**

Tabla 5-2 : Secuencia normal de un motor unipolar de dos bobinados

- **Simple**

  Con esta secuencia no se obtiene mucha fuerza, ya que en cada punto de la misma sólo se activa una bobina, que tendrá que realizar en solitario el esfuerzo de arrastrar y sujetar el rotor del motor. Como mayor ventaja cuenta con la sencillez de implementación.

PASO	A	B	C	D
1	**ON**	OFF	OFF	OFF
2	OFF	**ON**	OFF	OFF
3	OFF	OFF	**ON**	OFF
4	OFF	OFF	OFF	**ON**

Tabla 5-3 : Secuencia simple de un motor unipolar de dos bobinados

- **Medio paso**

Se trata de una combinación de las dos secuencias anteriores. Con este método es posible mover el motor en pasos más pequeños y precisos, en concreto, la mitad de un paso, produciendo un movimiento más suave.

La longitud de la secuencia es el doble que en la *simple* o la *normal*, ya que se basa en intercalar un estado de cada tipo:

PASO	A	B	C	D
1	**ON**	OFF	OFF	OFF
2	**ON**	**ON**	OFF	OFF
3	OFF	**ON**	OFF	OFF
4	OFF	**ON**	**ON**	OFF
5	OFF	OFF	**ON**	OFF
6	OFF	OFF	**ON**	**ON**
7	OFF	OFF	OFF	**ON**
8	**ON**	OFF	OFF	**ON**

Tabla 5-4 : Secuencia de Medio-paso de un motor unipolar de dos bobinados

5.2 Identificación de los terminales

Este proceso se hace necesario cuando no se cuenta con las hojas de características de los motores. La identificación de los terminales de estos motores es muy simple, y se basa en el método de ensayo-error.

5.2.1 Motores Unipolares

El cable correspondiente al común será aquel que presente una resistencia menor (la mitad que entre dos terminales) con respecto a los demás cables; esto se debe a que entre un terminal y el común sólo hay una bobina, mientras que entre dos terminales hay dos bobinas.

Para identificar el orden de los terminales en motores de dos bobinados (los más simples y comunes), se coge uno de los cables, que será tomado como referencia (*A*), y se polariza (previamente conectado el común). A partir de esta referencia, se probará a polarizar los cables restantes, de tal forma que aquel que produzca un movimiento en sentido horario será el D, y antihorario el B; por eliminación, el que quede será el *C*. Si el motor cuenta con más bobinados, será necesario acudir a las especificaciones, o a pruebas de secuencias.

5.2.2 Motores Bipolares

La identificación en estos motores es más sencilla. Basta con medir, utilizando un tester, la continuidad (realmente una baja resistencia) entre pares de cables, determinando de esta forma cuáles forman parte de la misma bobina. Para determinar la polaridad, la solución se encuentra en probar, de modo que si de una forma no funciona, bastará con invertir el orden de los cables.

6. Descripción del proyecto

El propósito de este capítulo es aclarar cuáles son los objetivos, las necesidades y requerimientos de la aplicación a realizar.

Como ya se comentó, el objetivo de este proyecto es el de construir una herramienta *genérica* capaz de orientar objetos, por medio de motores paso a paso, de forma remota. Dado el gran avance en la implantación del sistema USB, en detrimento de otros puertos de comunicaciones como el serie (hoy en día muchos ordenadores ya no lo incorporan), la comunicación entre el servidor de la aplicación gestora, y el sistema que controla los motores, se realizará a través de USB, con el objetivo de asegurar la mayor vigencia posible a la herramienta desarrollada.

Se pretende que la gestión de los motores (objetivos del proyecto) sea tal que tan sólo un usuario a la vez pueda actuar sobre ellos, mientras que todos los demás podrán conocer la dirección a la que apunta cada motor en todo momento. Se posibilita además que el control sobre los motores sea por tiempo limitado, o bien indefinido; de esta manera queda abierto el abanico de posibles usos de esta herramienta, dado su carácter *"de uso general"*.

El acceso a la herramienta requerirá del uso de algún tipo de identificación, con el objetivo de que sólo puedan hacer uso de la misma usuarios conocidos. Para controlar quién puede acceder o no a los motores, se establece una jerarquía de permisos: *Administrador* (acceso al movimiento y a la configuración), *Actuador* (se le permite mover los motores) y *Espectador* (sólo tiene permitido ver el estado en el que se encuentran los motores). Es por esta razón que la herramienta debe incorporar un gestor de usuarios que asegure su identidad y acceso a los servicios para los que tiene permiso.

6.1 Hardware y Software del prototipo

La parte física de la herramienta desarrollada la compone un circuito controlador, la conexión (cable) USB con el sistema principal y por supuesto, los motores paso a paso.

6.1.1 Hardware del dispositivo USB

El circuito encargado de ejecutar las órdenes del sistema principal sobre los motores es el mostrado en la figura 6-1.

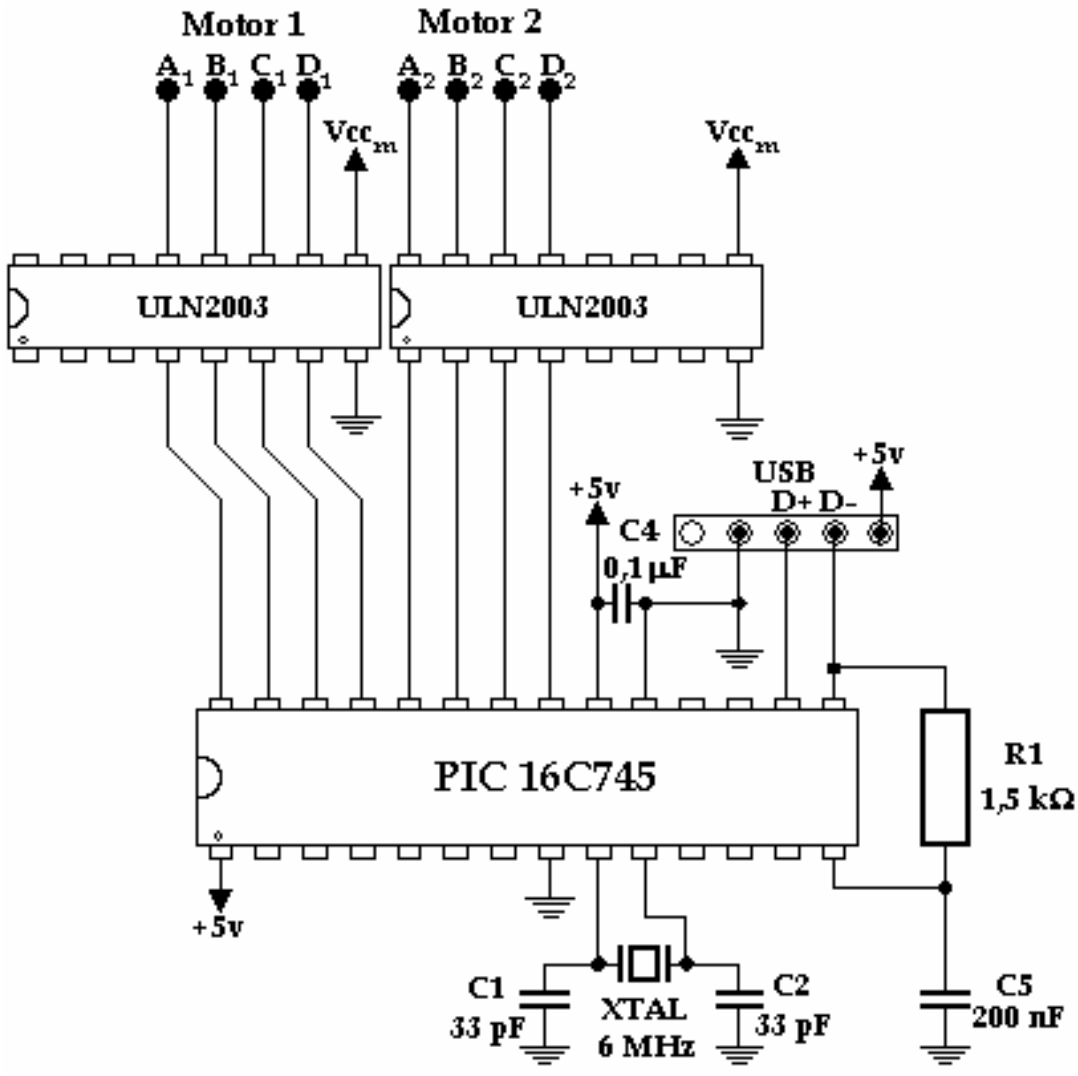


Figura 6-1 : Esquema del circuito controlador

El circuito integrado de mayor tamaño es el microcontrolador PIC 16C745 (tratado en el tema 4). Su misión es la de obtener las peticiones de movimiento generadas por el sistema principal, procesarlas y crear la secuencia de pasos apropiada para los motores. Una vez terminado el mismo, informará al sistema principal de este hecho.

Como todo sistema electrónico digital, el PIC requiere de un sistema de sincronismo; por este motivo, incluye un circuito oscilador interno, que tan sólo requiere una fuente externa de pulsos constantes. Para esta herramienta, y dadas las exigencias de velocidad requeridas por USB, estos pulsos serán dados a través de un cristal de cuarzo. Los cristales de cuarzo, debidamente conectados, proporcionan una señal tal, que sus pasos por cero se producen a una frecuencia constante, con gran fiabilidad y exactitud. El cristal empleado es de 6 MHz, lo cual nos da un ciclo de instrucción en el microcontrolador de 666,67 ns (4 / 6 MHz). Con el objetivo de mantener un buen rendimiento en el circuito oscilador, el fabricante

recomienda conectar un condensador de 33 pF entre cada uno de los terminales del cristal y tierra (C1 y C2).

La comunicación con el sistema principal se hace a través de las líneas D+ y D- del cable USB (en ambos sentidos), las cuales están conectadas a las patillas correspondientes en el microcontrolador.

Tal y como se indicó en el capítulo dedicado a USB, para que el dispositivo se identifique como *Low-Speed*, es necesario conectar una resistencia *pull-up* (a +Vcc) de 1,5 kΩ a la línea D- del cable USB. La tensión necesaria (3,3v) para que se lleve a cabo la detección, la aporta el pin 14 (V_{USB}), que además requiere de un condensador de 200 nF (C5) para asegurar la estabilidad de dicha tensión.

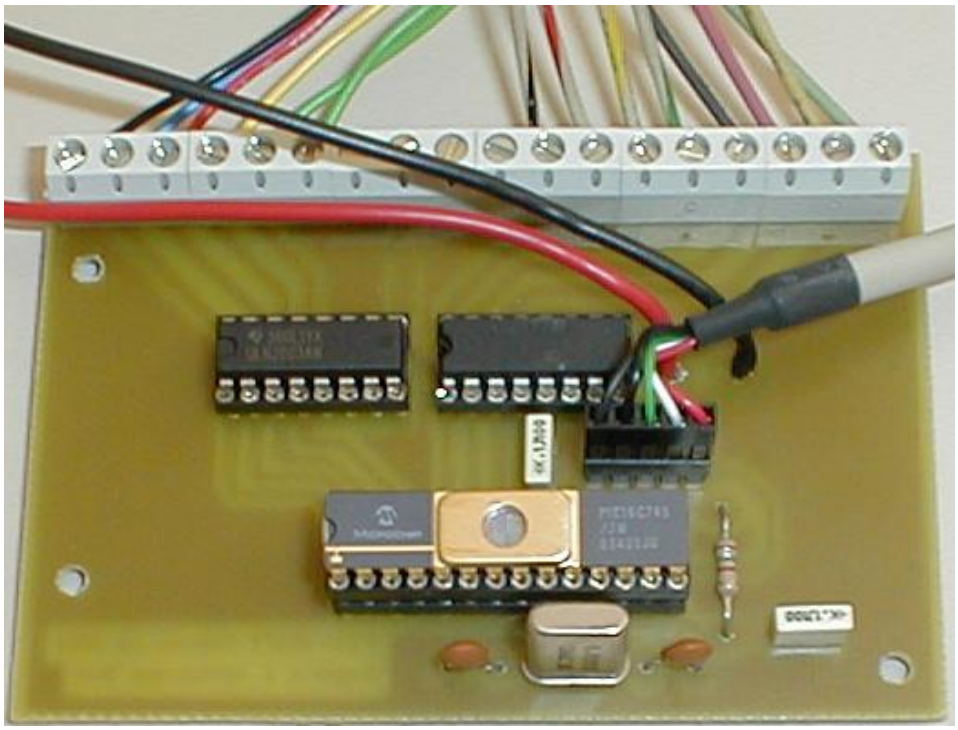


Figura 6-2 : Fotografía del circuito, en placa impresa

Los otros dos integrados presentes en el circuito son ULN2003. Estos dispositivos son utilizados para mantener la corriente necesaria en las bobinas de los motores (hasta 500 mA) a la tensión necesaria (Vcc_m), la cual, en este caso, es distinta de la suministrada por el microcontrolador (que trabaja con 5v). Con su uso, se evita que el PIC tenga que hacer frente al gran consumo que podrían ocasionar los motores, relegando dicha tarea a una alimentación externa (no dependiente de la potencia cedida por el cable USB), limitándose el PIC a indicar cuándo debe ser alimentada cada línea; además, el ULN2003 cuenta con

diodos de supresión que evitan el paso de las posibles corrientes inversas generadas por las bobinas de los motores, que podrían dañar el microcontrolador.

Utilizar una fuente de alimentación externa para alimentar los motores se debe básicamente a dos motivos; por un lado se encuentra el hecho de que así el consumo del dispositivo, desde el punto de vista USB, es mucho menor (utiliza como máximo 100 mA), haciendo posible el uso de esta herramienta en prácticamente cualquier entorno, ya que esta corriente es la mínima que un Hub está obligado a dar (véase capítulo de conceptos generales de USB). Por otro lado, la alimentación externa es casi un requisito para esta herramienta, dado su carácter *genérico*, ya que de esta forma es posible utilizar distintos pares de motores, sin importar la tensión de alimentación (basta con seleccionar la fuente apropiada).

La conexión de los motores debe realizarse según quedó indicado en el capítulo de *conceptos generales de los motores paso a paso*; es decir, los cables terminales deben conectarse al ULN2003 (hay uno de estos integrados por cada motor), según la secuencia establecida (A, B, C, D), que debe ser localizada entre los cables de cada motor. Por otro lado, el/los comunes, a Vcc_m (que en este caso será de +12v).

6.1.2 Software del PIC 16C745

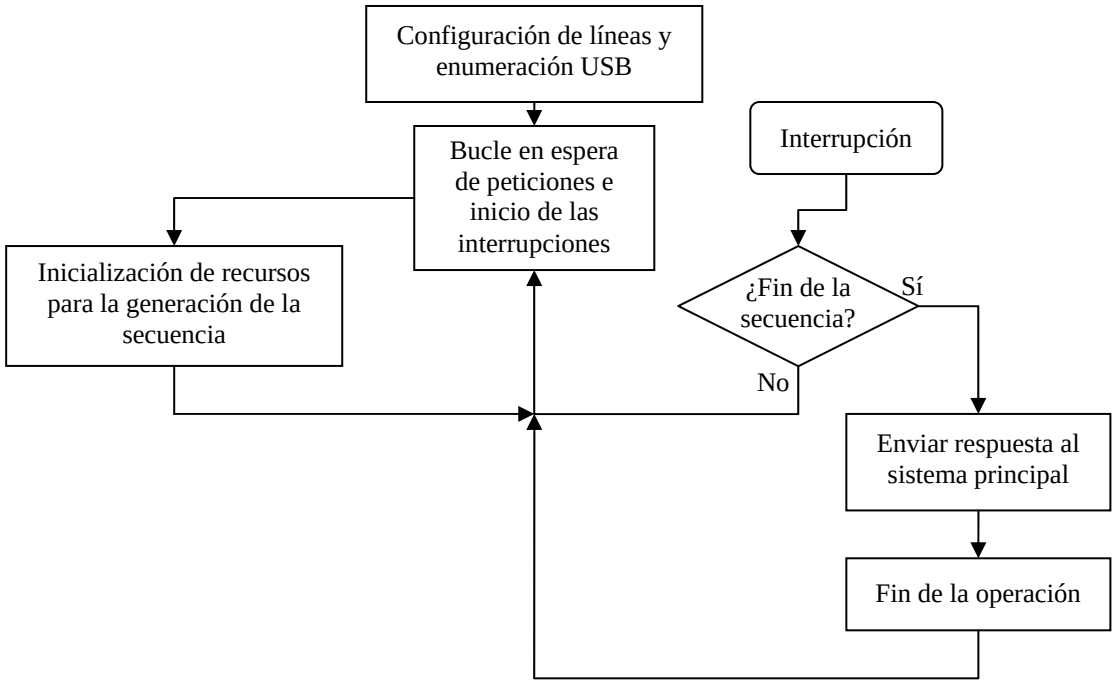


Figura 6-3 : Esquema general del programa ejecutado en el PIC

El programa que controla el funcionamiento del microcontrolador se encarga de la identificación del dispositivo USB ante el sistema principal que lo gobierna, de atender sus peticiones y de avisar al programa controlador de la herramienta cuando la ejecución de las mismas haya terminado.

Las peticiones realizadas por el sistema principal siempre irán enfocadas al movimiento de los motores que gobierna el dispositivo USB; es por ello que la ejecución de las peticiones supone la generación de las secuencias *paso a paso* en el orden necesario para mover el motor en el sentido indicado, de la longitud precisa para mover tantos pasos como se solicite y espaciada en el tiempo lo suficiente como para que el movimiento se realice a la velocidad oportuna. La figura 6-3 muestra un esquema del funcionamiento general del programa que ejecuta el microcontrolador.

6.1.2.1 Firmware USB

Microchip facilita de forma gratuita el código de funciones escritas en ensamblador (o en C), que facilitan en gran medida la labor del programador en las operaciones elementales con USB. El soporte que otorgan cubre las necesidades básicas y generales; con el objetivo de extender sus posibilidades y ajustarse de esta forma a los deseos del programador, el *firmware* cuenta con cierta documentación [17], en la que además de explicar su uso, hace referencia a las posibles modificaciones que pueden realizarse y la utilidad que éstas pueden tener.

El paquete de código ensamblador (empleado en este proyecto), cuenta con los siguientes archivos (en la versión 2.00):

- **movecurs.pjt :** Archivo de proyecto para el ensamblador MPASM, del programa de ejemplo incluido en el *firmware.*
- **16C745.lkr :** Archivo de *script* de *lincado*. Se utiliza para definir las diferentes regiones de memoria, y su uso (bancos de registros generales, especiales,...).
- **usb_defs.inc :** Define las variables y macros utilizadas por el *firmware*.
- **usb_main.asm :** Código del programa de ejemplo incluido en el *firmware*; supone el primer contacto para el programador inexperto con el interfaz USB del PIC. El

ejemplo se llama *"Cursor in a circle"* y hace básicamente eso: al conectarse a un sistema, mueve el cursor del ratón describiendo círculos.

- **usb_ch9.asm :** Implementa las funciones indicadas en la especificación de USB 1.1 [11]. Más adelante se hará referencia a las más importantes.

- **hidclass.asm :** Implementa las funciones indicadas en la especificación HID [13]. El uso de esta clase de dispositivos facilita en gran medida la programación de las aplicaciones en el sistema principal, ya que en la mayoría de los casos no es necesario construir unos controladores específicos (se utilizan los de la clase). Su uso es por tanto opcional, condicionado por las necesidades del proyecto a llevar a cabo (en este en concreto, sí se utiliza).

- **descript.asm :** Contiene el descriptor del dispositivo. En este caso implementa el descriptor del ejemplo. En cada proyecto que se lleve a cabo, si el tipo y/o los recursos requeridos cambian, el programador deberá implementar un nuevo archivo de estas características, ya que es utilizado por el dispositivo para identificarse ante el sistema principal (ver capítulo de *conceptos generales de USB*).

6.1.2.1.1 Funciones Básicas

Estas funciones se ocupan de la inicialización la comunicación USB y del intercambio de datos con el bus.

- **InitUSB :** Esta función debe ser invocada en la rutina principal, una vez alimentado el dispositivo, tras un retardo de 16μs; esta condición da tiempo al periférico USB para reiniciarse (ver capítulo de *conceptos generales de USB*). Esta función se encarga de realizar las operaciones oportunas para pedir al sistema principal que proceda a identificar y configurar el dispositivo, a la vez que habilita las interrupciones USB para su uso posterior.

- **ConfiguredUSB :** Esta macro debe ser invocada tras la llamada a InitUSB, ya que determinará si el dispositivo está configurado o no, y por tanto, si es posible la utilización de los registros *endpoint*. En realidad, bloquea la ejecución del programa (se mantiene en bucle infinito) hasta que el dispositivo sea configurado.

- **DeinitUSB :** Provoca al sistema principal a ignorar el dispositivo; es decir, el dispositivo se desconecta a sí mismo del BUS.

- **ServiceUSB :** Esta function debe ser invocada en el bucle de espera de peticiones, ya que es la encargada de procesar las transacciones USB, y actuar en consecuencia.

- **PutEP*n* :** Envía datos al sistema principal, a través del *endpoint "n"*. El número de bytes a transmitir se indica en el registro *W* (acumulador). La localización del bloque de datos a enviar debe encontrarse en FSR (registro de direccionamiento indirecto) y en los bits *IRP* (utilizados en direccionamiento indirecto, indican a qué banco de registros se hace referencia). Si el valor devuelto por el flan de acarreo tras la ejecución de la función es "0", indicará que ha ocurrido un error al enviar los datos.

- **GetEP*n* :** Recibe datos del sistema principal, a través del *endpoint "n"*. Al igual que ocurría en *PutEPn*, la localización del bloque de datos en que se almacenarán los datos entrantes debe encontrarse en FSR y los bits IRP. El flag de acarreo será el encargado de indicar si los datos han sido obtenidos correctamente (devolviendo un "1").

6.1.2.1.2 Recursos Utilizados

El firmware requiere ciertas cantidades de recursos del microcontrolador para llevar a cabo sus funciones. Es de vital importancia conocer estos consumos, ya que condicionarán el diseño del código propio de la aplicación, pues éste debe conformarse con los recursos sobrantes. En cualquier caso, el firmware puede modificarse para obtener mejores resultados en cuanto a la utilización de recursos. El código implementado en el paquete estándar, en su versión ensamblador, requiere:

- **Niveles de Pila :** El firmware requiere de dos niveles de pila. Dado que el PIC 16C745 cuenta con 8 niveles, el programador tan sólo podrá utilizar los seis restantes.

- **ROM :** La memoria de programa requerida por el firmware para implementar sus funciones es aproximadamente de 1,5 kB, sin contar el descriptor (cuya longitud es

variable). Por lo general, las tablas del descriptor y los descriptores de cadena ocupan cerca de 256 bytes adicionales.

- **RAM :** El firmware requiere de 40 bytes de memoria del banco 2; de esta manera quedan libres los bancos 0 y 1, a la vez que la segunda mitad del banco 2 para datos de la aplicación desarrollada.

6.1.2.2 Código del microcontrolador

A parte del archivo principal y el del descriptor de dispositivo, el código requiere la inclusión de los archivos del firmware: *16C745.lkr, usb_defs.inc, usb_ch9.asm* y *hidclass.asm.*

Con objeto de permitir una sencilla ampliación del programa, éste basa su funcionamiento en el vector de interrupciones. De esta manera, es posible gestionar varias acciones al mismo tiempo; por ejemplo, el uso de temporizadores para controlar el tiempo, en vez de bucles contadores, abre la posibilidad de realizar otras acciones mientras un periférico realiza la cuenta, adquiriendo éste el control del procesador sólo cuando haya transcurrido el periodo de tiempo establecido. Por esta razón, el programa principal es sencillamente un bucle de espera de peticiones; cuando se recibe una, se inicializan las variables contadoras de pasos y los valores de los timers (en *ProcessOUTBuffer*) , para a continuación volver al bucle y esperar a que la operación finalice, con el fin de enviar una respuesta al sistema principal.

```
LoopForData
  pagesel     GetEP1
  banksel outbuffer
  bankisel outbuffer
  movlw outbuffer
  movwf FSR
  movlw 0x8
  call  GetEP1                  ;intentamos obtener datos del PC

  pagesel     ProcessOUTBuffer
  btfsc STATUS,C                ;SI llegó un paquete
  call  ProcessOUTBuffer        ;procesamos el paquete
```

```
compruebaEnviar                    ;comprueba si terminó el
proceso pagesel LoopForData
  BANKSEL flag_motores
  btfss flag_motores,6             ;¿ha terminado el primer motor?
  goto LoopForData

  btfss flag_motores,2             ;¿ha terminado el segundo motor?
  goto LoopForData                 NO ha terminado
Pkt2PC                             ;SI han terminado
  pagesel     PutEP1
  bankisel outbuffer
  banksel outbuffer
  movlw outbuffer
  movwf FSR                        ;localización del buffer de datos
  movlw 0x8                        ;8 bytes
  call  PutEP1                     ;envía los datos al sistema principal

  pagesel LoopForData
  BANKSEL flag_motores    ;desactivamos la opción de volver a enviar
  bcf   flag_motores,2
  goto LoopForData        ;volvemos al bucle
```

En este fragmento se emplea la variable *flag_motores*, cuyo cometido es el de indicar los estados en que se encuentra la secuencia de cada motor (parte alta o parte baja), que son:

- **Fin de la secuencia**
- **Debe darse un paso o un medio-paso**
- **Situación de reposo en el puerto :** Una vez finalizada la secuencia, se espera el tiempo equivalente a un paso, para poner a cero la parte correspondiente del puerto; de esta manera se consigue el reposo del motor.

6.1.2.2.1 Generación de la secuencia para los motores paso a paso

En este proyecto, se ha decidido utilizar la técnica denominada *"medio-paso"*, que aunque resulte algo más larga y compleja de implementar que las otras dos opciones estipuladas (ver capítulo de *conceptos generales de motores paso a paso*), tiene como ventaja la posibilidad de realizar movimientos más suaves.

Paso	A	B	C	D
1	ON	OFF	OFF	OFF
2	ON	OFF	OFF	ON
3	OFF	OFF	OFF	ON
4	OFF	ON	OFF	ON
5	OFF	ON	OFF	OFF
6	OFF	ON	ON	OFF
7	OFF	OFF	ON	OFF
8	ON	OFF	ON	OFF

Figura 6-4 : Secuencia de Medio-paso de un motor unipolar de dos bobinados

Una de las posibles formas de implementar una secuencia de las características necesarias, podría ser la de almacenar en un vector cada una de las palabras que forman la secuencia, que atendiendo a la representada en la figura 6-4 serían (en hexadecimal): *8, 9, 1, 5, 4, 6, 2, A*. Pero no se trata de un método poco eficiente, a la par que costoso en un recurso tan preciado como la memoria de datos. Por otro lado, atendiendo a la estructura de la secuencia, se ha observado un comportamiento sistemático en la misma, es decir, sigue una serie de operaciones lógicas para su obtención, siguiendo ciertas reglas:

- Cada punto de la secuencia se genera a partir de una operación lógica entre la palabra correspondiente a la última posición *par* calculada de la tabla y una variable auxiliar. Es importante recordar que se hace referencia a la tabla representada en la figura 6-4, ya que debido a la característica cíclica de estas secuencias, esta tabla no representa la única secuencia válida, ya que el punto de partida podría ser cualquiera de los pasos en ella representados, o incluso, es posible generar otras secuencias que cumplan las premisas de los motores paso a paso. En cualquier caso, como ejemplo, se trata de una secuencia válida.

- Esta variable alterna su valor entre 0x03 y 0xC0 tras la generación de una palabra que ocupe una posición impar en la tabla. Dependiendo de si la secuencia comienza con esta variable a 0x03 ó 0xC0, se originará el movimiento en un sentido o en otro.

- Para calcular las posiciones impares de la tabla, la operación llevada a cabo es una AND. Estas posiciones coinciden con la secuencia de paso *simple*.

- Para calcular las posiciones pares, la operación realizada es una X-OR. Estas posiciones coinciden con la secuencia de paso *normal*.

Veamos un ejemplo para aclarar las ideas:

El paso 3 está formado por la palabra 0001, si la variable auxiliar cuenta en ese momento con el valor 0x03, para calcular el paso 4 tendremos que convertir esa variable en 0x0C y aplicar un X-OR con la última palabra *par* calculada, que en nuestro caso será la 2; por tanto:

$$1001_{\text{paso 2}} \text{ X-OR } 1100_{\text{variable}} = 0101_{\text{paso 4}}$$

Tras estas premisas, el código resultante es muy sencillo; para calcular las etapas de secuencia simple:

```
BANKSEL giro1
movf giro1+0,w              ;variable que almacena el paso
andwf giro1+1,w             ;variable auxiliar

BANKSEL giro1
movlw 0xF0                  ;invertimos el sentido
xorwf giro1+1,1             ;lo almacenamos
```

Y finalmente, para calcular las de secuencia normal:

```
BANKSEL giro1
movf giro1+0,w              ;variable que almacena el paso
xorwf giro1+1,w             ;variable auxiliar
movwf giro1+0               ;almacenamos el valor del paso
```

6.1.2.2.2 Tratamiento de la petición

Esta operación se realiza en la función *ProcessOutBuffer*, en la cual se asignan los recursos necesarios para el correcto movimiento de cada motor, en relación a los datos incluidos en el paquete entrante. Estos datos se almacenan en un buffer; la función de cada una de sus posiciones se detalla a continuación:

buffer+0 : Indica el sentido de giro (Motor 1/Motor 2)
buffer+1 : velocidad del motor 1
buffer+2 : Parte alta del número de pasos del motor 1
buffer+3 : Parte baja del número de pasos del motor 1
buffer+4 : velocidad del motor 2

buffer+5 : Parte alta del número de pasos del motor 2

buffer+6 : Parte baja del número de pasos del motor 2

De esta manera, se asigna un byte para determinar la velocidad de movimiento (1-255), y dos para el número de pasos (1-65536), ya que 255 no es un número suficiente como para dar una vuelta entera en los motores de gran precisión, y esta herramienta pretende ser lo más genérica posible.

En primer lugar se inicializa la variable contadora de pasos (*Npasos_motorN*) y se determina el sentido del giro (que fijará el orden en que la secuencia será desarrollada); para ello, según se ha explicado, se asignará a la variable auxiliar el valor 0x03 ó 0x0C

```
banksel outbuffer       ;variable en que se almacenan los datos
                        ;recibidos
movf outbuffer+2,w      ;parte alta de los pasos del motor 1
banksel Npasos_motor1
movwf Npasos_motor1+0   ;variable contadora de pasos
banksel outbuffer
movf outbuffer+3,w      ;parte baja de los pasos del motor 1
banksel Npasos_motor1
movwf Npasos_motor1+1   ;variable contadora de pasos
...
movlw 0xF0              ;si el sentido es contrario
BANKSEL giro1           ;al del movimiento anterior, invertimos
xorwf giro1+1,1         ;el valor de la variable auxiliar
```

Si el número de pasos a mover es distinto de cero, se inicializa el Timer correspondiente a cada motor (el Timer0 es el asignado al motor 1, mientras que el Timer1 es para el motor 2). Se aprovecha la posibilidad de contar con dos temporizadores para permitir el movimiento simultáneo de ambos motores.

El tiempo más largo que se puede manejar con el timer0 (el menor de los dos), utilizando un reloj de 6MHz, es de 43,52 ms; este tiempo es demasiado pequeño para muchos motores, por lo que se ha decidido añadir al código una variable contadora, que sólo permita dar un paso cuando haya sucedido un cierto número de interrupciones del timer.

```
movlw 0x40
movwf contador1         ;número de int TMR0 necesarias
...
```

```
BANKSEL outbuffer
movf outbuffer+1,w      ;velocidad
BANKSEL     TMR0
movwf TMR0              ;asignamos el valor al timer

BANKSEL     T1CON
movlw b'00110101'       ;prescaler 256
movwf T1CON
```

Con el objetivo de mantener la misma resolución en ambos temporizadores, en el Timer1, dado que consta de dos bytes, se asigna el valor fijo 0xFF al byte bajo, mientras que al alto, el valor señalado en el paquete de datos. Igualmente, la variable contadora de interrupciones se adapta al timer0, reduciendo su valor.

```
BANKSEL contador2
movlw 0x08
movwf contador2             ;número de int TMR1 necesarias
…
banksel outbuffer
movf outbuffer+4,w          ;velocidad
BANKSEL     TMR1H
movwf TMR1H                 ;byte alto
BANKSEL     TMR1L
movlw 0xFF
movwf TMR1L                 ;byte bajo

BANKSEL     T1CON
movlw b'00110101'           ;prescaler
movwf T1CON
```

6.1.2.2.3 Descriptor del dispositivo

El descriptor de dispositivo incluye información relativa al dispositivo USB, en una tabla de descriptores de distintas características. El código de este archivo se limita a devolver los valores de la tabla, según el índice indicado por el sistema principal; de esta manera, el sistema principal recopila la información necesaria para configurar el dispositivo.

Por ejemplo, la implementación de la tabla del endpoint1 sería:

```
Endpoint1
```

```
      retlw 0x07        ; longitud del descriptor
      retlw ENDPOINT
      retlw 0x81        ; EP1, Entrada
      retlw 0x03        ; modo Interrupcion
      retlw 0x08        ; tamaño máximo del paquete (8 bytes), byte bajo
      retlw 0x00        ; tamaño máximo del paquete (8 bytes), byte alto
      retlw 0x0A        ; intervalo de polling (10ms)
```

Además, también se incluye información en forma de cadenas de texto, que serán tratadas de la misma forma:

```
String1_l1              ;identificador textual del nombre de dispositivo
      retlw String2_l1-String1_l1   ; longitud de la cadena
      retlw 0x03  ; descriptor de cadena tipo 3 (definido antes)
      retlw 'P'
      retlw 0x00
      retlw '.'
      retlw 0x00
      retlw 'F'
      retlw 0x00
      retlw '.'
      retlw 0x00
      retlw 'C'
      retlw 0x00
      retlw '.'
      retlw 0x00
      retlw ' '
      retlw 0x00
      retlw 'U'
      retlw 0x00
      retlw 'S'
      retlw 0x00
      retlw 'B'
      retlw 0x00
String2_l1
```

En este caso, por ejemplo, se almacena la información textual del nombre del dispositivo: *“P.F.C. USB”*.

6.2 Análisis Funcional del Software

La herramienta presentada en este proyecto sigue el modelo clásico Cliente-Servidor, el cual se basa en la existencia de una aplicación servidora encargada de proporcionar servicios a varios clientes diferentes.

Para mostrar con mayor claridad el funcionamiento y capacidades de esta herramienta, pasamos a mostrar una visión de cómo trabajan ambas partes conjuntamente, además de analizar la forma de trabajo del servidor y del cliente de forma separada. Asimismo veremos las diferentes posibilidades que tendrán asociadas las aplicaciones según el caso.

6.2.1 Visión Modular del Proyecto

La gran mayoría de las funciones que se realizan en cada uno de los componentes que forman parte de la herramienta pueden ser incluidas dentro del funcionamiento de un módulo asociado a cada elemento. Cada módulo está compuesto por una o varias clases ó funciones (según sea el lenguaje en que ha sido implementado), que se encargan de un tipo de tarea específica para cada elemento. El Principal objetivo de mostrar este tipo de visión es el de formar una idea global del funcionamiento de este PFC. Estos módulos pueden verse en la figura 6-5.

La misión de cada módulo es la siguiente:

- **Módulo de gestión de interfaz grafica en el cliente:** Se encarga de la creación de una interfaz gráfica en el cliente, a través de la cual la aplicación pueda interactuar con el usuario, mostrándole información, o requiriendo (si le está permitido), información de movimiento.

- **Módulo de transmisión y recepción en el cliente:** Módulo encargado de establecer y mantener la comunicación con el servidor remoto, transmitiendo peticiones y procesando las respuestas.

- **Módulo de transmisión y recepción en el servidor:** Este módulo tiene como misión la de gestionar y mantener las conexiones con cada uno de los clientes, procesando sus peticiones y enviando respuestas.

- **Módulo de gestión de interfaz grafica en el servidor:** Se encarga de proporcionar una interfaz gráfica de la aplicación al sistema servidor. Su principal cometido es el de interactuar con el usuario administrador para configurar la herramienta.

- **Módulo de gestión de la configuración:** Módulo encargado de almacenar y gestionar los parámetros de configuración, al tiempo que envía la información oportuna a cada uno de los elementos del sistema servidor, para así ajustarlos a las exigencias de la configuración actual. La información almacenada es la referente a las opciones de la red y los parámetros de los motores.

- **Módulo de gestión de usuarios:** Su cometido es el almacenar los datos de los usuarios reconocidos por el sistema y sus permisos; en función de esta información, gestionará los recursos de la herramienta entre los clientes.

- **Módulo de gestión de pasos:** A partir de las peticiones de movimiento, el conocimiento de la posición actual, las características de los motores y las restricciones impuestas al giro, este módulo se ocupa de calcular el número de pasos necesarios para llevar a cabo la orientación de los motores. Suministra además información a cerca de los movimientos, que almacena en un fichero.

- **Módulo de comunicación con el gestor USB:** Encargado de establecer un pipe de comunicaciones entre el proceso servidor y el de gestión del USB. De esta manera, el servidor envía las peticiones de movimiento, y recibe confirmaciones ó errores por parte del gestor.

- **Módulo de comunicación con el servidor:** Se ocupa de crear un pipe de comunicaciones que lo conectará al servidor, a través del que responder a sus peticiones de movimiento.

- **Módulo de gestión de interfaz grafica en el gestor USB:** Proporciona una interfaz gráfica a la aplicación que gestiona el USB, facilitando al usuario información a cerca del dispositivo USB y los movimientos ejecutados (así como su velocidad).

- **Módulo de comunicación con el dispositivo USB:** Proporciona los procedimientos adecuados para acceder al dispositivo USB y enviarle las peticiones de movimiento.

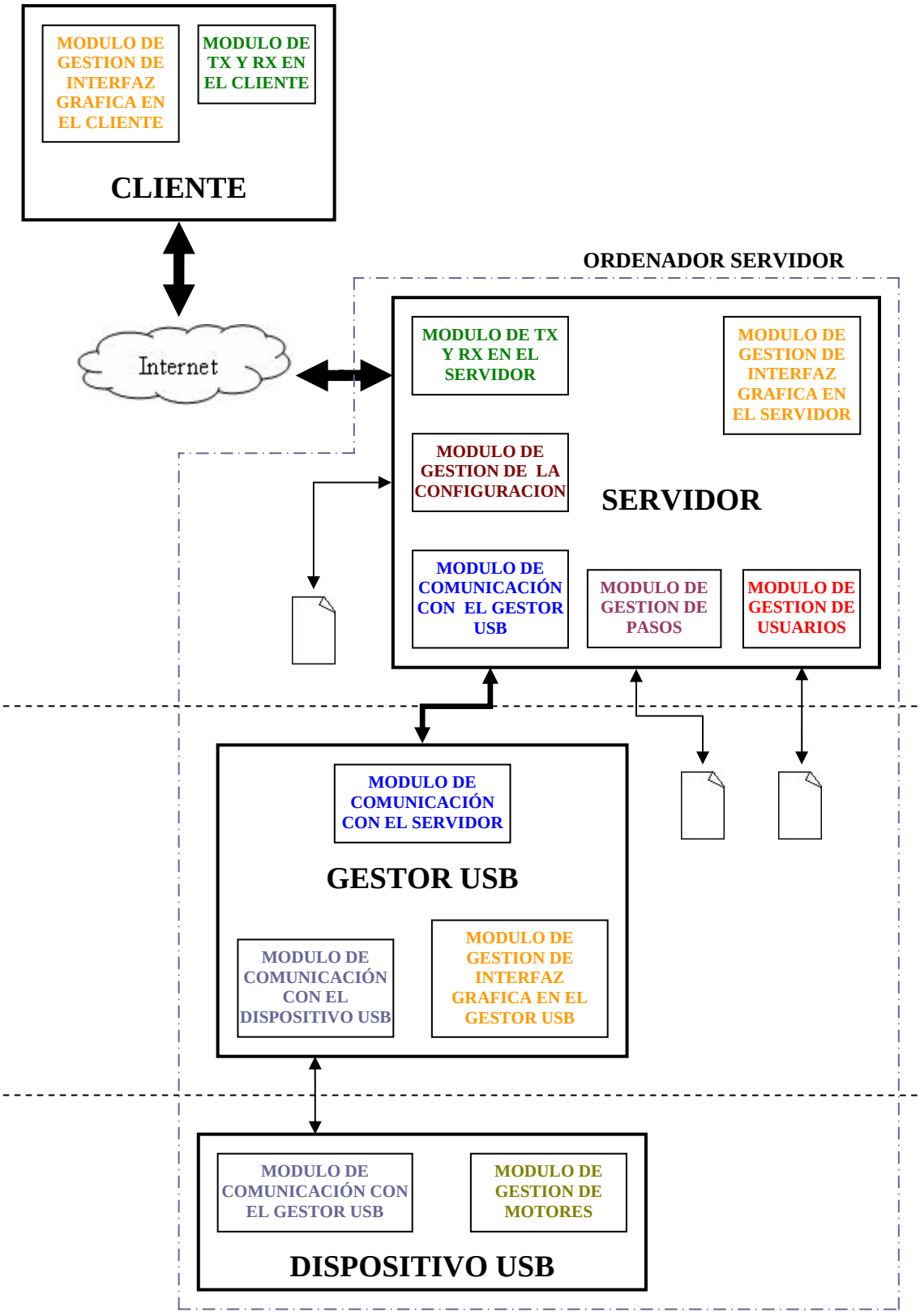


Figura 6-5 : Visión modular del Proyecto

- **Módulo de comunicación con el gestor USB:** Módulo encargado de recibir las peticiones del sistema principal y proporcionar el soporte necesario para responder a las mismas.

- **Módulo de gestión de motores:** Módulo que hace de interfaz con los motores, proporcionándoles la secuencia adecuada (a la velocidad indicada) para que sea posible su movimiento.

6.2.2 Análisis del Servidor

6.2.2.1 Funciones del servidor

El servidor asume un cierto número de facultades administrativas de los recursos existentes, así como la gestión de los clientes. Las principales características que podemos encontrar en el servidor son:

- **Capacidad para alternar movimiento local/remoto:** Es posible indicar si se permite a los clientes mover los motores, o el control se hace en la máquina servidora (los clientes por tanto, sólo pueden ver).

- **Capacidad de habilitar/deshabilitar la gestión de conexiones:** En cualquier momento, un usuario administrador puede detener las transacciones cliente-servidor, con el fin de proceder a cambios en la configuración, o de cerrar el servidor.

- **Capacidad para configurar los parámetros de conexión de los clientes**

    - **Posibilidad de definir el número máximo de usuarios que pueden acceder a la herramienta**

    - **Posibilidad de definir el puerto de gestión:** Con el fin de adaptarse lo mejor posible a cualquier sistema, se posibilita que el administrador defina el puerto por el que los clientes realizarán las tareas de peticiones de conexión e identificación de usuarios.

    - **Posibilidad de definir los tiempos de espera por paquetes:** Este tiempo es de gran utilidad, pues sirve (sobre todo en las conexiones UDP) para determinar

cuándo un usuario se ha ido sin despedirse. Tanto el servidor como el cliente deben respetar este tiempo; es decir, una vez transcurrido el mismo, deben enviar un mensaje preguntando si el otro extremo se encuentra aún en línea, al cual se deberá dar una respuesta (la ausencia de respuesta es una negativa). En el cliente, este tiempo es el doble que en el servidor, para así dar tiempo a recibir la petición de disponibilidad.

- **Posibilidad de definir los tiempos de conexión:** Es posible definir la duración tanto de la conexión inicial (que identifica usuarios y les cede recursos), como a la conexión actuadota (la que obtiene el permiso de movimiento). En ambos casos el objetivos es evitar que un usuario pueda monopolizar un tipo de conexión que no es compartible. En cualquier caso, también se puede indicar que estas conexiones tengan una duración ilimitada.

- **Capacidad para configurar los parámetros de los motores**

  - **Posibilidad de definir el ángulo de paso de cada motor:** A través de este parámetro se indica una de las características principales de los motores, la cual indica la precisión del motor, a la vez que el número de pasos necesarios para completar una vuelta.

  - **Posibilidad de definir el número máximo de vueltas:** Este parámetro puede ser de gran utilidad en aquellos casos en los que hay posibilidad de que se enrollen los cables. También es posible indicar que no haya limitación de vueltas.

  - **Posibilidad de definir un ángulo máximo de giro:** Puede ser de utilidad ante dificultades físicas (ubicación) o mecánicas.

  - **Posibilidad de definir la velocidad máxima:** Este parámetro resulta esencial cuando se trabaja de forma genérica con motores paso a paso, ya que cada uno requiere un determinado tiempo mínimo entre pasos; el incumplimiento de este tiempo puede derivar en comportamientos no deseados del motor (ver capitulo sobre motores paso a paso).

- **Posibilidad de fijar la posición inicial:** Esta característica es de vital importancia, sobre todo como paso previo a la espera de conexiones, ya que de esta manera es posible indicar que la posición actual es la inicial (de cara a lo que verán los clientes).

- **Capacidad para especificar el tamaño máximo del archivo de registro de movimientos**

- **Capacidad para gestionar usuarios**

  - **Posibilidad de añadir/eliminar usuarios:** A través del servidor, se pueden añadir ó eliminar datos de usuarios a los que se le permite el acceso a la herramienta, y sus privilegios.

  - **Posibilidad de modificar los datos de los usuarios**

  - **Posibilidad de desconectar clientes:** Un usuario administrador, a través de la aplicación servidora puede terminar con la conexión de un usuario.

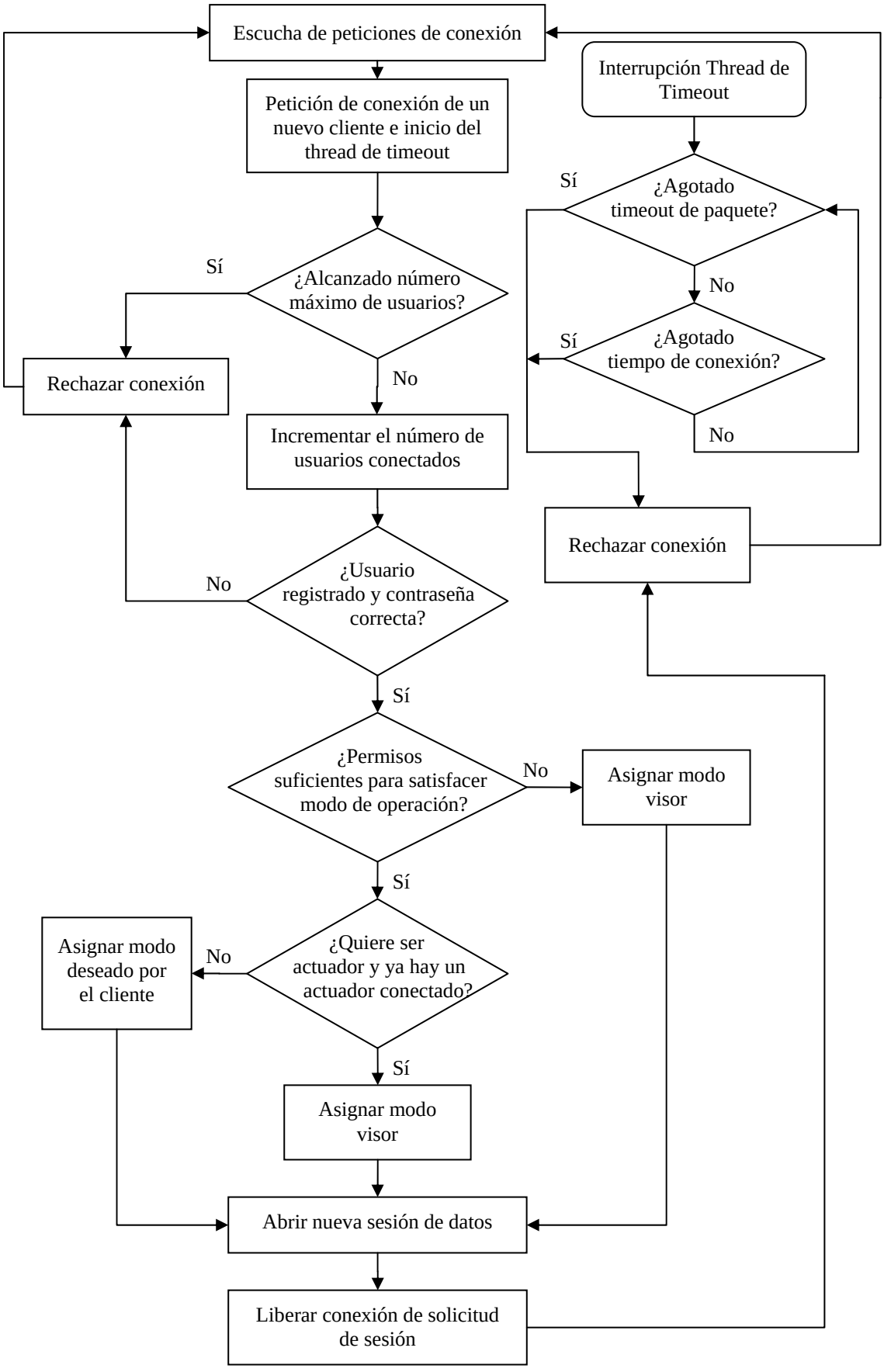


Figura 6-6 : Esquema de los pasos a seguir para la conexión de un usuario

6.2.2.2 Esquema de funcionamiento

Antes de que el servidor comience a atender peticiones de los clientes, hay que arrancarlo e indicarle los valores con los que trabajar. Estos valores van desde el puerto por el que se escucharán las peticiones hasta las características de los motores, pasando por la duración de las sesiones.

Una vez que el servidor se haya conectado a la red, será el elemento que permanezca permanentemente a la escucha de las peticiones realizadas por los clientes. Los pasos seguidos por el servidor para conectar usuarios remotos son los indicados en el esquema de la figura 6-6. Para proceder a la desconexión, se muestran en la figura 6-7.

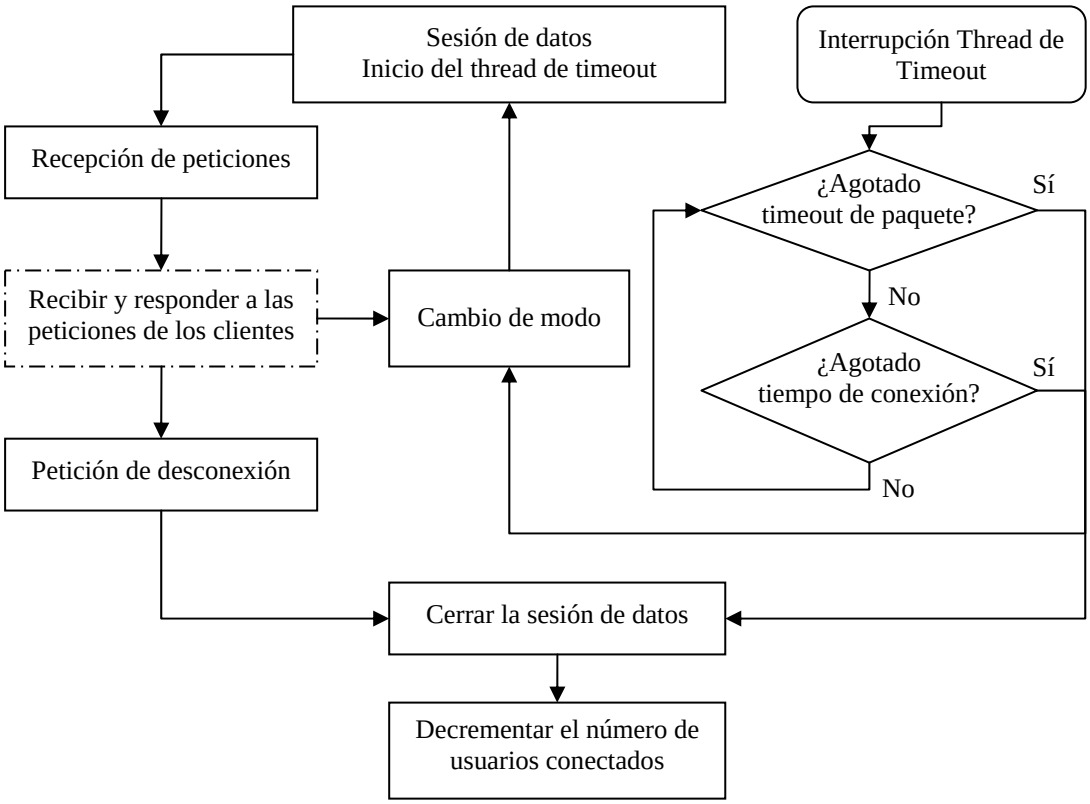


Figura 6-7 : Esquema de los pasos a seguir para la desconexión de un usuario

Para facilitar la comprensión de forma más completa las operaciones realizadas por el servidor cuando un cliente remoto desea establecer una comunicación y su posterior eliminación, veremos de forma más detallada las acciones realizadas:

Una vez iniciado, el servidor de red permanece a la espera de que algún cliente intente establecer una conexión TCP/IP por el puerto indicado en la configuración como *puerto de gestión*.

Inicialmente, cuando un cliente establece una conexión de gestión, si aún no se ha alcanzado el número máximo de conexiones definido en la configuración, el servidor insta al cliente a que le envíe el nombre de usuario, la contraseña, y el modo de acceso al que pretende acceder (Administrador, Actuador o Espectador). Una vez recibidos, se debe comprobar en primer lugar si el usuario es reconocido por la aplicación, y si se corresponde con el que intenta acceder a la misma (mediante la contraseña).

Si el usuario es quien dice ser, se comprueba que tiene permisos suficientes como para acceder al servicio; en caso de no tenerlos, se le asigna el modo visor. Los clientes remotos pueden acceder de dos modos: como *actuadotes* ó como *espectadores*. En el primer caso, siempre y cuando no se haya alcanzado el número máximo de conexiones, siempre se da acceso; esto se debe a que la visión de estado es un recurso compartible, y por el que no se compite, por lo que no necesita de una regulación adicional. Los problemas aparecerían si se permitiera que a la vez estuvieran conectados dos usuarios actuadotes, ya que podría producirse una lucha por el recurso, y un más que probable mal uso; es por ello que sólo se permite un usuario actuador a la vez. Cuando el actuador abandone el recurso, se avisa a todos los espectadores con el fin de que si alguno tiene permiso, pueda obtener su control.

En cualquier caso, una vez que se haya asignado un modo de funcionamiento, el usuario accederá a una sesión de transferencia de datos, abandonando la conexión de gestión, con el fin de que otro usuario puede utilizarla para acceder a la aplicación. Lo mismo ocurre en los casos en que la conexión de gestión es rechazada, con la salvedad de que en estos casos también se envía información sobre el motivo por el que la conexión fue rechazada.

Existen dos tipos de sesiones de datos (así como existen dos modos de acceso a la aplicación):

- **Sesión en modo actuador:**

  Debido a que se trata de un usuario con privilegios, que controla el recurso fundamental de la herramienta, se debe asegurar de alguna manera la integridad de los datos que envía; es por ello que la conexión que utiliza es TCP/IP, ya que se trata de un protocolo orientado a conexión que asegura la entrega de los datos, o la indicación de su pérdida si esta ocurriera, proporcionando de esta manera mecanismos eficaces para asegurar el correcto funcionamiento de la sesión.

- **Sesión en modo espectador:**

  Debido a que normalmente habrá un mayor número de sesiones de este tipo frente a las sesiones en modo actuador (que sólo puede haber una a la vez), y al hecho de que en este caso la integridad de los datos no es tan importante (la pérdida de información puntual no implica un error grave en el funcionamiento de la aplicación), estas conexiones utilizarán el protocolo UDP, que no supone garantías en la entrega, pero que al no estar orientado a conexión, supone un consumo menor de recursos del sistema.

Durante el proceso de gestión, desde que es aceptada la conexión, se llevan a cabo dos tipos de controles en un proceso paralelo:

- **Control de timeout:** Se encarga de evitar largos períodos de tiempo sin recibir información del cliente; para ello, si transcurrido un tiempo establecido no se ha recibido nada, se envía un paquete de reconocimiento, y se espera su respuesta. Si la respuesta no llega durante el intervalo siguiente (el mismo tiempo que se esperó anteriormente), se supone al cliente ausente, y se cancela su conexión. Su objetivo es el de detectar cuándo se ha ido (sin avisar) el cliente.

- **Control de duración de sesión:** Controla que la gestión no dure más de lo especificado en la configuración. De esta forma se evitar que algún usuario monopolice la conexión de gestión, de tal forma que quede inaccesible de cara a nuevos usuarios. Si se cumple este tiempo, la conexión será rechazada.

Un control similar se lleva a cabo en las sesiones de datos, aunque con fines ligeramente distintos. En ambas se controla el *timeout* de los paquetes, pero en la de espectador no se limita el tiempo de sesión. En la sesión actuadora, si así se especifica en la configuración, se lleva un control de la duración máxima que un usuario puede disponer de los motores, con el objetivo de evitar que un usuario monopolice su uso.

En el caso de que una conexión supere el tiempo máximo establecido entre paquetes, indicando de esta manera la ausencia (sin notificación) del usuario, supondría la expulsión del mismo de la aplicación, dada su inactividad. Un caso distinto se produce si el tiempo que se agota es el de sesión (sólo puede ocurrir en las actuadoras), pues la resolución tomada sería la de cambiar el modo de la sesión, relegando al actuador al rol de espectador.

El servidor también permite que las sesiones de datos se intercambien entre sí; es decir, un usuario de una sesión actuadora puede cambiar en cualquier momento su estado por de un espectador, con el consiguiente aviso a los demás espectadores acerca de la disponibilidad de la sesión actuadora. El caso contrario también es posible, pero siempre y cuando no haya una sesión actuadora activa.

En cualquier momento un cliente puede decidir abandonar la herramienta; para ello, enviará la correspondiente solicitud de desconexión. Ante este tipo de peticiones, el servidor da de baja al usuario de su tabla de conexiones, y libera los recursos correspondientes. Si el usuario que abandona es el actuador, se informará a todos los espectadores de dicha acción.

6.2.3 Análisis del Gestor USB

Desde el punto de vista del usuario, el gestor de USB es una aplicación pasiva, ya que, en apariencia, tan sólo muestra información acerca del estado del USB y los datos de todos los movimientos realizados, sin facilitar medios para realizar acciones. Esta característica se debe al hecho de que su principal función es la de hacer de intermediario entre el servidor y el USB.

La necesidad de utilizar esta aplicación separada del servidor se debe a que en el momento en que se desarrolla este proyecto, Java (lenguaje utilizado para construir la aplicación servidora) no cuenta con soporte USB para Windows (un grupo independiente ha desarrollado soporte para Linux, estando el de Windows aún en desarrollo, no ofreciendo por tanto, un soporte completo), plataforma para la que se desea la herramienta, por lo que debe recurrirse a recursos externos para realizar la comunicación. Además, el carácter genérico de la herramienta desarrollada, en que debe darse cabida multitud de propósitos de empleo, requiere una relación de las acciones realizadas e información de primera mano acerca de los posibles errores susceptibles de producirse, lo cual es posible gracias a la utilización de la API de Windows.

La aplicación está programada en C/C++, apoyándose en la API de Windows tanto para gestionar el USB como para comunicarse con el proceso del servidor, mediante el empleo de un pipe, el cual, en el lado del servidor, ha sido implementado utilizando código nativo, ya que el núcleo de la aplicación es Java.

6.2.3.1 Esquema de funcionamiento

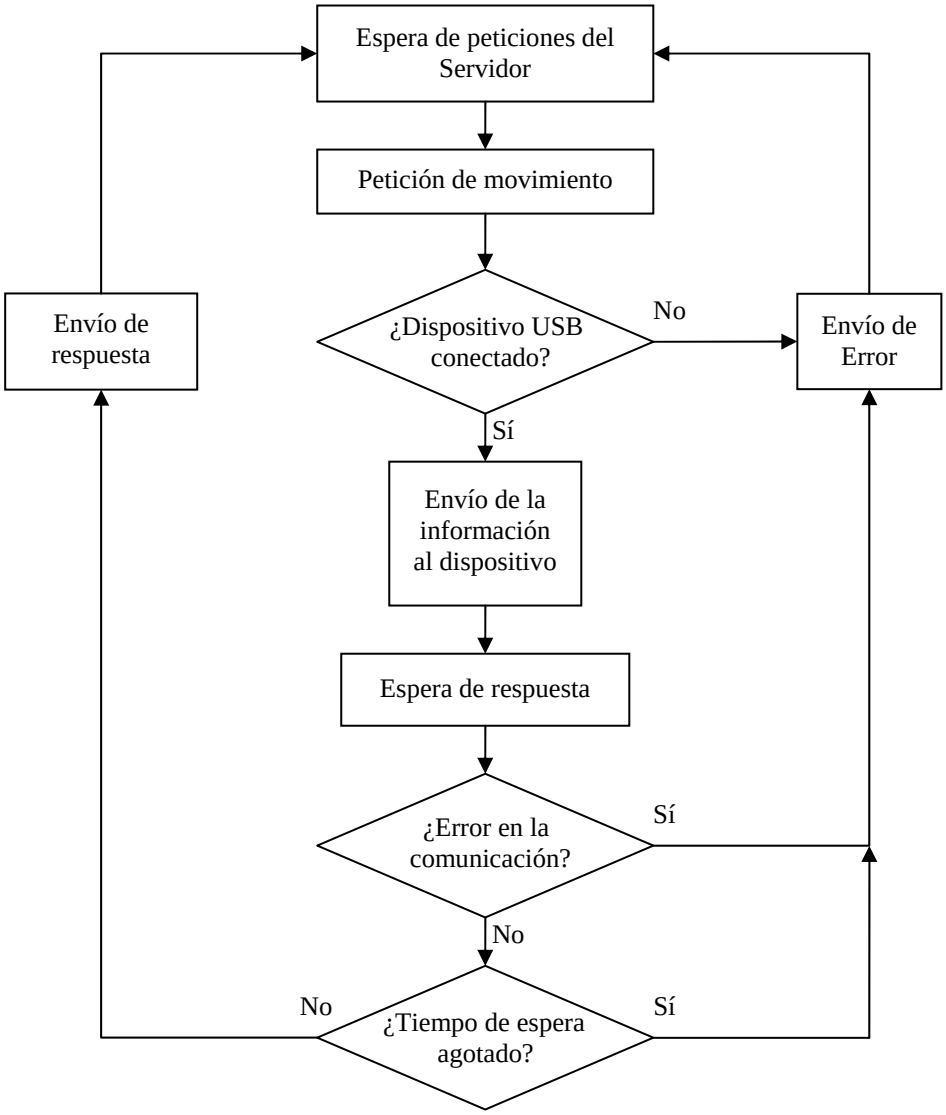


Figura 6-8 : Esquema de funcionamiento del gestor de USB

La figura 6-8 muestra el esquema de funcionamiento del gestor de USB, el cual, siguiendo la filosofía del sistema USB, sólo actúa cuando el proceso principal (en este caso el servidor) hace una petición.

En primer lugar, la aplicación debe asegurarse de que el dispositivo está conectado; en caso negativo, enviará un paquete de error indicando esta incidencia, a través del pipe de comunicaciones con el servidor. Tras la verificación, los datos pertinentes serán enviados al dispositivo, del cual se espera una respuesta. Es posible, aunque improbable, que el dispositivo quede bloqueado; para salvar esta situación, y bloquear a su vez al proceso que espera por una respuesta, se estima el tiempo que debe tardar el microcontrolador en

realizar esas operaciones (al alza), de tal forma que si transcurrido el mismo el dispositivo no ha contestado, indicará un error en el dispositivo.

Si todos los pasos se ejecutan de forma correcta y sin errores, antes de que transcurra el tiempo estimado el microcontrolador devolverá una respuesta, indicando las operaciones realizadas. En el caso que nos ocupa esta respuesta corresponderá con los datos enviados al dispositivo, ya que no es objetivo del presente proyecto evaluar la corrección con la que se ejecuta el movimiento, pero se deja abierta esa posibilidad, de tal forma que añadiendo al dispositivo periféricos que supervisen el movimiento, y en el programa controlador el código para su control, se podría ampliar de forma relativamente sencilla las opciones de la herramienta, ya que el resto de los componentes están preparados para esta posibilidad.

Una vez que el dispositivo ha respondido (tanto si el movimiento se ha realizado correctamente como si no), el gestor de USB pasa estos datos al servidor, para que realice las operaciones oportunas con ellos.

6.2.4 Análisis del Cliente

El cliente es la aplicación que deben utilizar los usuarios para hacer uso de la herramienta. Su objetivo principal es el de dar servicio a los usuarios remotos, a través de una interfaz con la que poder interactuar.

Básicamente soporta dos modos de funcionamiento:

- **Actuador**

  Podrán acceder a este modo de operación aquellos usuarios con permisos suficientes (es decir, con permisos de actuador o de administrador), siempre y cuando el recurso esté libre; es decir, siempre y cuando no haya otro usuario actuador en el sistema ejerciendo como tal, ni el administrador esté accediendo los motores.

  Este modo permite al usuario orientar los motores, bajo las restricciones de ángulo y velocidad impuestas en la configuración del servidor.

- **Espectador**

  Podrán acceder a este modo todos los usuarios registrados en la herramienta. Es posible acceder a este modo de forma voluntaria, o forzada, en caso de no tener permisos suficientes, ó si existe un usuario actuador moviendo los motores.

  En el modo espectador tan sólo es posible ver las evoluciones del motor, mostrando la posición actual, y la velocidad de movimiento.

En cualquier caso, tal y como se indicó en el Servidor, es posible, si se reúnen las condiciones necesarias, pasar de un modo a otro.

6.2.4.1 Funciones del Cliente

Entre las características más importantes del cliente cabe destacar:

- **Capacidad para gestionar la identificación de usuarios:** El cliente recoge los datos de identificación del cliente (nombre y contraseña), y lo envía al servidor, el cual emitirá una respuesta, ante la cual, el cliente actúa en consecuencia, expulsando o admitiendo, en el modo indicado, al usuario solicitante.

- **Capacidad para gestionar la desconexión de usuarios:** Ante una petición del usuario, puede indicar al servidor la intención de abandonar la herramienta. También es posible que sea el servidor el que quiera expulsar al usuario, emitiendo el cliente el correspondiente aviso.

- **Capacidad para gestionar modos de operación:** Desde la aplicación cliente, si se dan las condiciones de disponibilidad, el usuario puede cambiar el modo de operación, o actuar en consecuencia si es el servidor el que cree oportuno un cambio en el modo de conexión establecido.

- **Capacidad para interactuar con los motores:** Un usuario en modo actuador puede, desde el cliente de red, asignar una orientación a los motores, a una cierta velocidad, y obtener la posición resultante, sea cual sea su modo.

6.2.4.2 Esquema de funcionamiento

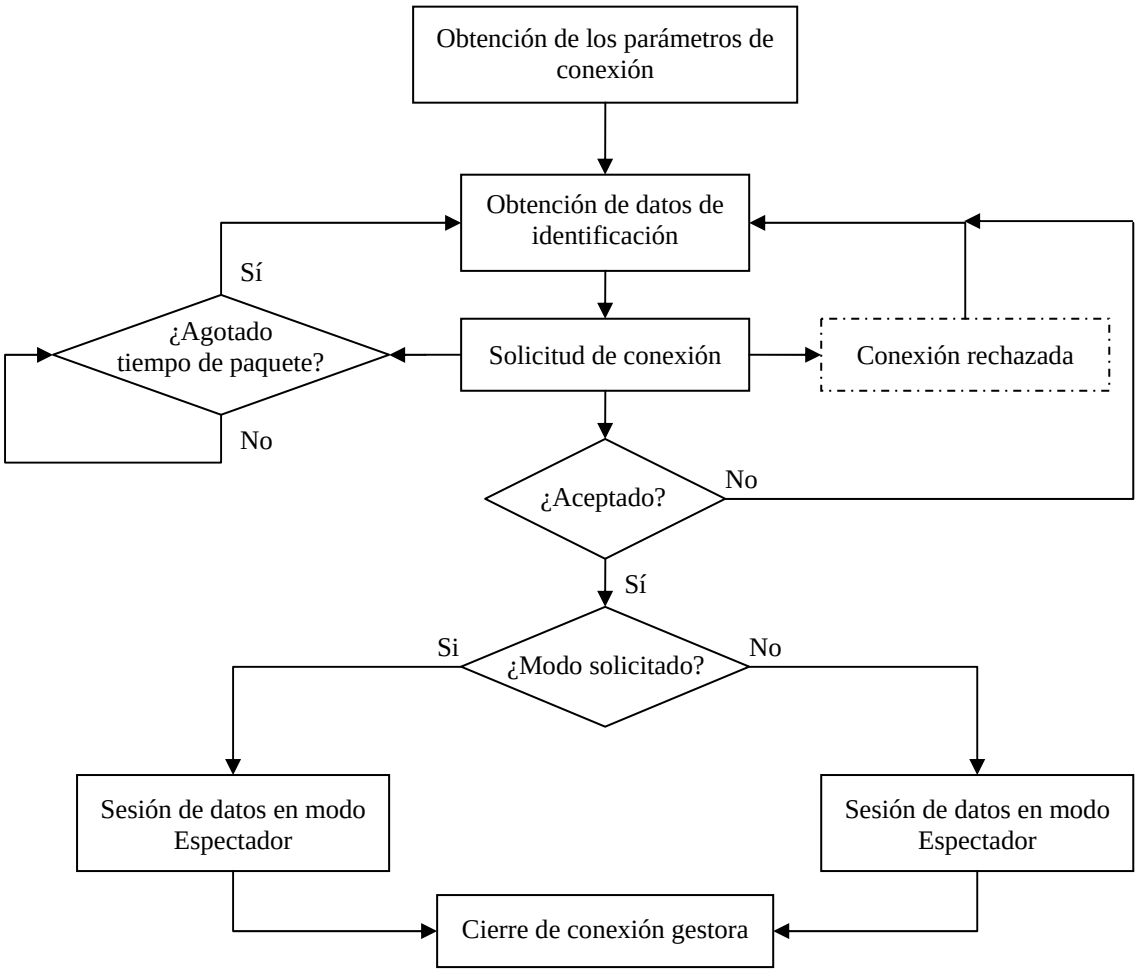


Figura 6-9 : Esquema de los pasos a seguir para establecer una sesión de usuario

El cliente ha sido implementado como un applet Java incrustado en una página Web, ya que de esta forma se facilita la accesibilidad a la aplicación (a través de cualquier navegador Web conectado a la red) y la portabilidad de la misma.

Esta característica, además, permite obtener ciertos recursos, como si de una página Web se tratara, residentes en el sistema servidor. De esta manera, el applet es capaz de obtener en el inicio de su ejecución, el archivo con los parámetros de red utilizados por el servidor, que a su vez serán utilizados para las conexiones con el mismo.

Una vez el usuario haya facilitado sus parámetros de identificación, se envía una petición de conexión al servidor, el cual responderá aceptando o denegando la sesión al usuario. En caso afirmativo, especificará también el modo en el que se permite su ingreso en la herramienta.

Todo el proceso de gestión se realiza utilizando una conexión TCP que el servidor habilita para este fin. Esta conexión cuenta con un limitador de tiempo, es decir, si no se llega a un acuerdo en el tipo de sesión antes de que se cumpla ese tiempo, la conexión será rechazada. Una vez que el servidor ha identificado al usuario, la sesión de datos se realiza en una nueva conexión, liberando la utilizada en la gestión para que pueda ser utilizada por un nuevo usuario.

Si el servidor rechaza al cliente, ya sea por no encontrar a un usuario con los datos de identificación indicados, o porque se ha alcanzado el número máximo de usuarios, la aplicación regresa al diálogo de petición de datos de identificación.

6.2.5 Sistema de comunicación Cliente-Servidor

Para que la comunicación entre ambos extremos sea posible, es necesario definir un mecanismo de comunicación aceptado por ambos. Por esta razón, se ha definido la estructura de las PDU a utilizar.

Figura 6-10 : PDU genérica

La figura 6-10 muestra el esqueleto de las características comunes de las PDU, que se centran en que todas comienzan con un campo que define el tipo, seguido por otro que especifica el subtipo, de forma que quede claro el cometido exacto de la transmisión. Estos campos también determinan el formato y número de campos de datos (en caso de haberlos) que contienen la información objeto de la transferencia.

- **Tipo**

Indica el tipo genérico de la PDU, es decir la operación con la que se relaciona. Existen 6 valores distintos para este campo, numerados de 0-5:

- **0 : Establecimiento de Conexión**. Utilizado al inicio de la conexión actuadora para asegurarse de que el que inicia la sesión de datos es el mismo que se identificó durante la conexión gestora

- **1 : Login.** Utilizado durante la conexión gestora, tanto para enviar la identificación del cliente, como para aceptar/denegar la operación por parte del Servidor.

- **2 : Datos.** Utilizado durante las sesiones de datos, es decir, para el envío de datos sobre los motores; desde los parámetros de configuración de los mismos, hasta las nuevas órdenes que pueda enviar un cliente, pasando, por supuesto, por el envío de los resultados de las mismas.

- **3 : Información de conexión.** La función de este tipo de paquetes se centra en mantener de forma correcta la conexión (enviando reconocimientos si se cumplen los *timeouts* de los paquetes) y de enviar la información correspondiente a la disponibilidad de la conexión actuadora a todos los espectadores, cuando ésta se produzca.

- **4 : Cambio de permisos.** Gestionan el proceso de cambio de modo de operación de los usuarios, es decir, envían las solicitudes de cambio de tipo de sesión, y devuelven las respuestas. Se utilizan tanto para pasar de actuador a espectador, como para el proceso contrario.

- **5 : Solicitud de desconexión.** Pueden ser enviadas tanto por el cliente como por el servidor. Existen tres casos en los que se puede solicitar una desconexión:

  - El cliente o el servidor deciden abandonar.
  - El cliente o el servidor no responden durante un largo periodo de tiempo.
  - El cliente ha agotado el tiempo que tenía asignado para la sesión.

▪ **Subtipo**

El objetivo de este campo es el de especificar el cometido de la PDU, dentro del rango de actuación definido por el campo de *Tipo*. Dentro de cada tipo, puede haber un número variable de subtipos. La tabla 6-1 muestra las PDUs utilizadas por esta herramienta, indicando los tipos y subtipos utilizados.

Función	Descripción	Tipo	Subtipo
ESTABLECIMIENTO DE CONEXIÓN	Autenticacion	0	0
LOGIN	Login	1	0
LOGIN	Aceptación	1	1
LOGIN	Denegación	1	2
DATOS	Parametros	2	0
DATOS	Posicion Motor	2	1
DATOS	Posicion motor (parcial)	2	2
DATOS	Error	2	3
INFORMACION CONEXIÓN	¿Estás ahí?	3	0
INFORMACION CONEXIÓN	Sí, sigo aquí	3	1
INFORMACION CONEXIÓN	Disponibilidad Actuador	3	2
INFORMACION CONEXIÓN	Error en los datos de pdu	3	3
CAMBIO PERMISOS	Solicitud TCP->UDP	4	0
CAMBIO PERMISOS	Solicitud UDP->TCP	4	1
CAMBIO PERMISOS	ACK	4	2
CAMBIO PERMISOS	NACK (UDP->TCP)	4	3
SOLICITUD DE DESCONEXION	Desconexion Total	5	0
SOLICITUD DE DESCONEXION	Tiempo agotado (TCP)	5	1
SOLICITUD DE DESCONEXION	Socket Timeout	5	2

Tabla 6-1 : Relación de PDUs utilizadas por la herramienta

6.3 Análisis estructural de la aplicación

La herramienta presentada en este proyecto implementa un gran número de funciones y emplea varias clases para realizarlas. Entre ellas es posible diferenciar 4 grandes grupos: las comunes para el servidor y el cliente, las específicas del cliente, las específicas del servidor, y las específicas del gestor de USB.

Para comprender con mayor profundidad el funcionamiento de cada uno de los elementos que componen la herramienta, a continuación se procede a analizar las clases de mayor relevancia, así como los métodos que las constituyen.

6.3.1 Clases compartidas por el servidor y el cliente

El hecho de que el servidor y el cliente utilicen las mismas clases para ciertos aspectos se debe esencialmente a que trabajan de forma muy similar (uno de los motivos por los que se ha empleado el mismo lenguaje de programación), por lo que éstas son útiles en ambos extremos. Estas clases son básicamente las que establecen y gestionan la comunicación entre ambos, algunos eventos, la presentación de la ayuda y el panel de la interfaz gráfica que controla los motores.

➢ Clases para la transferencia de información

Estas clases se encuentran en jerarquía, y su objetivo es el de dar una estructura a los paquetes de datos, de tal forma que sus campos puedan crearse y leerse de forma ordenada y segura. La jerarquía de clases puede verse en la figura 6-11.

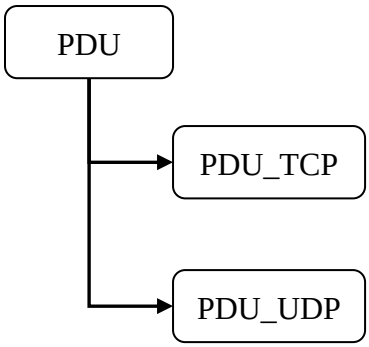


Figura 6-11 : Jerarquía de las clases de transferencia de información

❑ La clase PDU

Se trata de la clase de más alto rango dentro de la jerarquía de la figura 6-11. Su cometido es el de definir los parámetros que toda PDU debe tener, como las variables miembro, o las funciones básicas para añadir y rescatar campos.

La filosofía empleada se basa en las facilidades de *serialización* que las clases de Java ofrecen, las cuales permiten convertir el contenido de una clase en un flujo de bytes, susceptible de ser almacenado ó enviado a través de una red (como es nuestro caso), de tal forma que si en el otro extremo de la comunicación la aplicación cuenta con la implementación de dicha clase, es posible reconstruir la original. Esta posibilidad ofrece grandes garantías a la entrega íntegra de los datos, ya que éstos sólo serán correctos si coinciden perfectamente con los tipos especificados en la implementación de la clase remota, además de las comprobaciones propias, que java realiza en la carga de cada una de estas clases, añadiendo cierto nivel de seguridad a la transmisión.

Como punto en contra, aparece el hecho de que Java añade cierta cantidad de información aparte de los datos útiles, como el nombre de la clase, o cada uno de los tipos enviados (a partir de los cuales hará las comprobaciones). Dado el hecho de que el tipo y número de

campos de las PDU es variable, junto con el objetivo de minimizar la carga de datos de comprobación, se ha decidido enviar un único tipo de datos, que engloba a cualquiera de las susceptibles de ser enviados: *String*, agrupados dentro de un *Vector*. Cualquier dato puede ser incorporado a esta clase, y de ella se pueden sacar los datos en el formato que se requiera. De esta manera sólo se envía de dos clases, y no una por cada posible ocurrencia.

- **class PDU**

Definición de la clase

- **public PDU()**

  Constructor sin parámetros. Su misión es la de inicializar la única variable miembro: el *Vector* de campos.

```
public PDU(){
  datos = new Vector();
}
```

- **public void setTipo(int tipo, int subTipo)**

  Función miembro encargada de fijar el tipo y subtipo de la PDU. Para ello, en primer lugar inicializa el contenido de la PDU (ya que es habitual que la misma variable PDU se utilice para varios envíos), para luego colocar a la cabeza del vector los campos de tipo y subtipo.

```
public void setTipo(int tipo, int subTipo){
  datos.removeAllElements();
  datos.add((new String()).valueOf(tipo));
  datos.add((new String()).valueOf(subTipo));
}
```

- **public void addDato(String dato)**
- **public void addDato(int dato)**
- **public void addDato(double dato)**

  Estas funciones son las encargadas de añadir los datos a la PDU, según sea su tipo de origen. En ellas se convierte el dato en String, y se añade al Vector.

- **public short getTipo()**

Esta función se encarga de devolver el tipo de una PDU, para ello obtiene el primer elemento, y lo devuelve en formato *short*. Si esta conversión no sea posible, indicará un error en la PDU (devolviendo un *tipo* imposible como es 0xFF).

```
public short getTipo(){
    try{
      return (new Short( (String) datos.elementAt(0))).shortValue();
    }
    catch(NullPointerException e){return 0xFF;}
    catch(ArrayIndexOutOfBoundsException e){return 0xFF;}
  }
```

- **public short getSubtipo()**

Su implementación es idéntica a la función anterior, con la salvedad de que accede al segundo elemento del vector, en lugar del primero.

- **public String getCampo(int campoDATO)**

Devuelve el campo indicado por la variable campoDATO, en formato de *String*.

```
public String getCampo(int campoDATO){
   try{
     return (String) datos.elementAt(campoDATO+2);
   }
   catch(ArrayIndexOutOfBoundsException e){return null;}
  }
```

- **public int getIntcampo(int campoDATO) throws NumberFormatException**

Devuelve el campo indicado por la variable campoDATO, en formato entero. En caso de que la información contenida en ese campo no sea un entero, se lanzará una excepción *NumberFormatException* que indicará un error en la PDU.

```
public int getIntcampo(int campoDATO) throws NumberFormatException{
  try{
```

```
    return (int) (new Integer((String)(datos.elementAt(campoDATO+2)))).intValue();
  }
  catch(ArrayIndexOutOfBoundsException e){return 0xFFFF;}
  catch(NumberFormatException e){return 0xFFFF;}
 }
```

- **public double getDoubleCampo(int campoDATO) throws NumberFormatException**

Devuelve el campo indicado por la variable campoDATO, en formato *double*. En caso de que la información contenida en ese campo no sea un *double*, se lanzará una excepción *NumberFormatException* que indicará un error en la PDU. SU implementación es similar al caso anterior.

- **La clase PDU_TCP**

Segunda clase de la jerarquía, que hereda la variable y los métodos miembros de la clase PDU, pero añadiendo funciones para la serialización de la misma, lo que hace posible su transmisión utilizando un socket TCP.

- **class PDU_TCP extends PDU**

Definición de la clase, que hereda de PDU.

- **public PDU_TCP()**

Constructor sin parámetros. Se limita a llamar al constructor de la clase de la que hereda

- **public PDU_TCP(PDU_TCP pdu)**

Constructor de copia. Crea una nueva PDU_TCP a partir de otra creada previamente.

- **public synchronized Vector getInfo()**

Devuelve el Vector que contiene los campos de la PDU. Su utilidad centra en facilitar el contenido de la PDU, de tal forma que pueda ser serializada y enviada.

- **public synchronized void setInfo(Vector datos)**

Esta función actualiza los campos de la PDU a partir del Vector dado. Se utiliza en la recepción de PDUs, tras haber recibido el objeto serializado

```
public synchronized void setInfo(Vector datos){
    this.datos = datos;
}
```

- **La clase PDU_UDP**

Los paquetes asociados a sockets UDP presentan la peculiaridad de que, en recepción, debe establecerse un tamaño fijo para el paquete de datos, que en nuestro caso, tiene un tamaño variable. Si se fija un tamaño demasiado pequeño se perderán datos, y si es demasiado grande, se añade "basura" a la información útil, con el fin de rellenar el espacio.

La opción "menos mala" es la segunda, pero para que los bytes añadidos no afecten a la información útil, se debe añadir el tamaño al inicio del paquete (antes de los datos serializados, pues si fuera parte de los mismos, su inclusión sería inútil, ya que el cargador de clases Java fallaría igualmente), que en nuestro caso se hará mediante dos bytes (más que suficiente).

- **class PDU_UDP extends PDU**

Definición de la clase, que hereda de PDU.

- **public PDU_UDP()**

  Constructor sin parámetros. Se limita a llamar al constructor de la clase de la que hereda, e inicializa la variable miembro utilizada para medir la longitud de la PDU.

- **public PDU_UDP(PDU_UDP pdu)**

  Constructor de copia. Crea una nueva PDU_UDP a partir de otra creada previamente.

- **public synchronized byte[] getBytes()**

  Esta función miembro es la encargada de generar la serialización del objeto y añadir los dos bytes que indican el tamaño del mismo.

```
...
ObjectOutputStream out;

 out.writeObject(datos);      //serialización
 size = Bout.size();          //longitud del objeto
…
```

- **public synchronized void setBytes(byte[] array) throws IOException**

En este caso se hace la operación contraria, a partir de una secuencia de caracteres, se obtiene la PDU.

➢ **Clases de control de sockets**

Existen cuatro clases de este tipo, dos por protocolo: una para la transmisión y otra para la recepción.

❑ **La clase PaqueteTCPtransmision**

Como su propio nombre indica, su función es la de enviar PDUs TCP. Para ello, a partir de un socket TCP dado (se hace de esta manera para que pueda ser configurado según necesidades) crea un *stream* de objetos.

▪ **class PaqueteTCPtransmision**

Definición de la clase.

- **public PaqueteTCPtransmision(Socket cliente) throws IOException**

Constructor de la clase. A partir del *socket* dado, crea un stream de objetos.

- **public synchronized void close()**

Función miembro utilizada para cerrar el *stream* de objetos.

- **public synchronized boolean enviar(PDU_TCP pdu)**

Función utilizada para enviar las PDUs

```
public synchronized boolean enviar(PDU_TCP pdu) {
    try{
```

```
  out.flush();
  out.writeObject(new Vector(pdu.getInfo()));
  return true;
   }
catch(NullPointerException e) {return false;}
catch(SocketTimeoutException e){return false;}
catch(IOException e){return false;}
}
```

- **La clase PaqueteTCPrecepción**

Clase empleada para recibir PDU_TCP. Indica la recepción de un paquete a través de un evento.

- **class PaqueteTCPrecepcion extends Thread**

  Definición de la clase. Hereda de la clase *Thread*, por tanto implementa un hilo de ejecución.

  - **public PaqueteTCPrecepcion()**

  Se limita a crear una variable PDU_TCP.

  - **public void setParam(short id, Socket cliente)**

  Función miembro utilizada para indicar los parámetros de conexión, como son el socket a utilizar, y la identificación de conexión (gestión ó actuadora). Este identificador se utiliza a la hora de lanzar el evento de recepción.

  - **private short recibir() throws SocketException{**

  Esta función espera (dentro de los límites establecidos por el socket) hasta que llegue un objeto serializado.

  - **public void addTCPListener(PaqueteTCPListener l)**

  Añade un objeto a la lista de *listeners*; es decir, a la lista de objetos a los que avisar cuando se reciba una PDU_TCP.

- **public void removeTCPListener(PaqueteTCPListener l)**

Elimina un objeto de la lista de *listeners*.

- **protected synchronized void avisaRecepcion(short suceso, short id, PDU_TCP pdu)**

Esta función es la encargada de avisar a todos los objetos registrados de que se ha recibido una PDU (también informa de errores). Para ello, llama a la función *recibido(EventoPaqueteTCP)* con la que todos cuentan, ya que implementan la misma interfaz.

```
protected synchronized void avisaRecepcion(short suceso, short id, PDU_TCP pdu) {

    EventoPaqueteTCP event;
        ...
    Vector targets;
    synchronized (this) {
      targets = (Vector) listeners.clone();
    }

    Enumeration enum = targets.elements();
    while (enum.hasMoreElements()) {
    PaqueteTCPListener l = (PaqueteTCPListener) enum.nextElement();
    l.recibido(event);
    }
```

- **public void run()**

Esta función forma parte de la herencia dejada por la clase *Thread*, siendo su función principal, ya que en ella se realizan las operaciones del hilo. En este caso se trata de un bucle infinito en espera de conexiones.

```
...
while (true){
  avisaRecepcion(recibir(),id,pdu);
  }
…
```

- **La clase PaqueteUDPtransmision**

Esta clase obtiene la PDU serializada, una vez añadida su longitud, y la envía al extremo remoto.

- **class PaqueteUDPtransmision**

Definición de la clase

- **public PaqueteUDPtransmision(int puerto, DatagramSocket socket)**

  Constructor de la clase. Requiere el Puerto a utilizar, y el socket, una vez inicializado,

- **public short setAddress(InetAddress host)**

  Función miembro que permite cambiar la dirección destino de la PDU.

- **public short enviar(PDU_UDP pdu)**

  A través de esta función se envía la PDU, una vez obtenido el objeto serializado, a la dirección previamente establecida.

```
public short enviar(PDU_UDP pdu) {
  if (addr!=null) {
    try {
      DatagramPacket pack = new DatagramPacket(pdu.getBytes(),(pdu.size+2),
                                        addr, puerto);
       socket.send(pack);
      return 0;
    }
    …
}
```

- **La clase PaqueteUDPrecepción**

El cometido de esta clase se basa en la recepción de las PDU_UDP. Para ello obtiene el objeto serializado, y lo convierte en una PDU_UDP, informando de cualquier incidencia.

- **class PaqueteUDPrecepcion extends Thread**

  Definición de la clase. Hereda de la clase *Thread*, por tanto implementa un hilo de ejecución.

  - **public PaqueteUDPrecepcion(DatagramSocket socket) throws IOException**

  Constructor de la clase, a partir del socket UDP a utilizar.

  - **private byte[] conformar(byte[] dato)**

  Esta función extrae la PDU_UDP del paquete UDP recibido.

  - **private short recibir() throws SocketException**

  Función que permanece a la escucha (dentro de los parámetros de temporización establecidos en el socket), en espera de un paquete UDP

  ```
  socket.receive(packet);
  ```

  - **public void addUDPListener(PaqueteUDPListener l)**

  Añade un objeto a la lista de *listeners*; es decir, a la lista de objetos a los que avisar cuando se reciba una PDU_UDP

  - **public void removeUDPListener(PaqueteUDPListener l)**

  Elimina un objeto de la lista de *listeners*.

  - **protected void avisaRecepcion(short suceso, InetAddress addr, PDU_UDP pdu)**

  Esta función es la encargada de avisar a todos los objetos registrados de que se ha recibido una PDU (también informa de errores). Para ello, llama a la función *recibido(EventoPaqueteUDP)* con la que todos cuentan, ya que implementan la misma interfaz.

- **public void run()**

Esta función forma parte de la herencia dejada por la clase *Thread*, siendo su función principal, ya que en ella se realizan las operaciones del hilo. En este caso se trata de un bucle infinito en espera de conexiones.

```
…
 while (true){
    suceso = recibir();
    if (suceso == 0){
      pdu.setBytes(conformar(packet.getData()));
      avisaRecepcion(suceso, packet.getAddress(), pdu);
  }
…
```

➢ **Clase para la presentación de la ayuda**

Ambas aplicaciones cuentan con una ayuda, la cual se presenta utilizando esta clase.

❑ **La clase PanelAyuda**

Esta clase presenta una nueva ventana en la que elegir entre la ayuda de la aplicación o información *acerca de* la aplicación.

▪ **public class PanelAyuda extends Jframe**

Definición de la clase. Hereda los métodos de JFrame con el fin de habilitar una nueva ventana.

- **public PanelAyuda(boolean esApplet)**

Constructor de la clase. Toma como único parámetro un valor lógico que indica si la clase se está ejecutando en el applet (ya que la forma de obtener el archivo de ayuda difiere según el caso). En esta función se define tanto la colocación de los elementos en la ventana de elección (dos botones) como sus acciones.

```
public PanelAyuda(boolean esApplet) {
   super("USBMotor : Ayuda");
   this.applet = esApplet;
```

```
    JButton ayuda = new JButton("Ayuda de la aplicacion");
    JButton acerca = new JButton("Acerca de...");
     ...
    /**listeners*/
    ayuda.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
        try {
          JEditorPane izquierda;
          JSplitPane divisor;

          if(applet){
            URL url = new URL(CodeBase,"Ayuda/menu.htm");
            izquierda = new JEditorPane(url);
            url = new URL(CodeBase,"Ayuda/inicio.htm");
            derecha = new JEditorPane(url);
          }
          else{
            File fichero = new File("Ayuda/menu.htm");
            izquierda = new JEditorPane(fichero.toURL());

            fichero = new File("Ayuda/inicio.htm");
            derecha = new JEditorPane(fichero.toURL());
          }
          …
           }
          });
...
} });
    acerca.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
        foo().setVisible(false);
        Acerca acerca = new Acerca();
      } });
  }
```

- **Clases de la Interfaz gráfica**

Tanto el cliente como el servidor comparten uno de los paneles con los que interactúa con el usuario: el que controla los motores, y un componente gráfico que éste panel incluye: el *Dial*.

- **La clase Dial**

Esta clase se encarga de crear un componente gráfico que imita a un dial, es decir, una rueda con un cursor. En nuestro caso, en realidad, tendrá dos: una para indicar la posición en la que se encuentra el motor (blanco), y otra para indicar el nuevo movimiento (en caso de contar con permisos de actuador). La figura 6-12 muestra el aspecto de este nuevo componente.

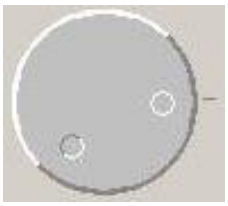


Figura 6-12 : Componente Dial

- **public class Dial extends JComponent implements MouseListener,MouseMotionListener**

Definición de la clase. Hereda de la clase JComponent, adquiriendo sus métodos, e implementa las interfaces de eventos del ratón *MouseListener* y *MouseMotionListener*.

Para indicar un cambio en el Dial, lanza un evento a todos los objetos registrados en la lista de eventos.

- **public Dial(int n_pasos, int valor_inicial, boolean permite_movimiento)**

  Constructor de la clase. Se encarga de inicializar los límites de giro, y el valor inicial de los cursores.

- **public void setValor(int paso)**

- **public void setValor(double angulo)**

  Fija la posición del cursor móvil. Uno dado en pasos y el otro en grados.

- **public void setValorReal(int n_pasos)**

  Fija la posición del cursor fijo (que indica la posición real).

- **public void mouseDragged(MouseEvent e)**

Sobrecarga del método declarado en la interfaz *MouseMotionListener*. De esta manera se avisa de cuándo el cursor está siendo movido. De esta manera se lanza un evento a la clase Dial para que lo *repinte*, es decir, actualice el estado gráfico del componente.

- **protected void spin(MouseEvent e)**

A partir de la posición del ratón, determina el ángulo recorrido por el cursor, a partir del cero tomado como referencia.

```
protected void spin(MouseEvent e){
  ...
  double th = Math.atan((1.0 * y - radio) / (x - radio));
  int valor = ((int)(th / (2 * Math.PI) * (valorMaximo - valorMinimo)));
  ...
 }
```

- **private void dibujaCirculo3D(Graphics g, int x, int y, int radio, boolean elevado)**

Función miembro que imita un círculo con relieve.

```
private void dibujaCirculo3D(Graphics g, int x, int y,
                        int radio, boolean elevado){
  …
  g.setColor(elevado ? light : dark);
  g.drawArc(x,y,radio * 2,radio *2,45,180);
  g.setColor(elevado ? dark : light);
  g.drawArc(x,y,radio * 2,radio * 2,225,180);
}
```

- **private void dibujaCirculo2D(Graphics g, int x, int y, int radio)**

Dibuja un círculo de borde claro.

```
private void dibujaCirculo2D(Graphics g, int x, int y,
                        int radio){
  ...
  g.drawOval(x,y,2 * radio, 2 * radio);
}
```

- **public void paintComponent(Graphics g)**

Se trata de la sobrecarga del método de JComponent, encargado de dibujar el componente.

```
public void paintComponent(Graphics g){
 ...
 g2.setPaint(getForeground().darker());
 g2.drawLine(radio * 2 + tick / 2, radio, radio * 2 + tick, radio);  //linea de origen
 g2.setStroke(new BasicStroke(2));
 dibujaCirculo3D(g2,2,2,radio,true);    //círculo externo
 int knobRadio = radio / 7;
 ...
 dibujaCirculo2D(g2, x + radio - knobRadio + 2, y + radio - knobRadio + 2,
          knobRadio); //círculo fijo
 ...
 dibujaCirculo3D(g2, x + radio - knobRadio +2, y + radio - knobRadio +2,
          knobRadio , false);  //círculo movil
}
```

- **La clase PanelMovimiento**

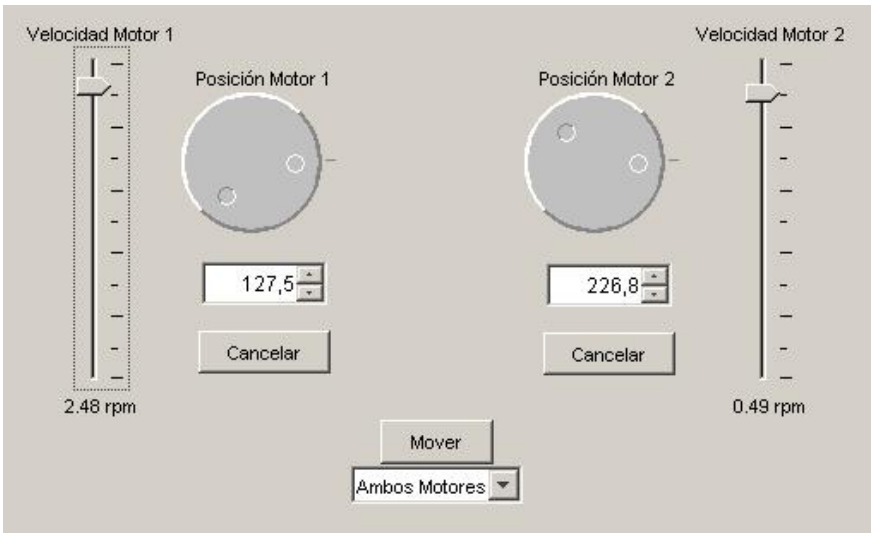


Figura 6-13 : Panel de Movimiento

Esta clase se ocupa de la creación de un panel a través del cual el usuario pueda interactuar con los motores, ya sea para ordenar movimientos, como para recibir información acerca de los mismos. Para ello, cada motor cuenta con un componente "Dial" que definirá la orientación del motor (asociado a un diálogo *Spinner*), una barra

deslizante con la que definir su velocidad y un botón que permita cancelar el valor asignado, antes de realizar la acción (botón Mover). Además, el panel presenta un botón que permite ejecutar el movimiento y una lista para elegir qué motor mover (incluso ambos). La figura 6-13 muestra el resultado visual de esta clase.

- **public class PanelMovimiento extends JPanel**

Definición de la clase. Como se observa, hereda de la clase *JPanel*, la cual facilita un contenedor en el que colocar componentes de interfaz gráfica.

- **public PanelMovimiento(int n_pasos1, int n_pasos2,**
  int paso_inicial1, int paso_inicial2,
  int velocidad_inicial1,int velocidad_inicial2,
  double anguloMaximo1,
  double anguloMaximo2,
  short velocidadMaxima1,
  short velocidadMaxima2,
  boolean permisos_movimiento)

Único constructor de la clase. Su cometido es el de inicializar todos los elementos que serán incluidos en el panel, además de colocarlos en su posición correspondiente, para lo que requiere gran cantidad de parámetros que especifiquen su comportamiento; indican el número de pasos del motor a mover, el ángulo máximo de giro, la velocidad o la posición inicial. Además se incluye un parámetro de tipo *booleano,* cuyo cometido es el de indicar si se permite o no el movimiento, de esta manera, un usuario espectador no podrá acceder al control de los motores, ya que todos los componentes del panel estarán deshabilitados (tan sólo podrá ver los cambios producidos).

Una vez seleccionada una posición, y pulsado el botón de movimiento, éste avisará a todos los objetos registrados como *"listeners"*, los cuales actuarán en consecuencia.

```
public PanelMovimiento(int n_pasos1, int n_pasos2,
        int paso_inicial1, int paso_inicial2,
        int velocidad_inicial1,int velocidad_inicial2,
        double anguloMaximo1,
```

```
                double anguloMaximo2,
                short velocidadMaxima1, short velocidadMaxima2,
                boolean permisos_movimiento){

//inicializamos los objetos
  dial1 = new Dial(n_pasos1,paso_inicial1,permiso_movimiento);
  dial1.setName("dial1");
  dial2 = new Dial(n_pasos2,paso_inicial2,permiso_movimiento);
  dial2.setName("dial2");

  listaMovimientos = new JComboBox(); //selecciona entre los motores a mover
  listaMovimientos.setEnabled(permisos_movimiento);
  listaMovimientos.setEditable(false);
  listaMovimientos.setName("lista");
  listaMovimientos.addItem("Ambos Motores");
  listaMovimientos.addItem("Motor 1");
  listaMovimientos.addItem("Motor 2");
  ...
                /**Listeners:*/
  moverButton.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
    if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
      equals("Ambos Motores")){
     avisaMovimiento(dial1.getValor(),  //enviamos aviso para ambos motores
                dial2.getValor(),
                velocidadSlider1.getValue(),
                velocidadSlider2.getValue());
     reiniciarButton1.setEnabled(false);  //botones que cancelan el valor del Dial
     reiniciarButton2.setEnabled(false);
     …
   });

 //cambio en el valor del Dial
  dial1.addDialListener(new DialListener(){
   public void dialAjustado(DialEvent ev){
    if (!angulo_motor1.getValue().toString().equals(ev.getValor()))
     angulo_motor1.setValue(ev.getValor());

     //analizamos si el motor afectado es susceptible de ser movido
     if ((permiso_movimiento) && (quieto))
```

```
            if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
                equals("Ambos Motores") ||
               listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
                equals("Motor 1")){
               moverButton.setEnabled(permiso_movimiento);
               reiniciarButton1.setEnabled(true);
             }
          }
        });
    …
```

- **public void setDatos(int paso1,int paso2, int velocidad1, int velocidad2, boolean actuador)**

A través de esta función se indica a la clase que un movimiento ha sido ejecutado correctamente, actualizando los valores de cada unos de sus componentes. El parámetro *booleano* indica si se debe dar permiso o no de movimiento al panel.

- **public void falloMover()**

Con la llamada a esta función se indica a la clase que un movimiento ha sido realizado incorrectamente, por lo que deben ser restaurados los valores anteriores de posición.

- **public void setPermisoMovimiento(boolean permiso)**

Modificando el permiso de movimiento se habilitan o deshabilitan los componentes del panel, de acuerdo con el parámetro *permiso*.

- **private String cuentaRevoluciones(int velocidad,int n_pasos)**

Con esta función la clase es capaz de presentar una estimación de la velocidad a la que se producirá el movimiento (en rpm), en función del valor asignado en el Slider de velocidad, y el número de pasos del motor. Para ello se basa en la temporización que producirán estos parámetros en el microcontrolador

```
private String cuentaRevoluciones(int velocidad,int n_pasos){
  Double rev = new Double(60.0 / (((0xFF - velocidad+1) * 256 * 0x40 *
                          4.0/6000000.0) * (2.0 * n_pasos)));
  //(carga_timer{C A2} * 64_repeticiones * 4(divisor) / frec_oscilador)
```

```
    // * 2 * numero_pasos = tiempo que tarda en dar 1 vuelta
    ...
}
```

6.3.2 Las Clases del Servidor

Dadas las características propias del Servidor, y los recursos que gestiona, podemos clasificar las clases en cinco grandes grupos: Gestión de Red, Gestión de Usuarios, Configuración, Gestión de USB e Interfaz gráfica.

➢ **Clases de Gestión de Red**

La misión de estas clases es controlar el intercambio de PDUs con el cliente, a partir de la información sobre los recursos con las que cuenta el servidor y mantener la conexión con los usuarios.

❑ **La clase entradaConexión**

Esta clase sirve de soporte para cada una de las entradas de la tabla de conexiones activas. Su utilidad se reduce a almacenar los valores para cada entrada, y proveer de métodos para el acceso a los mismos.

Los campos almacenados son los de: login, dirección IP, tipo de conexión y último acceso.

▪ **class entradaConexion**

Definición de la clase.

- **public entradaConexion()**

  Constructor por defecto. Asigna un valor conocido a cada campo.

- **public entradaConexion(String login, InetAddress direccion, short tipo)**

  Constructor con parámetros. Asigna el valor correspondiente a cada uno de los campos, además de asignar la fecha y ahora actual al campo que indica el último acceso.

- **public entradaConexion(entradaConexion conexion)**

Constructor de copia. Crea una entrada con los mismos valores que la indicada por parámetros.

- **protected void setUltimoAcceso()**

Obliga a la entrada a actualizar la fecha y hora del último acceso.

- **protected Date getUltimoAcceso() throws NullPointerException**

Devuelve la fecha y hora del último acceso.

❑ **La clase TablaConexiones**

Implementa una tabla, suyas entradas son objetos del tipo *entradaConexión*. Se utiliza para llevar el control sobre los usuarios conectados. La clase lanza un evento del tipo *EventoConexion* cuando se añade o se elimina una entrada.

▪ **class TablaConexiones**

Definición de la clase.

- **public TablaConexiones(int MAX_CONEXIONES)**

Constructor de la clase. Crea una tabla, cuyo número máximo de entradas es *MAX_CONEXIONES*.

- **protected synchronized int anadirConexion(String login, InetAddress direccion,short tipo)**

Añade una entrada a la tabla de conexiones, a partir de los datos facilitados en los parámetros.

- **protected synchronized boolean eliminarConexion(String login)**

Elimina la conexión cuyo campo *login* corresponda con el dado por parámetros.

- **protected synchronized boolean eliminarConexion(int pos)**

Elimina las conexiones cuyo campo *tipo* corresponda con el dado por parámetros.

- **protected synchronized boolean eliminarConexion(InetAddress addr)**

Elimina la conexión cuyo campo *dirección* corresponda con el dado por parámetros.

- **protected synchronized void desconectarTodos()**

Libera todas las conexiones.

- **protected synchronized boolean cambioTipo(entradaConexion entrada, short tipo)**

Este método hace posible el cambio del tipo de conexión (pasar de actuador a espectador, o viceversa).

- **protected entradaConexion getEntrada_login(String login)**

Devuelve la entrada correspondiente al *login* indicado.

- **protected entradaConexion getEntrada_direccion(InetAddress direccion)**

Devuelve la entrada correspondiente a la dirección indicada

- **protected entradaConexion getEntrada(int pos)**

Devuelve la entrada correspondiente a la posición indicada, dentro de la tabla. Se utiliza para recorrer la tabla.

- **protected entradaConexion getEntrada_tipo(int tipo,int inicio)**

Devuelve la primera entrada cuyo tipo corresponda con el indicado, a partir de una posición dada.

- **protected synchronized boolean actualizarAcceso_login(String login)**

Actualiza el campo de último acceso de la entrada correspondiente al *login* indicado.

- **protected synchronized boolean actualizarAcceso_direccion(InetAddress direccion)**

Actualiza el campo de último acceso de la entrada correspondiente a la *dirección* indicada.

- **protected synchronized boolean crearActuadorFicticio()**

Método utilizado para crear un actuador ficticio, es decir, que en realidad no existe. Se utiliza para permitir que el administrador adopte el rol de actuador (es decir mueva los motores), con las consecuencias que ello conlleva: Ningún usuario podrá ser actuador, ya que hay uno activo.

- **protected synchronized void eliminarActuadorFicticio()**

Una vez el administrador decide dejar de utilizar los motores, se libera el usuario ficticio, permitiendo que otros usuarios puedan utilizar el modo actuador.

- **synchronized public void addConexionListener(ConexionListener l)**

Añade un objeto a la lista de *listeners*; es decir, a la lista de objetos a los que avisar cuando se añada o elimine una entrada a la tabla de conexiones.

- **synchronized public void removeConexionListener(ConexionListener l)**

Elimina un objeto de la lista de *listeners*.

- **protected synchronized void avisaTabla(short suceso, entradaConexion entrada)**

Envía un evento a todos los objetos apuntados en la lista de listeners.

- **La clase PaqueteTCPServidor**

Esta clase se encarga del mantenimiento de la conexión TCP con el usuario y de controlar el cumplimiento de los tiempos de sesión asignados.

- **class PaqueteTCPServidor extends Thread**

Definición de la clase. Hereda de Thread, ya que requiere de un hilo de ejecución independiente para su funcionamiento.

- **public PaqueteTCPServidor (int puerto, short id, int socketTimeout, long tiempo_conexion, int puerto_udp, PaqueteTCPListener l) throws IOException**

Constructor de la clase. Se encarga de crear el socket de Servidor (permanece a la escucha en espera de conexiones) e inicializar el objeto controlador de PDUs en recepción *(PaqueteTCPrecepcion)*, ya que el de transmisión se crea una vez aceptada la conexión.

Una vez asignados todos los parámetros, inicia la ejecución del thread, con el fin de establecer la comunicación y controlar sus temporizaciones.

```
public PaqueteTCPServidor (int puerto, short id, int socketTimeout,
                           long tiempo_conexion, int puerto_udp,
                           PaqueteTCPListener l)
                           throws IOException{
  ...
  this.serverSocket = new ServerSocket(puerto);
  ...
  rx = new PaqueteTCPrecepcion();
  ...
  start();//inicia la comunicacion
}
```

- **private synchronized void esperar()**

Función encargada de detener el *thread*, una vez iniciada la conexión, hasta que finalice la conexión o se cumpla el tiempo de sesión.

```
private synchronized void esperar(){
   try{
    wait(tiempo_conexion);  //si es 0, esperará indefinidamente
   }
   catch(InterruptedException e){ }
}
```

- **public void run ()**

Sobrecarga del método principal de la clase Thread. En esta función se espera a que el cliente remoto conecte, tras lo cual se crea un objeto de envío de PDU_TCP *(PaqueteTCPTransmisión)* y se mantiene a la espera del fin de la conexión o de que termine la sesión.

Si la conexión que controla es la gestora, una vez finalice (por el motivo que sea), se reinicia, con el objetivo de esperar por un nuevo cliente. Es decir, sólo finaliza la ejecución del *thread* cuando se cierra el servidor. Además, espera indefinidamente por la conexión de usuarios (mientras permanezca abierto el servidor).

Por otro lado, la conexión actuadora esperará la conexión durante un tiempo limitado (para así no bloquear el recurso). Una vez finalice la sesión, también termina la ejecución del *thread*, volviendo a ser creado una vez se identifique un nuevo usuario como actuador.

```
public void run (){
  try{
     do{
     …
     socket = serverSocket.accept();   //espera de conexiones

     rx.setParam(id, socket);              //inicio del objeto de rx
     socket.setSoTimeout(socketTimeout);
     tx = new PaqueteTCPtransmision(socket); //inicio del objeto de tx
     rx.start();

     esperar();
      …
    }while ((id == gestor) && getFinalizar() == false);
    close();  //al salir definitivamente
  }
  …
}
```

- **public synchronized boolean enviar(PDU_TCP pdu)**

Método utilizado para enviar una PDU_TCP al cliente remoto.

- **public synchronized void close()**

Finaliza la conexión. Para ello primero envía una despedida al cliente remoto, tras lo cual "despierta" al thread para que finalice y libera los recursos utilizados (objetos de gestión de PDUs).

- **La clase PaqueteUDPServidor**

A diferencia de lo ocurrido en PaqueteTCPServidor, esta clase es completamente autónoma en lo que a gestión de PDUs se refiere, ya que es la encargada directa de su procesamiento. Además, implementa un temporizador que cada cierto tiempo encuesta a los clientes, con el fin de confirmar su presencia (ya que el protocolo UDP no está orientado a conexión). El esquema de funcionamiento de esta clase es el representado en la figura 6-14.

Para un control más eficiente de las PDUs, se ha implementado una lista de peticiones, de tal forma que el *thread* va cogiendo y procesando PDUs de la misma. En caso de que se agoten las peticiones, el *thread* pasa a un estado inactivo hasta que se reciba otra.

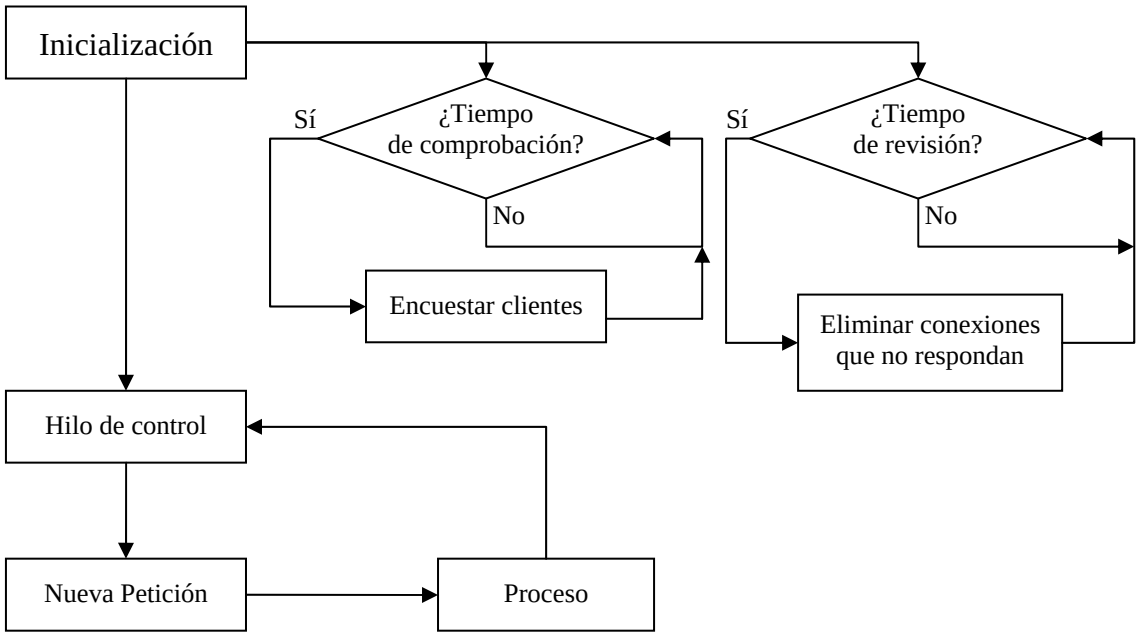


Figura 6-14 : Esquema de funcionamiento de la clase *PaqueteUDPServidor*

- **class PaqueteUDPServidor extends Thread implements PaqueteUDPListener**

Definición de la clase. Hereda de *thread*, ya que requiere de un hilo de ejecución independiente para realizar sus tareas. Implementa la interfaz de eventos PaqueteUDPListener, con el objetivo de conocer cuándo llega una PDU_UDP.

- **public PaqueteUDPServidor(int puerto_gestion, int UDPtimeout, int TCPtimeout, tiempoConexionTCP, int tiempo_revisiones, TablaUsuarios usuarios, TablaConexiones conexiones, PaqueteTCPServidor[] tcp, PaqueteTCPListener listenerTCP, ControlEventoServidor eventoServidor) throws IOException**

Constructor de la clase. Debido a la autonomía de la clase, requiere mucha información, incluso de las conexiones TCP, con el objetivo de poder realizar los cambios de modo.

Inicializa las clases de gestión de PDU_UDP y los *timers* encargados de encuestar a los clientes. Tras esto, inicia el *thread* con el objetivo de comenzar la gestión de peticiones.

- **public synchronized boolean enviar(PDU_UDP pdu, InetAddress addr)**

Envía una PDU a la dirección especificada.

- **public synchronized boolean multicast(PDU_UDP pdu)**

Función que envía un paquete a todos los clientes espectadores.

```
public synchronized boolean multicast(PDU_UDP pdu){
  ...
    while (correcto == true) {
      i = conexiones.buscaTipo(conexiones.udp, i);
      entrada = conexiones.getEntrada(i);
      correcto=enviar(pdu, entrada.getDireccion());
      i++;
  ...
}
```

- **private void setTimerTask()**

Método que fija las operaciones para encuestar a los clientes. La forma de actuación se basa en un proceso cíclico: en primer lugar se pregunta a todos los clientes si aún están conectados a la aplicación, para más tarde (tras un tiempo prudencial) comprobar cuáles no han contestado, siendo por tanto expulsados de la aplicación.

```
private void setTimerTask() {
  timerRevision = new Timer();
  TimerTask taskRevision = new TimerTask() {
    public void run() {
      revision();  //envía peticiones
    }
  };
  timerRevision.scheduleAtFixedRate(taskRevision, (long) tiempo_revisiones,
                    (long) (2*tiempo_revisiones));

  timerComprobacion = new Timer();
  TimerTask taskComprobacion = new TimerTask() {
    public void run() {
      comprobacion(); //coteja las fechas de respuesta
    }
  };
  timerRevision.scheduleAtFixedRate(taskComprobacion,
                    (long) (2*tiempo_revisiones)+(UDPtimeout),
                    (long) (2*tiempo_revisiones)+(UDPtimeout));
  }
```

- **public void recibido(EventoPaqueteUDP paquete)**

Implementación de la función declarada en la interfaz *PaqueteUDPListener*, cuyo objetivo es indicar la recepción de una PDU_UDP. La función se limita a añadir esta PDU a la lista de peticiones.

- **private synchronized void anadePeticion(EventoPaqueteUDP paquete)**

Añade una PDU a la lista de peticiones, de la cual el *thread* irá cogiendo y procesando.

- **private synchronized void esperar()**

Función que obliga al *thread* a pasar a un estado inactivo, si es que no han llegado nuevas PDUs.

- **private synchronized Object getPeticion()**

Función que devuelve la primera petición de la lista de PDUs.

- **public void run()**

En esta función se lleva a cabo todo el proceso de las PDU. Para ello, se clasifican por tipo y luego por subtipo, con el objetivo de dar la respuesta conveniente para cada caso y conocer con exactitud el número de campos y tipo de los mismos que llegan con la PDU.

```
public void run(){
  EventoPaqueteUDP paquete = null;
  PDU_UDP pdu = new PDU_UDP();

  while (getfinConexiones() == false)
  {
    try {
      esperar();
      paquete = (EventoPaqueteUDP)getPeticion();

      switch (paquete.suceso) {
          case 0: //recibido correctamente
            switch (paquete.pdu.getTipo()) {
            …
            }
            break;
          case 1: //Timeout
            …
            break;
          case 2: //Error I/O
            …
            break;
        }
        conexiones.actualizarAcceso_direccion(paquete.addr);
```

```
        }
          catch (NullPointerException e) { //se ha recibido una PDU con un numero
            //incorrecto de parámetros
           ...
          }
          catch (NumberFormatException e) { //algun parametro en formato numerico
    //es incorrecto
           ...
          }
        }
      }
```

- **public synchronized void close()**

  Función encargada de despedirse de todos los clientes UDP y cancelar los *timers* de encuesta.

- **La clase ServidorRed**

Esta clase tiene dos funciones fundamentales, por un lado se encarga de inicializar las conexiones (tanto UDP como TCP) y por otro se ocupa de la gestión de los mensajes enviados a las conexiones TCP.

Hay que destacar que la conexión Actuadora exige un nivel adicional de identificación, con el fin de evitar apropiaciones indebidas. Esta identificación consiste en el envío de una PDU conteniendo el *login*, a partir de la cual, junto con la dirección IP, es posible asegurar si es el mismo que facilitó los parámetros en la conexión gestora.

- **class ServidorRed extends Thread implements PaqueteTCPListener, MotorMovidoListener)**

  Definición de la clase. Hereda de Thread, con el objetivo de mantener un hilo de ejecución independiente del programa principal; además implementa las interfaces *PaqueteTCPListener* (para ser avisado cuando llegue un nuevo paquete TCP) y MotorMovidoListener (de esta manera podrá saber cuándo se ha producido un movimiento en los motores, y así poder avisar a los usuarios de esta situación).

- **public ServidorRed(TablaUsuarios tablaUsuarios, TablaConexiones tablaConexiones, int TCPtimeout, int tiempo_conexionActuador, int tiempo_gestion,int puerto_gestion, int UDPtimeout, int tiempo_revisionesUDP, OpcionesMotores tablaMotores, ServidorListener l) throws IOException**

Único constructor de la clase. Requiere de gran cantidad de parámetros, ya que es el encargado de organizar todas las conexiones (UDP y TCP).

En este constructor se asignan los valores a las variables miembro y se inicializan los objetos de control de conexiones UDP y TCP gestora (ya que la actuadora se debe habilitar previa identificación), además de iniciar el propio *thread* en espera de PDU_TCP.

Las conexiones TCP se agrupan en un array de objetos PaqueteTCPServidor.

```
…
tcp[gestor] = new PaqueteTCPServidor(puerto_gestion,
                                     gestor,
                                     TCPtimeout,
                                     timeout_gestor,
                                     this.getPuerto(ref_puerto_udp),
                                     this);
udp = new PaqueteUDPServidor(puerto_gestion,
                                    UDPtimeout,
                                    TCPtimeout,
                                    timeout_actuador,
                                    tiempo_revisionesUDP,
                                    tablaUsuarios,
                                    conexiones,
                                    tcp,
                                    this,
                                    eventoServidor);
```

- **public void recibido(EventoPaqueteTCP paquete)**

Implementación de la función declarada en la interfaz *PaqueteTCPListener*, cuyo objetivo es indicar la recepción de una PDU_TCP. La función se limita a añadir esta PDU a la lista de peticiones.

- **private synchronized void anadePeticion(EventoPaqueteTCP paquete)**

Añade una PDU a la lista de peticiones, de la cual el *thread* irá cogiendo y procesando.

- **private synchronized void esperar()**

Función que obliga al *thread* a pasar a un estado inactivo, si es que no han llegado nuevas PDUs.

- **private synchronized Object getPeticion()**

Función que devuelve la primera petición de la lista de PDUs.

- **public void run()**

En esta función se lleva a cabo todo el proceso de las PDU. Para ello, se clasifican por tipo y luego por subtipo, con el objetivo de dar la respuesta conveniente para cada caso y conocer con exactitud el número de campos y tipo de los mismos que llegan con la PDU.

Aquellas PDUs destinadas en exclusiva a la conexión gestora, o a la actuadora, serán tratadas en funciones específicas para cada caso.

```
public void run(){
  EventoPaqueteTCP paquete = null;
  PDU_TCP pdu = new PDU_TCP();

  while (getfinConexiones() == false)
  {
    try {
      esperar();
      paquete = (EventoPaqueteTCP)getPeticion();

      switch (paquete.suceso) {
         case 0: //recibido correctamente
           switch (paquete.pdu.getTipo()) {
          …
           default:       //paquetes concretos según servicio
             switch (paquete.id) {
```

```
                case gestor:
                  mensajesGestor(paquete.pdu);
                  break;
                case actuador:
                  mensajesActuador(paquete.pdu);
                  break;
               }
               break;
             }
              break;
            case 1: //Timeout
              …
              break;
            case 2: //Error I/O
             …
              break;
          }
           conexiones.actualizarAcceso_direccion(paquete.addr);       }
        catch (NullPointerException e) { //se ha recibido una PDU con un numero
           //incorrecto de parámetros
          ...
        }
        catch (NumberFormatException e) { //algun parametro en formato numerico
  //es incorrecto
          ...
        }
       }
      }
```

- **protected void permitirActuador(boolean permitir)**

Función utilizada para bloquear/desbloquear el recurso de actuador, de forma que, estando activo el servidor de red, el control de los motores se haga local/remoto. Para bloquear actuadores, cambia el modo del posible usuario actuador activo y crea un actuador ficticio, de tal forma que de cara a la tabla de conexiones existe un actuador, y por tanto, ningún otro puede acceder a sus privilegios. La figura 6-15 presenta la secuencia de pasos adoptada ante el cambio de control de los motores entre local y remoto.

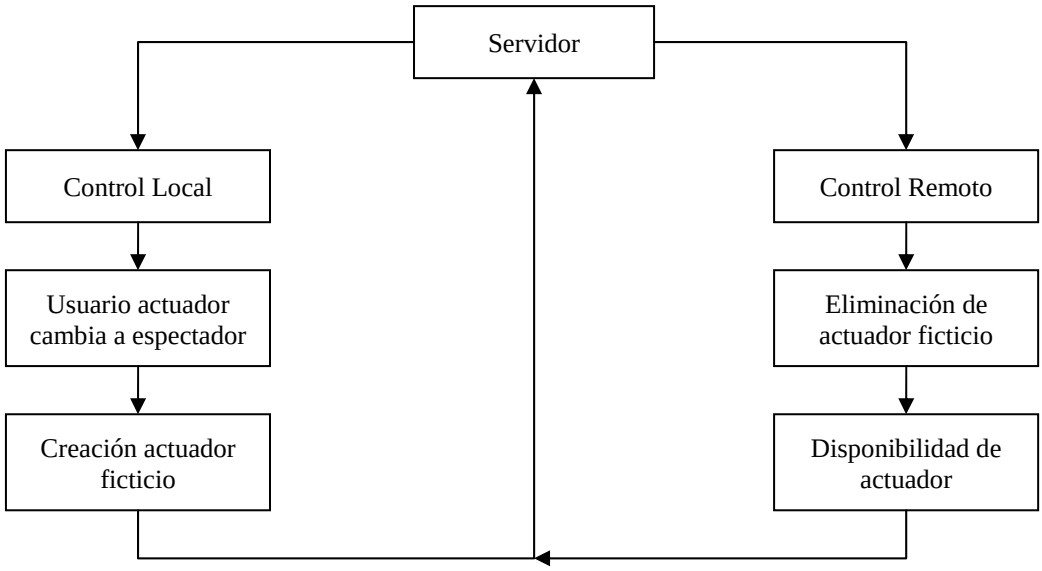


Figura 6-15 : Esquema de acciones ante el control local/remoto de los motores

- **private synchronized entradaConexion desconectarTCP(int id, PDU_TCP motivo)**

Método empleado para desconectar a un usuario de sesión TCP, que viene a ser el actuador (ya que el gestor nunca se añade a la tabla de conexiones). En primer lugar le envía el mensaje de despedida especificado en el parámetro *motivo*, para luego eliminarlo de la tabla, y avisar a todos los usuarios espectadores de la disponibilidad de la conexión actuadora.

Si el motivo de la desconexión es porque la identificación del actuador no es correcta, simplemente se desconecta al usuario, y se reinicia el *socket* a la espera del verdadero usuario que se identificó en la conexión gestora.

- **private void mensajesGestor(PDU_TCP pdu_recibida)**

Método empleado para gestionar las PDUs que sólo se emplean en la conexión gestora, como son las utilizadas en el proceso de identificación.

Esta función es la encargada de decidir, en base al contenido de las tablas de usuarios, si el cliente es apto para acceder al servicio, y con qué privilegios, enviando las oportunas PDUs confirmando ó denegando el acceso.

- **private void mensajesActuador(PDU_TCP pdu_recibida)**

Método que gestiona las PDUs específicas de la sesión actuadora. El requisito para llevar a cabo alguna acción es el de haber sido autenticado con anterioridad, o pretender hacerlo cuando se entra; en caso contrario, se envía una PDU de error y termina la sesión.

- **public PDU_TCP getConfiguracionMotor()**

Obtiene la configuración de los motores y la empaqueta en una PDU

```
public PDU_TCP getConfiguracionMotor(){
  PDU_TCP pdu = new PDU_TCP();

  pdu.setTipo(2, 0); //enviamos la configuración de los motores

  pdu.addDato( (int) (360.0 / tablaMotores.getAnguloPaso(1)));
  pdu.addDato( (int) (360.0 / tablaMotores.getAnguloPaso(2)));
  pdu.addDato(tablaMotores.getAnguloActual(1)); //ANGULO 1 (incial)
  pdu.addDato(tablaMotores.getAnguloActual(2)); //ANGULO 2 (incial)
  pdu.addDato(tablaMotores.getVelocidadMaxima(1));
  pdu.addDato(tablaMotores.getVelocidadMaxima(2));
  pdu.addDato(tablaMotores.getVelocidad(1));
  pdu.addDato(tablaMotores.getVelocidad(2));
  pdu.addDato(tablaMotores.getFlag(1)); //flag 1
  if (tablaMotores.getFlag(1) == 2) {
    //define un ángulo máximo (en número de pasos)
    pdu.addDato(tablaMotores.getAnguloMaximo(1));
  }
  pdu.addDato(tablaMotores.getFlag(2)); //flag 1
  if (tablaMotores.getFlag(2) == 2) {
    //define un ángulo máximo (en número de pasos)
    pdu.addDato(tablaMotores.getAnguloMaximo(2));
  }
  return pdu;
}
```

- **public void motorMovido(MotorMovidoEvent ev)**

Implementación del método declarado en la interfaz *MotorMovidoListener*. Esta función es utilizada para avisar al servidor de esta herramienta, del movimiento de algún motor o del fallo en la operación. En caso de producirse algún error, sólo se informa del mismo al actuador, ya que es el que ha provocado el movimiento, siendo transparente para los espectadores (para ellos simplemente hay ausencia de movimiento)

```
public void motorMovido(MotorMovidoEvent ev){
  PDU_UDP pduUDP = new PDU_UDP();
  PDU_TCP pduTCP = new PDU_TCP();

  if ((ev.tipo == 1) || (ev.tipo == 2)){
    if (tcp[actuador] != null){//si hay un actuador
      pduTCP.setTipo(2,1);
      pduTCP.addDato(tablaMotores.getAnguloActual(1));//paso 1
      pduTCP.addDato(tablaMotores.getAnguloActual(2));//paso 2
      pduTCP.addDato(tablaMotores.getVelocidad(1)); //velocidad 1
      pduTCP.addDato(tablaMotores.getVelocidad(2)); //velocidad 2
      tcp[actuador].enviar(pduTCP); //avisamos al actuador
    }
    //avisamos a los espectadores
    pduUDP.setTipo(2,1);
    pduUDP.addDato(tablaMotores.getAnguloActual(1));//paso 1
    pduUDP.addDato(tablaMotores.getAnguloActual(2));//paso 2
    pduUDP.addDato(tablaMotores.getVelocidad(1)); //velocidad 1
    pduUDP.addDato(tablaMotores.getVelocidad(2)); //velocidad 2
    udp.multicast(pduUDP);
  }
  else{//ha habido un error al acceder al dispositivo
     if (tcp[actuador] != null){ //si hay un actuador
       pduTCP.setTipo(2, 3);
       tcp[actuador].enviar(pduTCP); //avisamos al actuador
     }
  }
}
```

- **Clases de Gestión de Usuarios**

Estas clases tienen como misión llevar el control de una tabla en la que se reflejen todos los usuarios registrados, detallando sus datos y permisos.

- **La clase entradaUsuario**

Esta clase sirve de soporte para cada una de las entradas de la tabla de usuarios. Su utilidad se reduce a almacenar los valores para cada entrada, y proveer de métodos para el acceso a los mismos.

Los campos almacenados son los de: login, password, nombre, permisos (tipo) y último acceso.

- **class entradaUsuario implements Serializable**

Definición de la clase. Implementa la interfaz *Serializable*, la cual permite convertir los datos de la clase en un flujo de bytes, y con ello poder almacenarla en disco.

- **public entradaUsuario()**

  Constructor sin parámetros de la clase. Asigna un valor conocido a cada uno de los campos.

- **public entradaUsuario(String nombre, String password, short tipo)**

  Constructor con parámetros. Crea una entrada de usuarios con los valores dados por parámetros.

- **public entradaUsuario(entradaUsuario usuario)**

  Constructor de copia. Crea una nueva entrada de usuario, con los datos incluidos en la entrada indicada en los parámetros.

- **protected Date setUltimoAcceso()**

  Fija la fecha y hora del último acceso del usuario.

- **protected Date getUltimoAcceso()**

  Devuelve la fecha y hora del último acceso del usuario.

❑ La clase TablaUsuarios

Esta clase es la encargada de gestionar la tabla de usuarios, cuyas entradas son objetos del tipo *entradaUsuario*.

La implementación de la tabla se ha realizado utilizando una tabla *Hash*, por motivos de eficiencia en la búsqueda, siendo el *login* la clave de los campos, ya que no podrán existir dos usuarios con el mismo *login* (los demás campos sí pueden repetirse).

▪ class TablaUsuarios

Definición de la clase.

- **public TablaUsuarios()**

  Constructor sin parámetros. Crea una nueva tabla vacía.

- **public TablaUsuarios(String nombreArchivo)**

  Constructor con parámetros. Intenta construir una tabla de usuarios, a partir de la que debe encontrar en el archivo indicado por parámetros; si no encontrara este archivo, o éste no contiene información correcta, el constructor crearía una nueva tabla vacía.

- **protected synchronized boolean cargarTabla(String nombreArchivo)**

  Método encargado de recuperar la tabla de usuarios del archivo especificado.

```
protected synchronized boolean cargarTabla(String nombreArchivo){
  try {
    FileInputStream archivo = new FileInputStream(nombreArchivo);
    ObjectInputStream in = new ObjectInputStream(archivo);
    tabla = (HashMap)in.readObject();
    in.close();
  }
  catch (Exception e) {
    return false;
```

```
  }
  return true;
}
```

- **protected synchronized boolean guardarTabla(String nombreArchivo)**

Almacena la tabla de usuarios en el archivo especificado.

```
protected synchronized boolean guardarTabla(String nombreArchivo){
  try{
    FileOutputStream archivo = new FileOutputStream(nombreArchivo);
    ObjectOutputStream out = new ObjectOutputStream(archivo);
    out.writeObject(tabla);
    out.close();
    }
  catch (Exception e){
    return false;                    //NO se guardó correctamente
  }
  return true;                       //se guardó correctamente
}
```

- **protected synchronized boolean anadirUsuario(String login, String nombre, String password, short tipo)**

Función utilizada para añadir un usuario a la tabla. Para ello primero debe asegurarse de que el *login* especificado no está siendo utilizado actualmente, en cuyo caso, los datos del usuario serán añadidos a la tabla.

- **protected synchronized short getPermiso(String login, char[]password)**

Devuelve los permisos asignados al usuario indicado, a la vez que lo identifica, ya que el parámetro *password* es utilizado para asegurar la correspondencia con el login.

- **protected synchronized boolean eliminarUsuario(String login)**

Elimina la entrada de usuario correspondiente al *login* indicado.

- **protected synchronized entradaUsuario getEntrada(String login)**

Devuelve la entrada de usuario correspondiente al *login* indicado.

- **protected synchronized boolean setLogin(String oldlogin, String newlogin)**

Método utilizado para cambiar el *login* de un usuario. Para ello debe cumplir el requisito de que no haya otro usuario utilizando el mismo identificador.

```
protected synchronized boolean setLogin(String oldlogin, String
      newlogin){
  ...
  if ((tabla.containsKey(oldlogin) == true) &&
     (tabla.containsKey(newlogin) == false)){
    temp = (entradaUsuario) tabla.get(oldlogin);

    tabla.remove(oldlogin);
    tabla.put(newlogin, temp);
    ...
    return true;
  }
  else
    return false;
}
```

- **synchronized public void addUsuarioListener(UsuarioListener l)**

Añade un objeto a la lista de *listeners*; es decir, a la lista de objetos a los que avisar cuando se añade, modifica o elimina un usuario.

- **synchronized public void removeUsuarioListener(UsuarioListener l)**

Elimina un objeto de la lista de listeners.

- **protected synchronized void avisaTabla(short suceso, String login)**

Método empleado para avisar a los objetos incluidos en la lista de *listeners*.

➢ **Clases de Configuración**

El objetivo de estas clases es el de almacenar los datos referentes a la configuración de la herramienta y velar porque éstos sean coherentes.

- **La clase valoresMotores**

Clase de soporte para los valores de configuración de los motores. No implementa ningún tipo de método (salvo el constructor), ya que sus campos serán gestionados por la clase *OpcionesMotores*. Se ha implementado de esta manera para simplificar el almacenamiento de los datos.

Los valores almacenados para cada motor son:

- Posición actual
- Velocidad actual
- Velocidad máxima
- Angulo de paso
- Flag indicador de límites: 0: Multivueltas, 1:nºde vueltas, 2: angulo maximo
- Número de vueltas máximas (si especificado en el flag)
- Angulo máximo de giro (si especificado en el flag)
- Vuelta actual

Además, también almacena el tamaño máximo que el registro de movimientos puede tener.

- **class valoresMotores implements Serializable**

Definición de la clase. Implementa la interfaz *Serializable*, la cual permite convertir los datos de la clase en un flujo de bytes, y con ello poder almacenarla en disco.

- **public valoresMotores(int paso1,int velocidad1,double angulo_paso1, short flag1, int vueltas1, double angulo_maximo1, short velocidadMaxima1, int paso2, int velocidad2, double angulo_paso2, short flag2, int vueltas2, double angulo_maximo2, short velocidadMaxima2, int tamano_log)**

Único constructor de la clase. Asigna el valor a cada uno de los campos.

- **La clase OpcionesMotores**

Encargada del almacenamiento y gestión de la configuración de los motores, así como del mantenimiento del archivo de registro de movimientos.

El registro de movimientos es un archivo de texto en el que se almacena el movimiento de cada motor, junto con la velocidad a la que se realizó, además de la fecha/hora y el usuario que llevó a cabo el movimiento. Este fichero no puede crecer indefinidamente, por lo que debe establecerse un tamaño máximo.

Para implementar el archivo de registro se utilizan tres archivos temporales, cuyo tamaño corresponde con la mitad de la longitud asignada al archivo de registro definitivo. De esta manera, una vez se llene uno, se pasa al siguiente (siguiendo un orden de asignación cíclico). Una vez que la aplicación finalice, o se desee ver el registro, se cogen todas las entradas del último archivo en el que se ha escrito, las del archivo escrito justo antes que el actual y si aún quedara espacio, el número pertinente de entradas del tercer archivo, conformando el fichero de registro definitivo.

- **public class OpcionesMotores**

Definición de la clase.

- **public OpcionesMotores()**

Único constructor de la clase. En primer lugar trata de recuperar los valores almacenados en el archivo de configuración; en caso de no encontrarlo, o que sus datos no sean válidos, se cargarán valores por defecto.

```
if (cargarTabla("motores.inf") == false){
    tablaValores = new valoresMotores(0, 1, 1, (short) 0, 0, 360.0,
                                      (short)127,0, 1, 1, (short) 0, 0,
                                      360.0, (short)127, 1);
    …
}
```

Además, también se ocupa de inicializar los archivos temporales, en los que se va almacenando información durante la ejecución del programa, cargando en ellos el archivo de registro guardado previamente, teniendo siempre en cuenta las

restricciones actuales de tamaño del registro, ya que pueden no ser iguales que las impuestas cuando se creó dicho archivo.

```
archivos = new RandomAccessFile[3];

        archivos[0] = new RandomAccessFile(archivoTemporal(0), "rw");
        archivos[1] = null;
        archivos[2] = null;
```

En el caso de que se detecte la presencia de un archivo de registro previo, de éste se obtendrá la posición y velocidad del último movimiento realizado para cada uno de los motores, siendo ahora estos datos sus valores iniciales.

```
tablaValores.paso1 = (int)(getUltimaPosicion((short)1)/
                        tablaValores.angulo_paso1);
tablaValores.velocidad1 = obtenerVelocidad(getUltimaVelocidad((short)1),
        (int)(360.0/tablaValores.angulo_paso1));
```

- **public synchronized boolean guardarTabla()**

Función encargada de guardar la configuración de los motores.

- **private synchronized boolean cargarTabla(String nombreArchivo)**

Función encargada de cargar los valores de la configuración, en el objeto que representa a la tabla.

- **protected synchronized void movido(double angulo_final1, int velocidad1, double angulo_final2, int velocidad2, String usuario)**

Función invocada tras la ejecución de un movimiento. Añade una nueva entrada al registro de movimientos.

```
cadena = new String(fecha.format(new Date()) + "\t\t"
            +  "Posicion Motor 1 : " + cadenaNumero(angulo_final1,"º")
            +  "Velocidad Motor 1 : " +
            cadenaNumero(cuentaRevoluciones(velocidad1,
            (int)(360/tablaValores.angulo_paso1)),"rpm") +
            "Posicion Motor 2 : " + cadenaNumero(angulo_final2,"º")
            + "Velocidad Motor 2 : " +
            cadenaNumero(cuentaRevoluciones(velocidad2,
```

```
                (int)(360/tablaValores.angulo_paso2)),"rpm") +
        " Usuario : " + usuario + "\n");
```

- **protected synchronized double getUltimaVelocidad(short ID)**

Obtiene del archivo de registro previamente almacenado, el valor de la velocidad con la que se ejecutó el último movimiento del motor indicado.

- **protected synchronized double getUltimaPosicion(short ID)**

Obtiene del archivo de registro previamente almacenado, la posición en la que quedó el motor indicado por parámetros, tras llevar a cabo el último movimiento.

- **protected synchronized boolean crearLogFinal()**

Método responsable de crear el fichero de registro (Log.txt) a partir de las entradas almacenadas en los archivos temporales. Su esquema de funcionamiento es el mostrado en la figura 6-16.

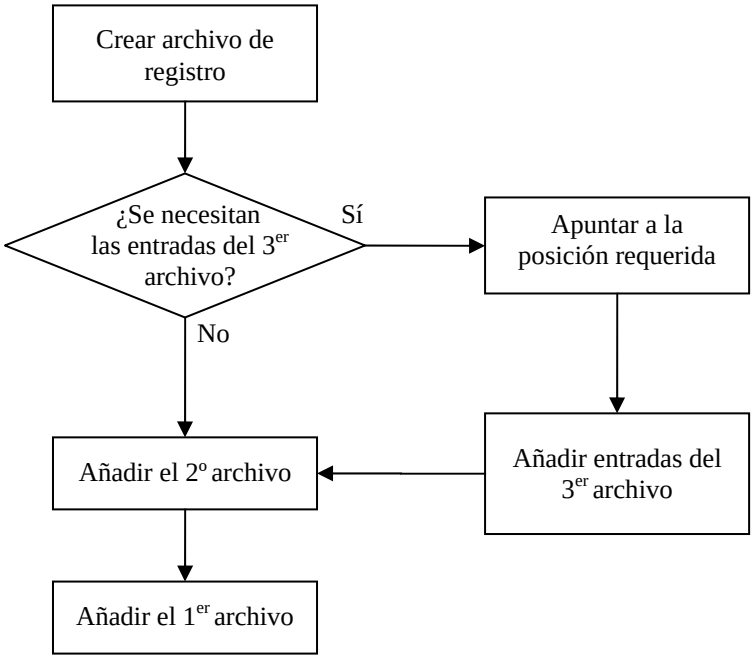


Figura 6-16 : Generación del archivo de registro de movimientos, a partir de la recolección de entradas

- **public synchronized void close()**

- Función invocada cuando la aplicación *Servidor* está a punto de cerrarse. Su función es la de construir el archivo de registro y eliminar los archivos temporales utilizados.

- **La clase valoresRed**

Clase de soporte para los valores de configuración de la red. No implementa ningún tipo de método (salvo el constructor), ya que sus campos serán gestionados por la clase *OpcionesRed*. Se ha implementado de esta manera para simplificar el almacenamiento de los datos.

Los valores almacenados para cada motor son:

- Número máximo permitido de conexiones
- Puerto de la conexión gestora
- Timeout de la conexión gestora
- Timeout de la conexión actuadora
- Timeout de la conexión espectadora
- Tiempo de la sesión gestora
- Tiempo de la sesión actuadora
- Ruta de localización del applet

- **class valoresRed implements Serializable**

Definición de la clase. Implementa la interfaz *Serializable*, la cual permite convertir los datos de la clase en un flujo de bytes y con ello poder almacenarla en disco.

- **public valoresRed(int maximoConexiones, int puertoGestion, int timeoutGestion, int timeoutActuador, int timeoutEspectador, long tiempoConexionGestion, long tiempoConexionActuador)**

Único constructor de la clase. Asigna el valor a cada uno de los campos.

- **La clase OpcionesRed**

Encargada del almacenamiento y gestión de la configuración de la red.

- **public class OpcionesRed**

Definición de la clase.

- **public OpcionesRed()**

Único constructor de la clase. En primer lugar trata de recuperar los valores almacenados en el archivo de configuración; en caso de no encontrarlo, o que sus datos no sean válidos, se cargarán valores por defecto.

```
public OpcionesRed(){
  if (cargarTabla("red.inf") == false) //no se pudo cargar
    tablaValores = new valoresRed(5,8000,5000,5000,60000,30,200);
    …
  }
```

- **public synchronized boolean guardarTabla()**

Función encargada de guardar la configuración de la red.

- **private synchronized boolean cargarTabla(String nombreArchivo)**

Función encargada de cargar los valores de la configuración en el objeto que representa a la tabla.

- **public synchronized boolean copiaParaApplet()**

Guarda una copia del archivo de configuración de la red en el directorio en el que reside el applet (sólo aquellos parámetros que necesita), de forma que pueda

acceder al mismo, y así conocer los parámetros con los que se debe conectar al servidor.

- **Clases de Gestión de USB**

La gestión USB en el servidor se realiza mediante dos clases Java y una librería dinámica (.dll) que implementa, en código nativo, un pipe de comunicaciones con la aplicación gestora de USB.

- **La clase PipeC**

Esta clase declara los métodos a implementar por el código nativo (ver capítulo conceptos generales de Java, *código nativo*).

La información enviada al gestor USB se encapsula en un array de caracteres, cuya estructura es:

array[**0]** : TIPO
array[**1**] : sentido1/sentido2
array[**2**] : velocidad1
array[**3**] : pasos1 (alto)
array[**4**] : pasos1 (bajo)
array[**5**] : velocidad2
array[**6**] : pasos2 (alto)
array[**7**] : pasos2 (bajo)
array[**8**] : tiempo a esperar (alto)
array[**9**] : tiempo a esperar (bajo)

- **public native boolean escribir(char[] bufferOut)**

Declaración de la función que envía datos al pipe de comunicaciones.

- **public native boolean abrir()**

Declaración de la función que abre la aplicación que gestiona el USB.

- **public native void cerrar();**

Declaración de la función que cierra la aplicación que gestiona el USB.

- **Implementación en código nativo**

A continuación se describen las funciones definidas en C.

- **bool iniciaPipe(HANDLE *hPipe, HANDLE *hEvento, OVERLAPPED *hOverlapped)**

Función encargada de establecer la comunicación con el pipe creado por el gestor de USB.

```
bool iniciaPipe(HANDLE *hPipe, HANDLE *hEvento, OVERLAPPED *hOverlapped){
        LPTSTR lpszPipename = "\\\\.\\pipe\\pfc_usb_pipe";

        *hPipe=CreateFile
                (lpszPipename,
                GENERIC_READ|GENERIC_WRITE,  //lectura y escritura
                0,
                (LPSECURITY_ATTRIBUTES)NULL,
                OPEN_EXISTING,
                FILE_FLAG_OVERLAPPED,
                NULL);
        …
 }
```

- **long calculaTiempo(char alto, char bajo)**

Esta función calcula el tiempo a esperar, a partir de dos bytes en los que se almacena dicho valor (parte alta y baja) y añade cinco segundos de margen.

- **bool leePipe(HANDLE hPipe, char *buffer, DWORD bufferSize, HANDLE hEvento, LPOVERLAPPED hOverlapped, long tiempo)**

Función encargada de leer los datos del pipe. Espera por los datos durante el tiempo especificado por el parámetro *tiempo*; si transcurrido el mismo aún no se han recibido datos, se avisa del error.

```
bool leePipe(HANDLE hPipe, char *buffer, DWORD bufferSize, HANDLE hEvento,
    LPOVERLAPPED hOverlapped, long tiempo)
```

```
{
        …
        ReadFile(  //leemos del pipe
            hPipe,
            buffer,
            bufferSize,
            &bytesEscritos,
            (LPOVERLAPPED) hOverlapped);

        Result = WaitForSingleObject (hEvento, tiempo);
          //esperamos por los datos

        switch (Result)
        {
        case WAIT_OBJECT_0:
                {
                //recibido correctamente
                break;
                }

        case WAIT_TIMEOUT:
                {
                … //llegan tarde
                return false;
                break;
        default:
                return false;
                break;
                }
        }
        …
        return true;
}
```

- **JNIEXPORT jboolean JNICALL Java_PipeC_escribir(JNIEnv *env, jobject obj, jcharArray arr)**

Se trata de la implementación del método *escribir*. Su objetivo es el de enviar al gestor de USB el *array* indicado en los parámetros y esperar por su respuesta, la cual será devuelta en una variable miembro de la clase *PipeC*. De igual forma,

cualquier incidencia será señalada con un código de error en el primer byte del array (TIPO):

1 : Recepción correcta

2 : los datos devueltos no coinciden exactamente con los enviados

3 : error al iniciar el pipe

4 : error en la aplicación USB

5 : error al acceder al dispositivo (ha sido desconectado o no responde)

```
JNIEXPORT jboolean JNICALL Java_PipeC_escribir(JNIEnv *env, jobject obj,
        jcharArray arr) {
        ...
        if(iniciaPipe(&hPipe,&hEvento,&hOverlapped) == false)
        {
                datoJava[0]=3;//error al iniciar el pipe
                conseguido = false;
        }
        else
        {
        ...
        int tiempo = calculaTiempo(envia[LongitudTrama-2],
                envia[LongitudTrama-1]);

        WriteFile(hPipe,            //enviamos los datos al pipe
                  envia,
                  len,
                  &bytesEscritos,
                  (LPOVERLAPPED) &hOverlapped);


        if (GetLastError() == ERROR_IO_PENDING)
                {
                 Result = WaitForSingleObject (hEvento, tiempo);//esperamos
                }
                switch (Result)
                {
                case WAIT_OBJECT_0: //devolvió valores en el tiempo estipulado
```

```
                    {
                    ...
                    //enviamos al servidor los valores devueltos
                    ...
                    }
                case WAIT_TIMEOUT: //no respondió a tiempo
                    {
                    ... // enviamos la notificación de error al servidor
                    }
                }
                ...
        }
```

- **JNIEXPORT jboolean JNICALL Java_PipeC_abrir(JNIEnv *env, jobject obj)**

Definición en código nativo de la función abrir().

```
CreateProcess( NULL,  TEXT("pipe.exe"),  NULL,  NULL,  FALSE,  0,  NULL,
    NULL,  &si,  &pi);
```

- **JNIEXPORT void JNICALL Java_PipeC_cerrar(JNIEnv *env, jobject obj)**

Definición en código nativo de la función cerrar().

```
JNIEXPORT void JNICALL Java_PipeC_cerrar(JNIEnv *env, jobject obj){
        TerminateProcess(proceso,0);
        CloseHandle(proceso);
        return;
}
```

- **La clase ToUSB**

Encargada de gestionar las peticiones enviadas al dispositivo USB y sus respuestas. Su funcionamiento se basa en la recepción de eventos de movimiento, ya sea desde el panel de la interfaz gráfica (cuando el control es local al servidor) o desde el servidor de red (cuando el control es remoto), a partir de los cuales determina cuántos pasos deben ser dados por cada motor. Estos datos son empaquetados de la forma convenida y pasados como parámetros al método *escribir* de la clase *PipeC*.

- **public class ToUSB extends Thread implements PanelMovimientoListener, MoverMotorRedListener**

  Definición de la clase. Implementa dos interfaces de eventos; el primero se utiliza para obtener eventos del panel de motores del servidor (si está activado el control local), mientras que el segundo implica como fuente de eventos al servidor de red, es decir, proceden de peticiones de un cliente remoto.

  Además, hereda de la clase Thread, ya que al tratarse de una operación bloqueante (pues debe esperar a que el pipe responda), debe realizarse en un hilo de ejecución separado.

  - **public ToUSB(OpcionesMotores opcionesMotores)**

  Constructor de la clase. Inicializa el pipe de comunicaciones (llamando al método *abrir* de *PipeC*) y obtiene los datos necesarios (para realizar los cálculos de pasos) del objeto pasado por parámetros.

  - **private synchronized void anadePeticion(char[] mensaje)**

  Añade a la lista de peticiones de movimientos, un nuevo array de solicitud de movimiento, para ser enviado en el método *run()*.

  - **private synchronized void esperar()**

  Método empleado para esperar hasta que se produzca una nueva petición de movimiento.

  - **private synchronized Object getPeticion()**

  Devuelve la petición de movimiento más antigua de la lista.

  - **public void run()**

  Sobrecarga del método principal de la clase *Thread*. En él se realiza la comunicación con el *pipe*.

```
public void run(){
    ...
    while (tipo == 1) { //datos
```

```
        esperar();

      info = (char[])getPeticion();
      if (pipe.escribir(info) == true){
        recibido(info);
      }
      else{ //problemas para acceder al pipe
        if (pipe.datoRecibido[0] != 5){
          ...  //reiniciamos el gestor
        }
        else{
          ... //dispositivo desconectado
        }
      }
    }
  ...
  }
```

- **public void close()**

Función utilizada para cerrar la comunicación con el pipe. Para ello se cierra la aplicación de gestión de USB.

- **private void recibido(char[] info)**

Función invocada tras recibir la respuesta (correcta) del pipe, ya que esta función se encarga de actualizar la configuración de los motores, indicando cuántos pasos se han movido y a qué velocidad.

- **public void moverMotor(PanelMovimientoEvent ev)**

Definición del método declarado en la interfaz *PanelMovimientoListener*, utilizado para saber cuándo el panel de motores del servidor desea mover los motores.

Invoca a las funciones de cálculo de pasos (y sentido); estos datos, junto a la velocidad de los motores y el tiempo estimado de ejecución, son enviados a la aplicación gestora de USB.

- **public void moverMotorRed(MoverMotorRedEvent ev)**

Definición del método declarado en la interfaz *MoverMotorRedListener*, utilizado para saber cuándo un cliente remoto (se supone con los privilegios oportunos) desea mover los motores.

Invoca a las funciones de cálculo de pasos (y sentido); estos datos, junto a la velocidad de los motores y el tiempo estimado de ejecución, añadidos a la lista de peticiones de movimiento.

- **private int calculaPasos(short IDMotor,double anguloFinal)**

Método encargado de determinar el camino más corto para alcanzar la posición requerida por el usuario, desde la posición actual; esta operación requiere conocer, además, el ángulo que supone cada paso del motor. La figura 6-17 muestra el esquema de funcionamiento de esta función.

- **private synchronized void setInfoMotor(short IDMotor,double anguloFinal, short velocidad)**

Método que asigna los valores a los campos del array que será enviado al gestor USB (a partir del método *calculaPasos*), excepto los referentes a la temporización.

- **synchronized public void addMotorMovidoListener (MotorMovidoListener l)**

Añade un objeto a la lista de *listeners*; es decir, a la lista de objetos a los que avisar cuando se reciba una respuesta del pipe (o un error).

- **synchronized public void removeMotorMovidoListener (MotorMovidoListener l)**

Elimina el objeto indicado de la lista de *listeners*.

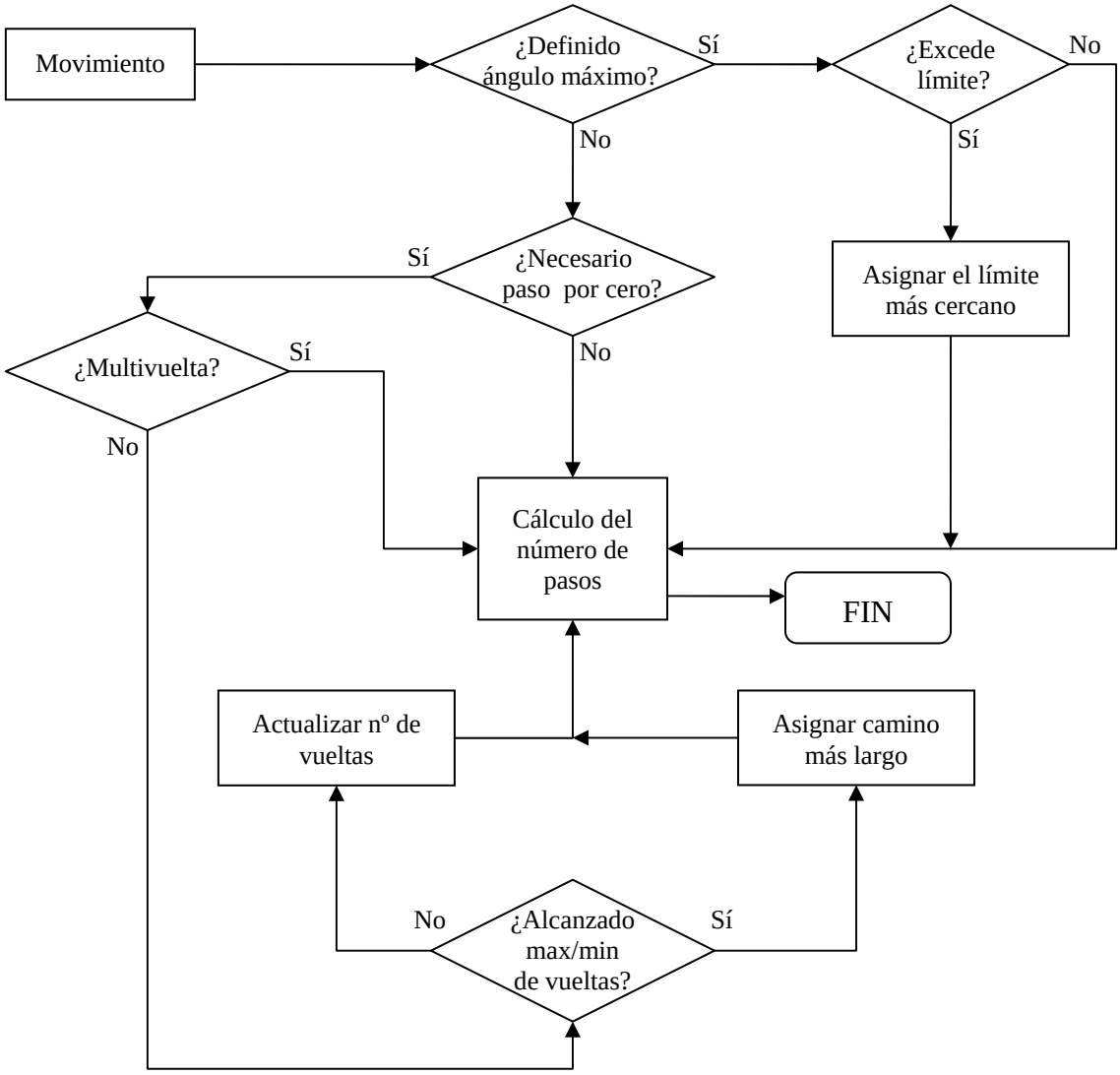


Figura 6-17 : Esquema de cálculo del número de pasos

- **protected synchronized void avisaMovido(int tipo)**

  Avisa a todos los objetos registrados en la lista de *listeners* de la llegada de una respuesta procedente del pipe, o un error en el mismo,

➢ **Clases de la Interfaz Gráfica**

El conjunto de estas clases tienen como objetivo mantener un medio de interacción con el usuario del servidor lo más sencillo y atractivo posible; de esta forma se facilitará la adaptación del usuario al entorno y con ello, un mayor aprovechamiento de la herramienta realizada.

- **La clase PanelLoginServidor**

Clase encargada de crear un panel que pide la identificación de usuario. Para ello ofrece dos entradas de texto, una para introducir el login, y otra para el password; esta última cuenta con la peculiaridad de que oculta la contraseña escrita, sustituyendo los caracteres introducidos por asteriscos (*). La figura 6-18 muestra el aspecto de este panel.

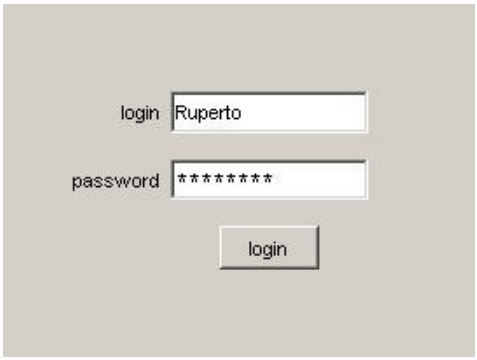


Figura 6-18 : Panel de identificación en el servidor

Dada la responsabilidad que supone permitir el acceso a un usuario a todos los recursos de la herramienta, sólo se permite el acceso a aquellos usuarios que cuenten con los permisos de administrador.

- **public class PanelLoginServidor extends JPanel**

Definición de la clase. Hereda los métodos de JPanel, utilizados para albergar los elementos a representar en el panel.

- **public PanelLoginServidor(TablaUsuarios tusuarios, TablaConexiones tconexiones)**

Constructor de la clase. Se utiliza para inicializar los elementos utilizados en el panel, así como indicar las acciones a llevar a cabo tanto si la identificación es correcta como si no, una vez sea accionado el botón de identificación *"login"*.

La figura 6-19 muestra el mensaje lanzado cuando la identificación es correcta.

Figura 6-19 : Mensaje de identificación correcta

```
public PanelLoginServidor(TablaUsuarios tusuarios, TablaConexiones
        tconexiones){
        ...
    boton.addActionListener(new ActionListener(){ //botón de login
        public void actionPerformed(ActionEvent ev){
            if (((JButton) ev.getSource()).getActionCommand().equals("login")) {
                switch (usuarios.getPermiso(alias.getText(), //comprueba permisos
                                password.getPassword())){
                case TablaUsuarios.admin:
                        JOptionPane.showMessageDialog(foo(),
                            "Ha sido identificado correctamente");
                        ...
                    break;
                case TablaUsuarios.actuador:
                case TablaUsuarios.udp:
                        JOptionPane.showMessageDialog(foo(),
                            "No tiene permisos de Administrador");
                        ...
                    break;
                default:
                        JOptionPane.showMessageDialog(foo(),
                            "La información de login no es correcta");
                    break;
                }
    }
```

Pero también puede darse el caso de que una vez identificado, el usuario decida abandonar la sesión sin haber cambiado de panel. Es por ello que tras la identificación, el botón de *login* cambia por el de *logout*, el cual, una vez pulsado, cierra la sesión de administrador abierta (previa confirmación mediante un diálogo). Esta situación es la representada en la figura 6-20.

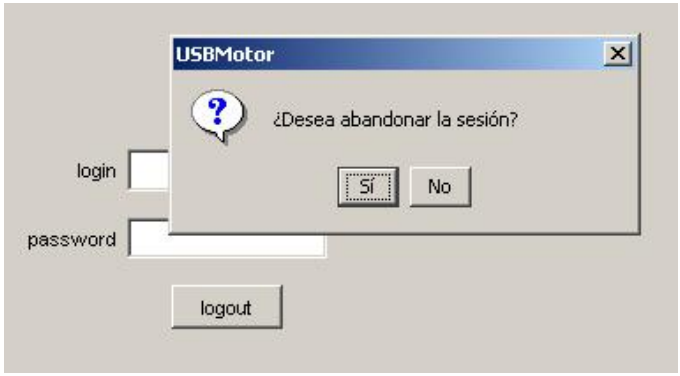


Figura 6-20 : Mensaje de abandono cierre de sesión administradora

- **public void addPanelLoginListener(PanelLoginListener listener)**

Añade un objeto a la lista de *listeners*; es decir, a la lista de objetos a los que avisar cuando un usuario se identifica correctamente como administrador.

- **public void removePanelLoginListener(PanelLoginListener listener)**

Elimina el objeto indicado por parámetros de la lista de *listeners*.

- **void avisaLogin(String login,short permiso)**

Método utilizado para avisar a los objetos inscritos en la lista de listeners, una vez se ha identificado un usuario administrador.

❑ **La clase PanelOpciones**

Se trata de un simple contenedor de paneles de opciones. Muestra un panel dividido en dos partes: a la izquierda será colocado el panel de configuración pertinente, mientras que a la derecha se muestran los botones que activan esos paneles.

▪ **public class PanelOpciones extends JSplitPane**

Definición de la clase. Hereda de *JSPlitPane*, que es el tipo de panel que permite una división en su interior.

- **public PanelOpciones(OpcionesMotores valoresMotores, OpcionesRed valoresRed, boolean conectadoAred)**

Constructor de la clase. Inicializa la división del panel y coloca los motores, además de asignar las acciones a los mismos. El parámetro *conectadoAred* es

utilizado para indicar a los diferentes paneles de configuración si el servidor está a la escucha de conexiones, ya que en este caso no se podrán alterar ciertos parámetros.

```
public PanelOpciones(OpcionesMotores valoresMotores,
    OpcionesRed valoresRed, boolean conectadoAred){
...
motoresButton.addActionListener(new ActionListener() {//conf. motores
    public void actionPerformed(ActionEvent ev) {
      if (! (getLeftComponent() instanceof PanelOpcionesMotor)){
        setLeftComponent(new PanelOpcionesMotor
                    (tablaMotores, conectado_red));
        setDividerLocation(420);
      }
    }
  });
  redButton.addActionListener(new ActionListener() { //cong. de red
    public void actionPerformed(ActionEvent ev) {
      if (! (getLeftComponent() instanceof PanelOpcionesRed)){
        setLeftComponent(new PanelOpcionesRed(tablaRed, conectado_red));
        setDividerLocation(420);
      }
    }
  });
}
```

- **La clase PanelOpcionesMotor**

Esta clase presenta el panel de opciones asociadas a los motores. En él es posible definir los parámetros de cada motor, además de indicar el tamaño máximo del archivo de registro en el que se almacenarán los datos de los movimientos producidos y acceder al mismo. La figura 6-21 muestra el aspecto de este panel dentro del panel de opciones general.

Los botones de *aceptar* y *cancelar* sólo estarán activos si se produce alguna modificación en los elementos del panel, con respecto a los datos almacenados en la configuración

Las opciones que ofrece este panel, para cada motor, son:

- Angulo de cada paso
- Escoger el tipo de restricción al movimiento

- Multivueltas : Sin límites
- Máximo número de vueltas (y su definición)
- Angulo máximo de giro (y su definición)

❖ Asignar una posición a la orientación actual del motor

❖ Velocidad máxima de giro permitida para el motor

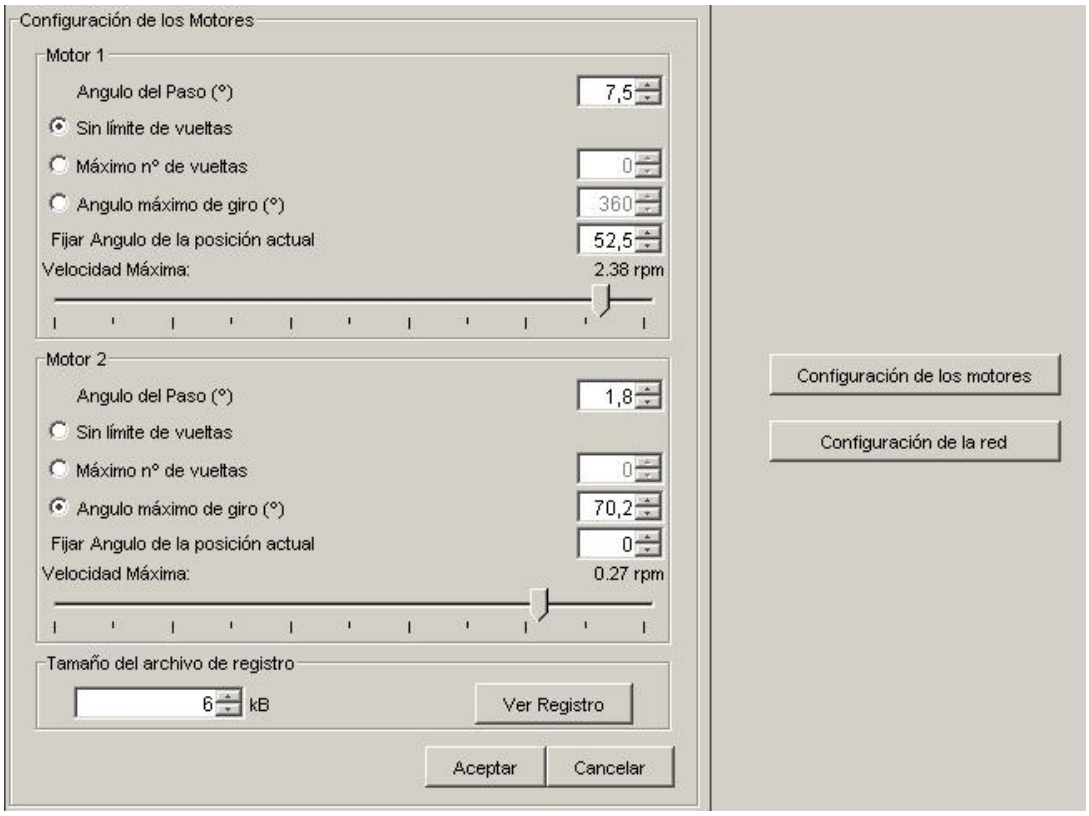


Figura 6-21 : Aspecto del panel de opciones de motores

- **public class PanelOpcionesMotor extends JPanel implements ActionListener, ChangeListener**

Definición de la clase. Hereda de *JPanel* con el objetivo de contar con los métodos necesarios para crear el panel; además implementa las interfaces de eventos *ActionListener* y *ChangeListener* para generalizar las acciones a realizar por los botones y los cuadros numéricos.

- **public PanelOpcionesMotor(OpcionesMotores tablaMotor,boolean conectado_red)**

Constructor de la clase. Su función principal es la de colocar cada uno de los componentes en su lugar, además de asignarles su valor inicial, a partir de la tabla de opciones de los motores. Además, requiere conocer si actualmente el servidor

está atendiendo usuarios, en cuyo caso los elementos del panel aparecerán deshabilitados, con el fin de no permitir su modificación; en cualquier caso, el panel cuenta con funciones que indican los cambios en el estado de la conexión.

- **private void actionAceptar()**

Este método es empleado cuando se acciona el botón *"aceptar"*. Se encarga de encuestar a cada uno de los elementos del panel a cerca de sus estado, para ir almacenándolo en la configuración de los motores.

Si entre los parámetros modificados está el tamaño del archivo de registro, un mensaje advertirá al usuario de que los cambios tendrán efecto una vez se reinicie la aplicación.

Una vez finalizadas sus operaciones, desactiva los botones de *aceptar* y *cancelar*, indicando de esta manera que se han almacenado los nuevos valores, y por tanto, no hay diferencia entre los valores guardados y los presentados en los componentes del panel.

- **private void actionCancelar()**

Este método es invocado cuando se acciona el botón *"cancelar"*. Se encarga de devolver a cada uno de los componentes del panel al estado anterior a la modificación, es decir, al último estado almacenado en el archivo de configuración de los motores.

Una vez finalizadas sus operaciones, desactiva los botones de *aceptar* y *cancelar*, indicando de esta manera que no hay diferencia entre los valores guardados y los presentados en los componentes del panel, ya que se ha vuelto a la configuración anterior.

- **public void stateChanged(ChangeEvent ev)**

Método invocado tras la modificación de un cuadro numérico (*JSpinner*). Se limita a activar los botones de *aceptar* y *cancelar*, para que de esta manera los datos de la configuración puedan ser modificados, o los cuadros restaurados.

```
public void stateChanged(ChangeEvent ev){
```

```
  aceptar.setEnabled(true);
  cancelar.setEnabled(true);
}
```

- **private double anguloCorrecto(double angulo, double angulo_paso)**

Los cuadros numéricos que hacen referencia a ángulos no pueden tomar cualquier valor, ya que éste debe depender del ángulo de paso de cada motor; es decir, no es posible indicar, por ejemplo, como posición actual del motor, un ángulo que nunca alcanzará. Es por ello que esta función determina si el ángulo indicado en estos cuadros es realizable por el motor en cuestión; en caso negativo, cambia este valor por el ángulo más próximo, pero realizable por el motor.

```
private double anguloCorrecto(double angulo, double angulo_paso){
  double temp = (double)(angulo / (angulo_paso));

  if (Math.floor(temp) == temp)
    return angulo; //es correcto

  if (Math.floor(temp) + 0.5 > temp)
    return (Math.floor(temp))*(angulo_paso);
  else
    return (Math.floor(temp)+1)*(angulo_paso);
}
```

- **public void conectado(boolean conectado)**

Método empleado para avisar al panel de un cambio en el estado del servidor de red. De esta manera, si se conecta la red, se desactivan las opciones del panel y si se desconecta, se vuelven a activar.

- **La clase PanelOpcionesRed**

Esta clase presenta el panel de opciones asociadas a la red. En él es posible definir un buen número de parámetros relacionados con la gestión de conexiones, como son:

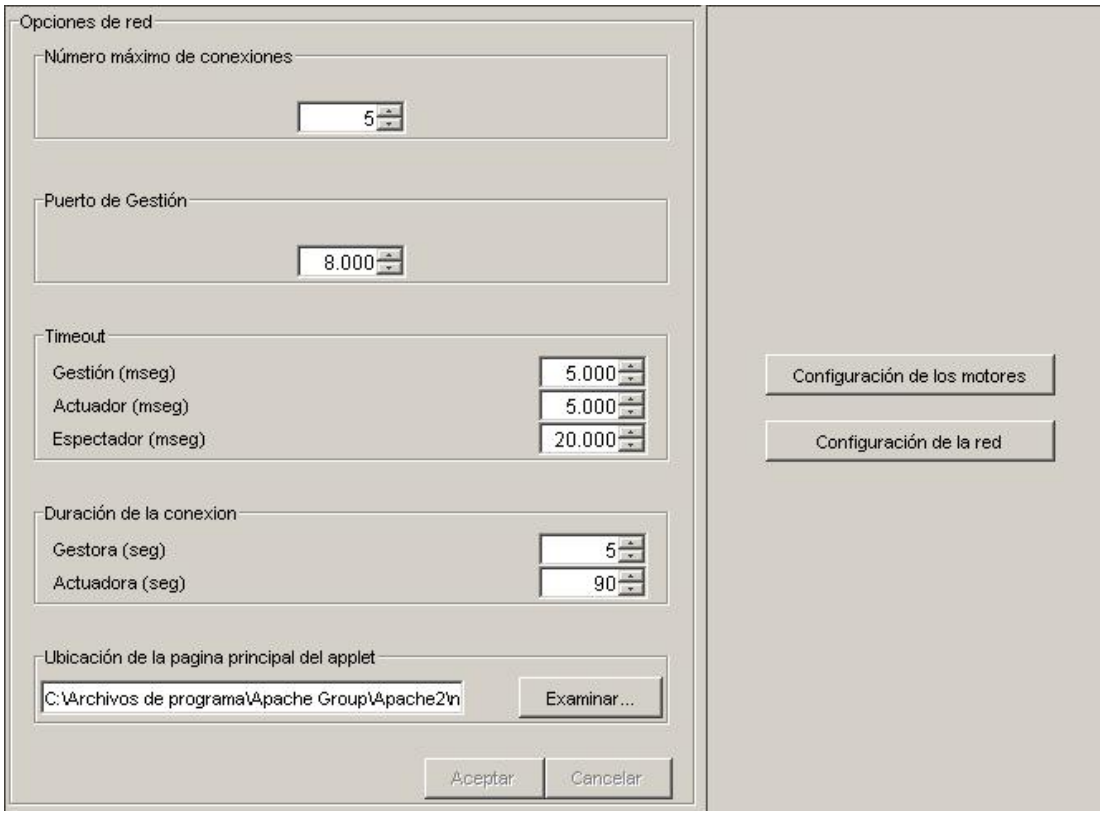


Figura 6-22 : Panel de configuración de la red

- Número máximo de conexiones
- Puerto de gestión
- Tiempo de *timeout* de paquetes
  - Conexión gestora
  - Conexión actuadora
  - Conexión espectadora
- Duración de las conexiones
  - Gestora
  - Actuadora
- Ubicación de la página principal del applet

La figura 6-22 muestra el aspecto de este panel dentro del panel general de opciones.

- **public class PanelOpcionesRed extends JPanel implements ChangeListener,ActionListener**

Definición de la clase. Hereda de *JPanel* con el objetivo de contar con los métodos necesarios para crear el panel; además implementa las interfaces de eventos

ActionListener y *ChangeListener* para generalizar las acciones a realizar por los botones y los cuadros numéricos.

- **public PanelOpcionesRed(OpcionesRed valoresRed, boolean conectadoAred)**

Constructor de la clase. Su función principal es la de colocar cada uno de los componentes en su lugar, además de asignarles un valor inicial, a partir de la tabla de opciones de la red. Además, requiere conocer si actualmente el servidor está atendiendo usuarios, ya que en caso afirmativo, los cambios realizados en el panel sólo tendrán efecto una vez se reinicie el servidor de red (situación que será indicada al usuario por medio de un mensaje, cuando éste trate de confirmar los cambios); en cualquier caso, el panel cuenta con funciones que indican los cambios en el estado de la conexión.

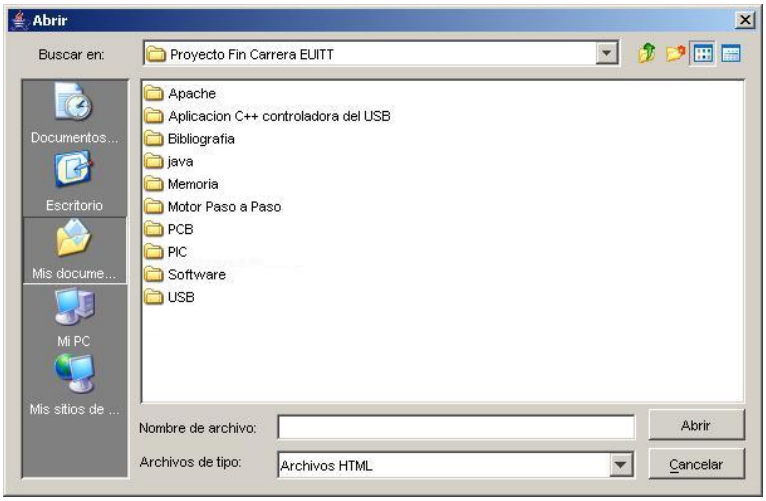


Figura 6-23 : Diálogo de selección de archivos

El apartado de indicación de la ruta en la que reside la página principal del applet cuenta con un botón que permite desplegar un diálogo de selección de archivos, como el mostrado en la figura 6-23, con el objetivo de realizar esta tarea de un modo más cómodo y sencillo.

- **private void actionAceptar()**

Este método es empleado cuando se acciona el botón *"aceptar"*. Se encarga de encuestar a cada uno de los elementos del panel a cerca de sus estado, para ir almacenándolo en la configuración de la red.

Si entre los parámetros modificados está la ruta de la página principal del applet, ésta es comprobada (su existencia). En caso de error, se informa al usuario de que el valor indicado no se tendrá en cuenta, y se mantendrá el valor almacenado anteriormente.

Una vez finalizadas sus operaciones, desactiva los botones de *aceptar* y *cancelar*, indicando de esta manera que se han almacenado los nuevos valores, y por tanto, no hay diferencia entre los valores guardados y los presentados en los componentes del panel.

- **private void actionCancelar()**

Este método es invocado cuando se acciona el botón *"cancelar"*. Se encarga de devolver a cada uno de los componentes del panel al estado anterior a la modificación, es decir, al último estado almacenado en el archivo de configuración de la red.

Una vez finalizadas sus operaciones, desactiva los botones de *aceptar* y *cancelar*, indicando de esta manera que no hay diferencia entre los valores guardados y los presentados en los componentes del panel, ya que se ha vuelto a la configuración anterior.

- **public void stateChanged(ChangeEvent ev)**

Método invocado tras la modificación de un cuadro numérico (*JSpinner*). Se limita a activar los botones de *aceptar* y *cancelar*, para que de esta manera los datos de la configuración puedan ser modificados, o los cuadros restaurados.

```
public void stateChanged(ChangeEvent ev){
   aceptar.setEnabled(true);
   cancelar.setEnabled(true);
 }
```

- **La clase JModeloTabla**

Esta clase es empleada para definir la estructura de la tabla que gestiona los usuarios y su estado, no desde el punto de vista visual, sino en cuanto a su implementación (añadir, modificar, ordenar y modificar las filas)

- **public class JModeloTabla extends DefaultTableModel**

Definición de la clase. Hereda los métodos de la clase *DefaultTabeModel*, que es el modelo por defecto utilizado para controlar los JTable de Java, sobrecargando sus métodos más importantes para adaptarlos a las necesidades de la tabla.

- **public JModeloTabla()**

Constructor de la clase. En él se definen los nombres de cada columna.

```
public JModeloTabla(){
    super();

     setColumnIdentifiers( new String[] {"Login", "Nombre", "Permiso",
      "Activo", "Conexión", "Dirección", "Ultimo Acceso"});

    columna_referencia = 3;
 }
```

- **public boolean isCellEditable(int fila, int columna)**

Sobrecarga del método utilizado por *DefaultTableModel* para indicar si la celda seleccionada es editable. Siempre devuelve *false*.

- **public synchronized void anadirFila(String login, entradaUsuario usuario, entradaConexion conexión)**

Método utilizado para añadir una fila; es decir, la información correspondiente a un usuario, para lo cual se buscan las correspondencias con el login en las tablas de usuarios y de conexiones.

```
public synchronized void anadirFila(String login, entradaUsuario usuario,
                    entradaConexion conexion){
  entradaJTable entrada = new entradaJTable(conformaTabla(
                                login, usuario, conexion));

    int fila = buscaFila(entrada);
    insertRow(fila,entrada.getDataVector());
 }
```

- **public synchronized boolean eliminaFila(String login)**

Elimina la fila cuyo *login* corresponda con el indicado por parámetros.

- **public synchronized boolean eliminaFila(int fila)**

Elimina la fila cuyo índice coincide con el indicado por parámetros.

- **public synchronized boolean desconectaUsuario(entradaConexion entrada)**

Marca al usuario como *"No Conectado"*.

- **public synchronized boolean modificaFila(String oldLogin, String newLogin, entradaUsuario usuario, entradaConexion conexion)**

Actualiza el contenido de una fila. Si lo que se intenta modificar es el *login*, los parámetros *oldLogin* y *newLogin* tendrán valores diferentes.

- **private int buscaSitioString(String busca, int primeraFila, int ultimaFila, int columna)**

Método empleado para la inserción ordenada en la tabla. Busca dentro del rango definido, el índice idóneo en el que introducir una entrada, cuyo campo *columna* tiene el valor *busca*.

- **private int buscaSitioPermiso(String busca, String login, int primeraFila, int ultimaFila, int columna)**

Método empleado para la inserción ordenada en la tabla. Busca dentro del rango definido, el índice idóneo en el que introducir una entrada, cuyo campo *columna* tiene el valor *busca*, teniendo en cuenta además la jerarquía de permisos: Administrador, Actuador y Espectador.

- **private int buscaSitioConectado(boolean busca, String login, int primeraFila, int ultimaFila, String tipoConexion)**

Método empleado para la inserción ordenada en la tabla. Busca dentro del rango definido, el índice idóneo en el que introducir una entrada, ordenando por *conectados/no conectados*, y entre ellos, por *login*.

- **private int buscaSitioUltimoAcceso(Date busca, String login, boolean conectado, String tipoConexion)**

Método empleado para la inserción ordenada en la tabla. Busca dentro del rango definido, el índice idóneo en el que introducir una entrada, ordenando la fecha de último acceso, y entre ellos, por *login*.

- **La clase PanelUsuarios**

Esta clase crea un panel que contiene una tabla con los datos de los usuarios registrados en la aplicación, así como los medios para añadir, modificar o eliminar entradas, además de permitir desconectar a los mismos. La figura 6-24 muestra el aspecto de este panel.

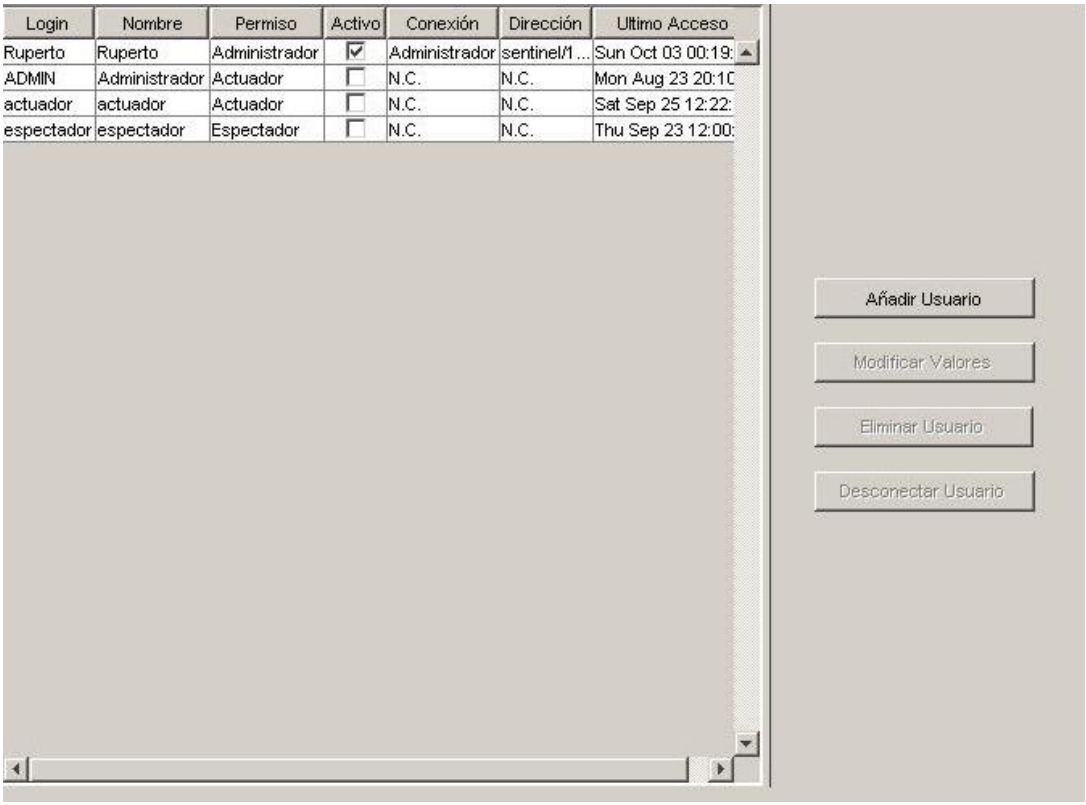


Figura 6-24 : Panel de Usuarios

Los datos contenidos en la tabla son los siguientes:

- Login
- Nombre del usuario
- Permisos
- Conectado
- Tipo de conexión
- Dirección desde la que se conecta
- Fecha de último acceso

- **public class PanelUsuarios extends JSplitPane implements UsuarioListener, ConexionListener**

Definición de la clase. Hereda de *JSPlitPane*, con el objetivo de crear un panel dividido (una parte para la tabla y la otra para los botones de acción); implementa las interfaces *UsuarioListener* y *ConexionListener*, con el objetivo de estar al tanto de los sucesos en las tablas de usuarios y de conexiones (respectivamente).

- **public PanelUsuarios(TablaUsuarios tablaU, TablaConexiones tablaC, ServidorRed servidorRed)**

Constructor de la clase. Entre sus funciones están las de colocar los componentes en los paneles (tabla y botones), además de indicar sus acciones. También inicializa la tabla de la interfaz gráfica; para ello añade todas las entradas de la tabla de usuarios, y marca como conectadas (añadiendo además la dirección del cliente) aquellas cuyo *login* también resida en la tabla de conexiones.

Las acciones de los botones de la parte derecha del panel son:

- **Añadir Usuario**

Esta opción es accesible en cualquier momento. Abre una nueva ventana en la que se solicitan los datos del nuevo usuario; como requisitos se exige cumplimentar los campos de *login*, *password* y comprobación de *password*, el *login* no debe coincidir con el asignado a otro usuario registrado (y de longitud inferior a doce caracteres) y el password sólo puede contar con caracteres alfanuméricos. La figura 6-25 muestra el diálogo junto a un

mensaje de error, debido a la repetición del *login* (con el de un usuario ya registrado).

Figura 6-25 : Diálogo de usuarios, error al añadir

❖ **Modificar Valores**

Opción accesible una vez se ha seleccionado un usuario de la tabla. Abre una nueva ventana (idéntica a la empleada para crear una nueva entrada), que incorpora los datos almacenados del usuario. Los criterios para la modificación de los campos es el mismo que el empleado para crearlos.

❖ **Eliminar Usuario**

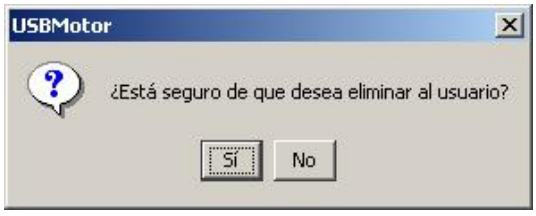


Figura 6-26 : Diálogo de confirmación de usuario

Opción accesible una vez se ha seleccionado un usuario de la tabla. Tras la confirmación de la acción, elimina un usuario de la tabla, el cual quedará permanentemente excluido de la lista de usuarios registrados. La figura 6-26 muestra el diálogo de confirmación de eliminación.

❖ **Desconectar Usuario**

Permite desconectar (expulsar de la herramienta) al usuario seleccionado en la tabla de usuarios, si es que éste se encuentra actualmente conectado (si no lo

está, la opción no aparece disponible). También requiere confirmar la acción en un diálogo creado para ese cometido.

- **public void recibido(EventoUsuario e)**

Implementación del método declarado en la interfaz *UsuarioListener*, el cual indica que se ha producido alguna acción sobre la tabla de usuarios.

```
public void recibido(EventoUsuario e) {

 switch (e.suceso) {
  case 0: //nueva entrada
   … // la añade a la tabla
   break;
  case 1: //entrada eliminada
   if (!modelo.eliminaFila(e.login))
    …
   break;
  case 2: //entrada modificada
   …
   break;
  case 3: //eliminar
   … //elimina de las tablas, y desconecta al usuario
   break;
   }
 }
}
```

- **public void recibido(EventoConexion e)**

Implementación del método declarado en la interfaz *ConexionListener*, el cual indica que se ha producido alguna acción sobre la tabla de conexiones.

```
switch(e.suceso){
  case 0://nueva entrada
  case 2://modificación
   … //actualiza los datos de la entrada
   break;
  case 1://eliminacion (desconexión)
       … //marca la entrada como desconectada
   break;
```

```
    }
}
```

- **La clase FrameServidor**

Esta clase es la que da soporte a toda la interfaz gráfica de la aplicación servidora, ya que ofrece los recursos de una ventana convencional, en la que colocar los diferentes paneles.

Su estructura se basa en el empleo de tres elementos:

- **Barra superior**

Cuenta con los botones necesarios para cambiar entre los diferentes paneles controladores de la aplicación, tal y como muestra la figura 6-27.

Figura 6-27 : Barra superior del servidor

Su funcionalidad, de izquierda a derecha es la siguiente:

- Proceder a la desconexión del usuario (abandonar la sesión)
- Acceder al panel de movimiento de los motores
- Acceder al panel de configuración
- Acceder al panel de gestión de usuarios
- Abrir la ayuda
- Cerrar la aplicación

- **Panel central**

Panel destinado a ser contenedor de los paneles que el usuario vaya necesitando.

- **Barra Inferior**

Esta barra dispone básicamente de dos botones y un cuadro de texto (ver figura 6-28).

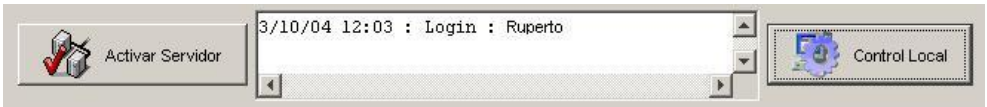


Figura 6-28 : Barra inferior

El botón de la izquierda es empleado para activar/desactivar la escucha de peticiones procedentes de los clientes remotos, mientras que el de la derecha es utilizado para alternar entre control local (los motores son gestionados por el administrador) y control remoto (se permite que un usuario actuador controle, remotamente, los motores).

El cuadro de texto se emplea para informar al usuario de los sucesos (junto a su fecha y hora) producidos en el servidor.

- **public class FrameServidor extends JFrame implements PanelLoginListener, ServidorListener, MotorMovidoListener, PanelMovimientoListener**

Definición de la clase. Hereda de *JFrame*, con el objetivo de disponer de los métodos necesarios para crear una ventana. Implementa varias interfaces de gestión de eventos: *PanelLoginListener*, utilizada para conocer cuándo un usuario se ha identificado correctamente, *ServidorListener* es empleada para recibir eventos del gestor de la red, *MotorMovidoListener* avisa una vez se ha completado un movimiento, o este ha fallado y finalmente *PanelMovimientoListener* se utiliza para avisar de que el usuario administrador ha decidido ejecutar un movimiento.

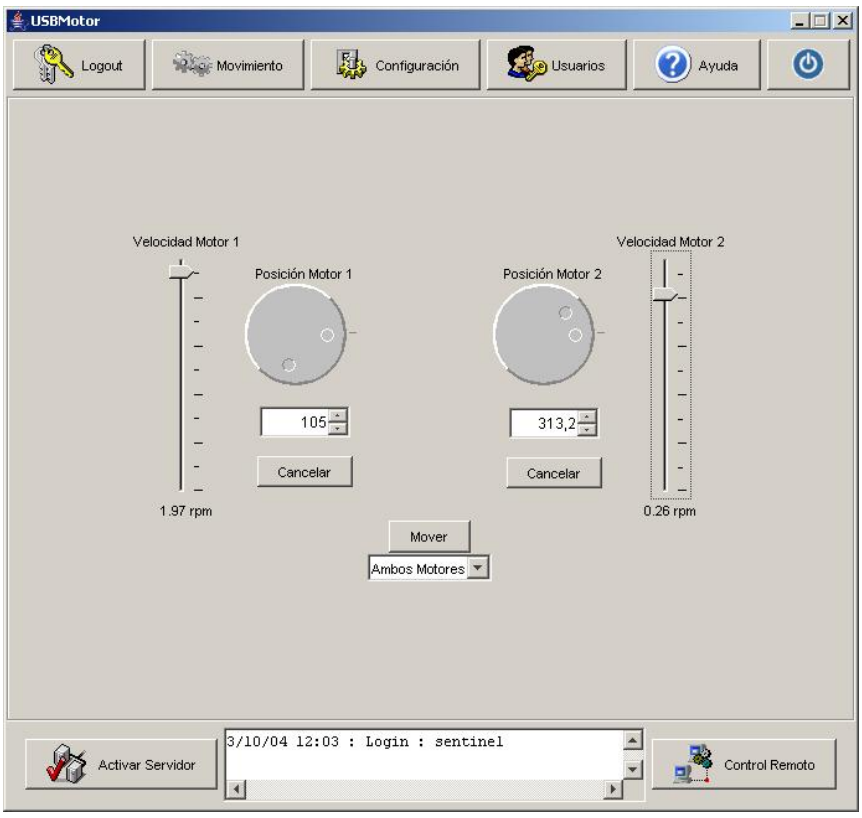


Figura 6-29 : Ventana principal de la aplicación *servidor*

La figura 6-29 muestra el aspecto de la ventana principal de la aplicación, estando activo el control local y el panel de control de los motores.

- **public FrameServidor(String titulo, ServidorRed servidor, OpcionesMotores tablaMotores, OpcionesRed tablaRed, TablaConexiones tablaConexiones, ToUSB usb)**

Constructor de la clase. Se encarga de inicializar los elementos colocados en la ventana, asignando como panel inicial para el bloque central, el panel de *login*.

- **private JPanel barraSuperior()**

Método utilizado para crear e inicializar la barra superior de botones e indicar sus acciones correspondientes.

Entre estos botones, cabe destacar el que hace referencia al *login*, ya que mientras se permanezca en el panel de identificación, el botón permanece inactivo y con el texto “login”. Una vez que el usuario se ha identificado y ha cambiado de panel, el botón se activa y cambia su texto por el de “logout”; si el botón es accionado en este estado, abrirá un diálogo de confirmación de abandono de la sesión, que en caso de ser afirmativo, fija como panel central el de *login*.

Mientras el usuario no se identifique, todos los botones, exceptuando los de *ayuda* y *salir* permanecerán inactivos, evitando que un usuario no registrado pueda acceder a la herramienta. La figura 6-30 muestra esta situación.

Figura 6-30 : Estado inicial del servidor

- **private void login(boolean loged)**

Función encargada de alterar el estado del botón de *login* de la barra superior, según el estado que se indique por parámetros.

```
...
  if (loged) {
    boton.setText(" Logout");
   }
   else {
    boton.setText("  Login");
...
```

- **private JPanel barraInferior()**

Método utilizado para crear e inicializar la barra inferior e indicar las acciones correspondientes a cada botón.

Se debe destacar que los botones de esta barra cambian su icono y texto, una vez pulsados, ya que de esta manera se indica que, si son pulsados otra vez, se retorna al estado anterior. La figura 6-31 muestra los posibles cambios de estado de estos botones.

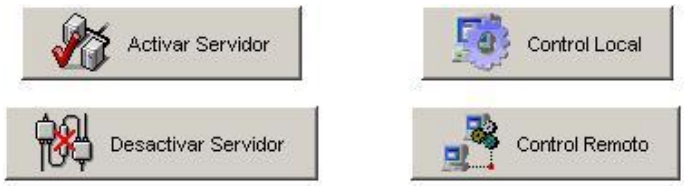


Figura 6-31 : Combinaciones de botones de la barra inferior

- **private void estadoBotonServidor(JButton boton, boolean conectado)**

Cambia el estado del botón que inicia o finaliza la escucha de peticiones de usuarios remotos, según el estado pasado por parámetros.

- **private void estadoBotonControl(JButton boton, boolean local)**

Cambia el estado del botón que alterna entre control local o remoto de los motores, según el estado pasado por parámetros.

- **public synchronized void anadeAviso(String mensaje)**

Añade un texto (precedido por la fecha), al cuadro de texto destinado para los avisos.

```
public synchronized void anadeAviso(String mensaje) {
  DateFormat fecha = DateFormat.getInstance();

  avisos.append(fecha.format(new Date()) + " : " + mensaje + "\n");
}
```

- **public void loged(PanelLoginEvent ev)**

Definición del método declarado en la interfaz *PanelLoginListener*, cuya función es avisar de que un usuario se ha identificado.

```
public void loged(PanelLoginEvent ev) {
  if (ev.permiso == (TablaUsuarios.admin + TablaUsuarios.actuador +
                TablaUsuarios.udp)) { //comprobamos permisos
    login(false);
    anadeAviso("Logout");
  }
  else {
    login(true);
    anadeAviso("Login : " + ev.login);
  }
}
```

- **public void recibido(EventoServidor ev)**

Definición del método declarado en la interfaz *ServidorListener*, cuya función es la de indicar cambios en el estado del servidor de red (conectado / desconectado).

- **public void moverMotor(PanelMovimientoEvent ev)**

Definición del método declarado en la interfaz *PanelMovimientoListener*. Su función es avisar cuando el panel de movimientos ha solicitado un movimiento (local); de esta manera podrá presentarse un aviso de dicha solicitud en el cuadro de texto de la barra inferior y controlar cuántos motores se desean mover.

```
public void moverMotor(PanelMovimientoEvent ev){
  if (ev.ID == 0)
    contador = 2;
  else
    contador = 1;

  anadeAviso("Acción solicitada ...");
 }
```

- **public void motorMovido(MotorMovidoEvent ev)**

Definición del método declarado en la interfaz *MotorMovidoListener*, utilizado para indicar que se ha producido un movimiento, o un error en el mismo. En cualquier caso, el motivo del suceso se verá reflejado en el cuadro de avisos de la barra inferior; si además el movimiento falla, y éste ha sido producido localmente, se abrirá un diálogo de mensaje informando de dicha situación.

```
if ((ev.tipo == 1) || (ev.tipo == 2)){
  ...
   if (ev.tipo == 2){ //se han movido, pero no exactamente como se esperaba
     anadeAviso("No ha sido posible mover correctamente los motores");
     if (tablaConexiones.buscaLogin("ADMIN") ==
          tablaConexiones.MAX_CONEXIONES) { //control local
        dialogo("No ha sido posible mover correctamente los motores");
      }
   }
   else
     anadeAviso("Movimiento ejecutado correctamente");
  }
 ...
  switch(ev.tipo){ //en caso de error
    case 3: //error al INICIAR el pipe
      if (tablaConexiones.buscaLogin("ADMIN") !=
           tablaConexiones.MAX_CONEXIONES) {
        dialogo("Error en la comunicación con la aplicación " +
             "controladora de USB");
       }
      anadeAviso("Error en la comunicación con la aplicación " +
           "controladora de USB");
      break;
```

```
        case 4: //error en la aplicacion USB
          if (tablaConexiones.buscaLogin("ADMIN") !=
              tablaConexiones.MAX_CONEXIONES) {
            dialogo("Error en la aplicación controladora de USB");
          }
          anadeAviso("Error en la aplicación controladora de USB");
          break;
        case 5: //error al acceder al dispositivo
          if (tablaConexiones.buscaLogin("ADMIN") !=
              tablaConexiones.MAX_CONEXIONES) {
            dialogo("Error al acceder al dispositivo : Ha sido desconectado");
          }
          anadeAviso("Error al acceder al dispositivo : Ha sido desconectado");
          break;
      }
    }
```

Si el movimiento se ejecuta correctamente, se añade una nueva entrada al archivo de registro de los movimientos.

```
tablaMotores.movido(tablaMotores.getAnguloActual(1),
                    tablaMotores.getVelocidad(1),
                    tablaMotores.getAnguloActual(2),
                    tablaMotores.getVelocidad(2),
                    tablaConexiones.getEntrada_tipo(
    TablaConexiones.actuador, 0).getLogin());
```

6.3.3 Las Clases del Cliente

Además de las clases que comparte con el servidor, el cliente requiere de otras tantas que permitan la gestión de las conexiones y controlen la interfaz gráfica, dadas las peculiaridades del entorno en el que se ejecutará la aplicación (una página Web).

- **Clases de Gestión de Red**

  - **La clase PaqueteTCPCliente**

Esta clase se encarga del mantenimiento de la conexión TCP con el servidor.

- **class PaqueteTCPCliente extends Thread**

Definición de la clase. Hereda de Thread, ya que requiere de un hilo de ejecución independiente para su funcionamiento.

- **public PaqueteTCPCliente (int puerto, InetAddress direccion_servidor, short id, int socketTimeout, PaqueteTCPListener l) throws IOException**

Constructor de la clase. Se encarga de inicializar los valores utilizados durante la conexión, e iniciar el proceso de conexión (invocando al método *Start()*).

```
public PaqueteTCPCliente (int puerto, InetAddress direccion_servidor,
                          short id, int socketTimeout,
                          PaqueteTCPListener l) throws IOException{
  this.socketTimeout = socketTimeout;
  this.id = id;
  this.direccion_servidor=direccion_servidor;
  this.puerto=puerto;

  this.listener=l;
  rx = new PaqueteTCPrecepcion();
  rx.addTCPListener(listener);
  start();
}
```

- **public synchronized void run ()**

Sobrecarga del método principal de la clase Thread. Su función es la de iniciar la comunicación con el servidor en un hilo de ejecución separado, de tal forma que no bloquee el funcionamiento normal de la aplicación.

```
public synchronized void run (){
  try{
    if (socket != null)
      close();

    socket = new Socket(direccion_servidor,puerto);
    socket.setSoTimeout(socketTimeout);

    tx = new PaqueteTCPtransmision(socket);
```

```
        rx.setParam(id,socket);

        rx.start();
      }
    ...
```

- **public boolean enviar(PDU_TCP pdu)**

Método empleado para enviar una PDU al servidor.

```
public boolean enviar(PDU_TCP pdu){
  if ((socket != null) && (socket.isClosed() == false))
    return tx.enviar(pdu);
  else
    return false;
}
```

- **public void close()**

Método encargado de cerrar la conexión con el servidor.

- **La clase PaqueteUDPCliente**

Esta clase se encarga del mantenimiento de la conexión TCP con el servidor.

- **class PaqueteUDPCliente**

Definición de la clase.

- **public PaqueteUDPCliente(int puerto, int UDPtimeout, PaqueteUDPListener l) throws IOException**

Constructor de la clase. Se encarga de inicializar los valores utilizados durante la conexión.

```
public PaqueteUDPCliente(int puerto, int UDPtimeout, PaqueteUDPListener l)
                    throws IOException{
  try{
    socket = new DatagramSocket(puerto);
    socket.setSoTimeout(UDPtimeout);
```

```
    tx = new PaqueteUDPtransmision(puerto,socket);
    rx = new PaqueteUDPrecepcion(socket);

    rx.addUDPListener(l);
    rx.start();
   }
   catch(IOException e){throw e;}
  }
```

- **public void close()**

Cierra el socket UDP.

- **La clase ClienteRed**

Es la clase encargada de gestionar todo el proceso de comunicación en el cliente (controla todos los mensajes enviados al cliente y se ocupa de su respuesta).

- **class ClienteRed implements PaqueteTCPListener, PaqueteUDPListener, PanelMovimientoListener**

Definición de la clase. Implementa las interfaces *PaqueteTCPlistener* (eventos TCP), *PaqueteUDPlistener* (eventos UDP), *PanelMovimientoListener* (eventos del panel de movimientos).

- **public ClienteRed(InetAddress direccion_servidor,int TCPtimeout, int UDPtimeout, int puerto_gestion, String login, String password, short tipo_conexion, DatosMotoresCliente tablaMotores, ClienteListener l) throws IOException**

Constructor de la clase. Inicializa los valores necesarios para ser utilizados en la clase, e inicia la conexión gestora.

```
 public ClienteRed(InetAddress direccion_servidor,int TCPtimeout, int
UDPtimeout, int puerto_gestion, String login, String password, short tipo_conexion,
DatosMotoresCliente tablaMotores, ClienteListener l) throws IOException {
...
   tcp = new PaqueteTCPCliente[2];
   arrayTCP = new PDU_TCP[2];
   ultimaUDP = null;
```

```
tcp[gestor] = new PaqueteTCPCliente(puerto_gestion,
                    direccion_servidor,
                    gestor,
                    TCPtimeout,
                    this);

tcp[actuador] = null;
}
```

- **public void close()**

Método empleado para cerrar todas las conexiones abiertas y liberar sus recursos asociados.

- **public boolean enviarTCP(short conexionID,PDU_TCP pdu)**

Método utilizado para enviar una PDU al servidor, a través de la conexión indicada por *conexionID*.

```
public boolean enviarTCP(short conexionID,PDU_TCP pdu){
  try{
    return tcp[conexionID].enviar(pdu);
  }
  catch(NullPointerException e){return false;}
  }
```

- **public boolean enviarUDP(PDU_UDP pdu)**

Envía una PDU a través del socket UDP.

```
public boolean enviarUDP(PDU_UDP pdu){
  return udp.enviar(pdu,direccion_servidor);
}
```

- **public void recibido(EventoPaqueteTCP paquete)**

Definición del método declarado en *PaqueteTCPListener*, el cual es invocado cada vez que llega una PDU_TCP.

Su función es la de gestionar los paquetes entrantes y, según el caso, darles respuesta.

```
public void recibido(EventoPaqueteTCP paquete){
  PDU_TCP pdu = new PDU_TCP();
  try{
  switch (paquete.suceso){
    case 0://recibido correctamente
      switch(paquete.pdu.getTipo()){
        …
        default:
          switch (paquete.id) { //PDUs específicas por servicio
            case gestor:
              mensajesGestor(paquete.pdu);
              break;
            case actuador:
              mensajesActuador(paquete.pdu);
              break;
          }
          break;
      }
      break;

    case 1://timeout
      … //envío de petición de respuesta
      break;

    case 2://Error
        …
      break;

    case 3:
      … //Error
      break;

    case 7://Se ha establecido una conexión con el servidor
       ...
    break;
   }
  }
```

```
    …
  }
```

- **private void desconectarTCP(int id,PDU_TCP motivo)**

Desconecta la sesión TCP indicada, pero antes envía el *motivo* para tal acción.

- **private void mensajesGestor(PDU_TCP pdu_recibida) throws NullPointerException, NumberFormatException, IOException**

Gestiona aquellos paquetes que son exclusivos de la conexión gestora.

```
private void mensajesGestor(PDU_TCP pdu_recibida)
   throws NullPointerException,NumberFormatException,IOException{

 switch(pdu_recibida.getTipo()){
  case 1: //gestion del login
   switch (pdu_recibida.getSubtipo()) {
    case 1: //conexion aceptada
     arrayTCP[gestor] = null;
     switch (pdu_recibida.getIntcampo(0)) {
      case 0: //UDP
       …
       break;
      case 1: //Actuador
       …
       break;
     }
     tcp[gestor].close();
     break;
   }
   break;
  case 2: //datos de motores
   …
   break;
 }
}
```

- **private void mensajesActuador(PDU_TCP pdu_recibida) throws NullPointerException, NumberFormatException, IOException**

Gestiona los mensajes propios de la connexion actuadora.

```
private void mensajesActuador(PDU_TCP pdu_recibida)
  throws NullPointerException,NumberFormatException,IOException{
PDU_TCP pdu = new PDU_TCP();

switch(pdu_recibida.getTipo()){
 case 2: //información de movimiento
  switch (pdu_recibida.getSubtipo()) {
   case 1: //correcto
    …
    break;

   case 3://ha habido un error al mover
    …
    break;
  }
 break;

 case 4: //cambio de permisos
  …
  break;

 case 5: //desconexión
  …
  break;
 }
}
```

- **public void recibido(EventoPaqueteUDP paquete)**

Definición del método declarado en *PaqueteUDPListener*, el cual es invocado cada vez que llega una PDU_UDP.

Su función es la de gestionar los paquetes entrantes y, según el caso, darles respuesta.

```
public void recibido(EventoPaqueteUDP paquete){
  PDU_UDP pdu = new PDU_UDP();

  switch(paquete.suceso){
    case 0: //recibido correctamente
      …
      break;
    case 1: //timeout
      …
    break;
}
```

- **public void moverMotor(PanelMovimientoEvent ev)**

Definición del método declarado en la interfaz PanelMovimientoListener, el cual es invocado cuando un cliente actuador desea actuar sobre los motores (así lo ha indicado en el panel de movimientos).

- **synchronized public void addClienteListener(ClienteListener l)**

Añade un objeto a la lista de *listeners*; es decir, a la lista de objetos a los que avisar cuando el servidor envía ciertas PDUs

- **synchronized public void removeClienteListener(ClienteListener l)**

Elimina el objeto indicado de la lista de *listeners*.

- **protected synchronized void avisaCliente(short ID,short suceso,short motivo)**

Avisa a todos los objetos inscritos en la lista de listeners, del evento.

➢ Clases de la Interfaz Gráfica

❑ La clase PanelLoginServidor

Clase encargada de crear un panel que pide la identificación de usuario. Para ello ofrece dos entradas de texto, una para introducir el login, y otra para el password; esta última cuenta con la peculiaridad de que oculta la contraseña escrita, sustituyendo los caracteres introducidos por asteriscos (*). Además, presenta dos botones con los que indicar el tipo de

conexión que se desea realizar (actuadora o espectadora). La figura 6-32 muestra el aspecto de este panel.

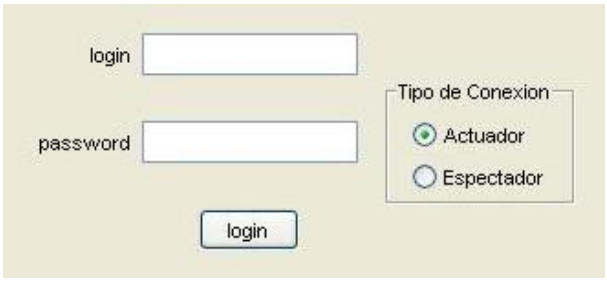


Figura 6-32 : Panel de login en el cliente

Una vez el usuario se identifica, automáticamente se activa el panel de los motores (sea cual sea el tipo de conexión aceptado)

- **public class PanelLoginCliente extends JPanel**

Definición de la clase. Hereda los métodos de JPanel, utilizados para albergar los elementos a representar en el panel.

- **public PanelLoginCliente()**

Constructor de la clase. Se utiliza para inicializar los elementos utilizados en el panel, así como indicar las acciones a llevar a cabo tanto si la identificación es correcta como si no, una vez sea accionado el botón de identificación *"login"*.

- **public void addPanelLoginListener(PanelLoginListener listener)**

Añade un objeto a la lista de *listeners*; es decir, a la lista de objetos a los que avisar cuando un usuario se identifica correctamente.

- **public void removePanelLoginListener(PanelLoginListener listener)**

Elimina el objeto indicado por parámetros de la lista de *listeners*.

- **void avisaLogin(String login, char[] password, short permiso)**

Método utilizado para avisar a los objetos inscritos en la lista de listeners, una vez se ha identificado un usuario.

- **La clase AppletCliente**

Es la clase que sustenta la aplicación cliente, aportando los recursos necesarios para poder ser ejecutada en una página Web. Se encarga de invocar a cada uno de los paneles y recursos necesarios, además de gestionar los eventos enviados por los mismos.

Su estructura visual se basa en el empleo de tres elementos:

❖ **Barra superior**

Cuenta con los botones necesarios para cambiar entre los diferentes paneles controladores de la aplicación, tal y como muestra la figura 6-33.

Figura 6-33 : Barra Superior

Su funcionalidad, de izquierda a derecha es la siguiente:

- Proceder a la desconexión del usuario (abandonar la sesión)
- Abrir la ayuda

❖ **Panel central**

Panel destinado a ser contenedor de los paneles que el usuario vaya necesitando.

❖ **Barra Inferior**

Esta barra dispone básicamente de un botón y un cuadro de texto (ver figura 6-34).

Figura 6-34 : Barra Inferior

El botón de la barra es empleado para alternar (si el servidor lo permite) entre conexión actuadora y espectadora.

- **public class AppletCliente extends JApplet implements ClienteListener, PanelLoginListener**

Definición de la clase. Hereda de *JApplet* con el fin de obtener los métodos necesarios para desenvolverse en el entorno de una página web. Implementa las interfaces *ClienteListener* (cliente de red) y *PanelLoginListener* (avisa de la identificación del usuario).

- **public void init()**

Método sobrecargado a la clase *JApplet*. Su misión es la de inicializar los valores que posteriormente utilizará el Applet (como puede ser la obtención de los parámetros de red) y de colocar los elementos gráficos a utilizar por la aplicación.

```
public void init() {
    …
    try{  //obtenemos el archivo de configuracion de la red
      URL url = new URL(getCodeBase(),"red.inf");
      ObjectInputStream in = new ObjectInputStream(url.openStream());
      tablaRed = (valoresRed) in.readObject();
      in.close();
    }
    catch (Exception e){
      JOptionPane.showMessageDialog(this,"No ha sido posible cargar los " +
                                    "parámetros de la conexión desde el " +
                                    "servidor");
      throw(e);
    }
    …
}
```

- **public void destroy()**

Método sobrecargado a la clase *JApplet*. Representa la última oportunidad para liberar recursos, los cuales en nuestro caso se centran en las conexiones de red.

```
public void destroy(){
   if (cliente != null)
     cliente.close();  //desconectamos
 }
```

El cuadro de texto se emplea para informar al usuario de los sucesos (junto a su fecha y hora) producidos en el servidor.

- **private JPanel barraSuperior()**

Método utilizado para crear e inicializar la barra superior de botones e indicar sus acciones correspondientes.

Mientras el usuario no se identifique, el único botón activo será el de la ayuda. La figura 6-35 muestra esta situación.

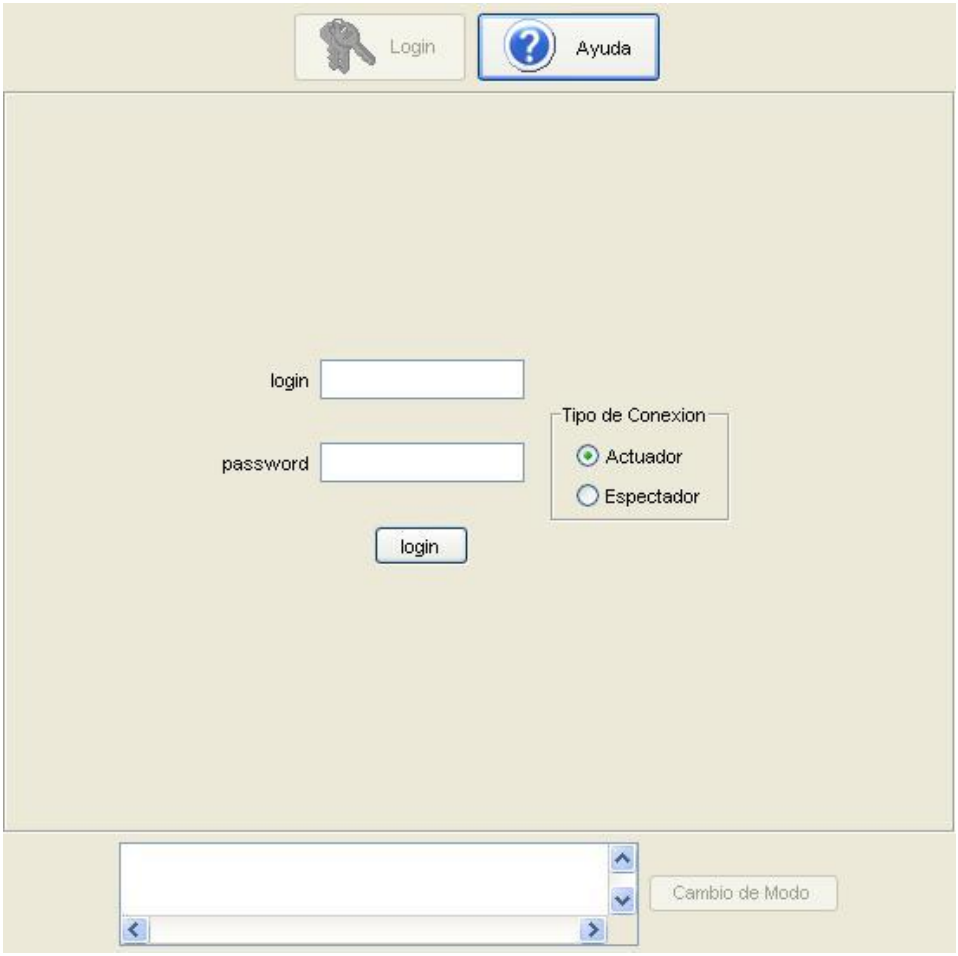


Figura 6-35 : Situación inicial del applet

- **private JPanel barraInferior()**

Método utilizado para crear e inicializar la barra inferior e indicar las acciones correspondientes al botón.

El botón de esta barra cambia su texto y su icono, según sea el modo de conexión del usuario y la disponibilidad de conexiones; es decir si el usuario es actuador, la activación del botón supondrá pasar a espectador, mientras que si actualmente es espectador y no hay ningún actuador, la activación del botón supondrá el inicio de la negociación del cambio de modo. La figura 6-36 muestra los distintos estados del botón.

Figura 6-36 : Distintos estados del botón de la barra inferior

- **public synchronized void anadeAviso(String mensaje)**

Añade un texto (precedido por la fecha), al cuadro de texto destinado para los avisos.

```
public synchronized void anadeAviso(String mensaje) {
   DateFormat fecha = DateFormat.getInstance();

   avisos.append(fecha.format(new Date()) + " : " + mensaje + "\n");
 }
```

- **public void recibido (EventoCliente ev)**

Definición del método declarado en la interfaz *ClienteListener*. De esta manera el gestor de la red avisa de eventos importantes (como información de los motores o errores en la conexión).

- **public void loged(PanelLoginEvent ev)**

Definición del método declarado en la interfaz *PanelLoginListener*, cuya función es avisar de que un usuario se ha identificado. En este caso, se inicializa el gestor de conexiones (y con él la conexión gestora)

```
public void loged(PanelLoginEvent ev){

  String host = getCodeBase().getHost();

  if (cliente != null)
    cliente.close();
```

```
    try{
      cliente = new ClienteRed(InetAddress.getByName(host),
                               tablaRed.timeoutGestion,
                               tablaRed.timeoutEspectador,
                               tablaRed.puertoGestion,
                               ev.login,
                               new String(ev.password),
                               ev.permiso,
                               tablaMotores,
                               this);
      anadeAviso("Login : " + ev.login);
    }
    catch (IOException e){
      JOptionPane.showMessageDialog(this,"El servidor no responde");
    }
  }
```

- **private void login(boolean loged,short tipo)**

Función invocada tras la confirmación de la identidad del usuario, por parte del servidor, o tras el deseo del cliente de salir de la sesión.

En cualquiera de los dos casos, se ocupa de fijar el estado correcto a las barras (superior e inferior), según el estado de la aplicación.

```
  private void login(boolean loged,short tipo) {
    …
    if (loged) { //intento de acceso
      boton.setText("  Logout");

      cambioModo(tipo,false);
      boton.setEnabled(true);
    }
    else { //intento de desconexión
      boton.setText("  Login");
      boton.setEnabled(false);
      …
    }

  }
```

- **private void cambioModo(short tipo,boolean permiso)**

Una vez realizado un cambio en el modo de conexión, actualiza el estado del botón de la barra inferior.

```
private void cambioModo(short tipo,boolean permiso){
 …
 if (tipo == 1){ //modo actuador
  boton.setText("  Espectador");
  boton.setIcon(crearIcono("iconos/controlLocal.gif"));
  boton.setActionCommand("espectador");
  boton.setEnabled(true); //siendo actuador, puede pasarse a espectador
 }
 else{
  boton.setText("  Actuador");
  boton.setIcon(crearIcono("iconos/controlRemoto.gif"));
  boton.setActionCommand("actuador");
  boton.setEnabled(permiso);
 }
}
```

6.3.4 El gestor USB

A diferencia del código escrito para las aplicaciones *servidor* y el *cliente* (escritas en Java), esta aplicación está construida utilizando el lenguaje C/C++.

➢ Control del pipe

❑ **La clase Pipe**

Esta clase se ocupa de crear y mantener el pipe de comunicaciones con la aplicación *servidor*. Una vez creado el pipe, permanece a la espera de peticiones de movimiento por parte del servidor; una vez ejecutadas estas operaciones (o detectado un error), se enviará una respuesta a través del pipe.

▪ **class Pipe**

Definición de la clase.

- **Pipe()**

Constructor de la clase. Inicializa el pipe, y permanece a la espera (indefinidamente) de que alguna aplicación se conecte al mismo.

```
Pipe::Pipe()
{
        if (IniciaPipe() == true)
                ConectaPipe(INFINITE);
}
```

- **~Pipe()**

Destructor de la clase. Cancela las operaciones del *pipe* y libera sus recursos asociados.

```
Pipe::~Pipe()
{
        CancelIo(hPipe);
        DisconnectNamedPipe(hPipe);
        CloseHandle(hPipe);
}
```

- **bool leer(char *bufferRead,DWORD bufferSize)**

Se encarga de leer una petición del pipe.

```
bool Pipe::leer(char *bufferRead,DWORD bufferSize)
{
        …
        fSuccess = ReadFile(
                hPipe,
                bufferRead,
                bufferSize,
                &cbRead,
                (LPOVERLAPPED) &hOverlapped);
        …
}
```

- **bool escribir(char *bufferWrite,DWORD bufferSize)**

Método empleado para enviar una confirmación al pipe.

```
bool Pipe::escribir(char *bufferWrite,DWORD bufferSize)
{
        …
        fSuccess = WriteFile(
                hPipe,
                bufferWrite,
                bufferSize,
                &cbWritten,
                (LPOVERLAPPED) &hOverlapped);
        …
}
```

- **bool activo()**

Método utilizado para conocer el estado del pipe (activo o no).

- **bool IniciaPipe()**

Método encargado de crear el pipe de comunicaciones. Lo implementa como un *"named pipe"*, es decir, entre otros privilegios, permite ser localizado en el sistema mediante su nombre.

```
bool Pipe::IniciaPipe()
{
      …
       hPipe = CreateNamedPipe(
                lpszPipename,                  // nombre
                PIPE_ACCESS_DUPLEX |     // acceso en ambos sentidos
                FILE_FLAG_OVERLAPPED,
                PIPE_TYPE_MESSAGE |       // pipe de tipo mensaje
                PIPE_READMODE_MESSAGE |
                PIPE_WAIT,                     // bloqueante
                2,                             // número de instancias
                tamanoBuffer,                  // salida
                tamanoBuffer,                  // entrada
                PIPE_TIMEOUT,
                NULL);
       ...
}
```

- **bool ConectaPipe(DWORD delay)**

Permanece a la espera de que se active una instancia del pipe (que una aplicación se conecte al mismo) durante un tiempo equivalente al especificado por la variable *delay*.

```
bool Pipe::ConectaPipe(DWORD delay)
{
  …
  ConnectNamedPipe(hPipe, &hOverlapped);
  …
}
```

- **Hilo de ejecución para el control del pipe**

Se trata de una función que se ejecuta como un hilo de ejecución paralelo al proceso principal de la aplicación. De esta manera evitamos la carga de un proceso bloqueante, como puede ser el control del Pipe, que ocasionaría problemas en la gestión de la interfaz gráfica y el control del dispositivo USB.

La figura 6-37 muestra el esquema de funcionamiento de este hilo.

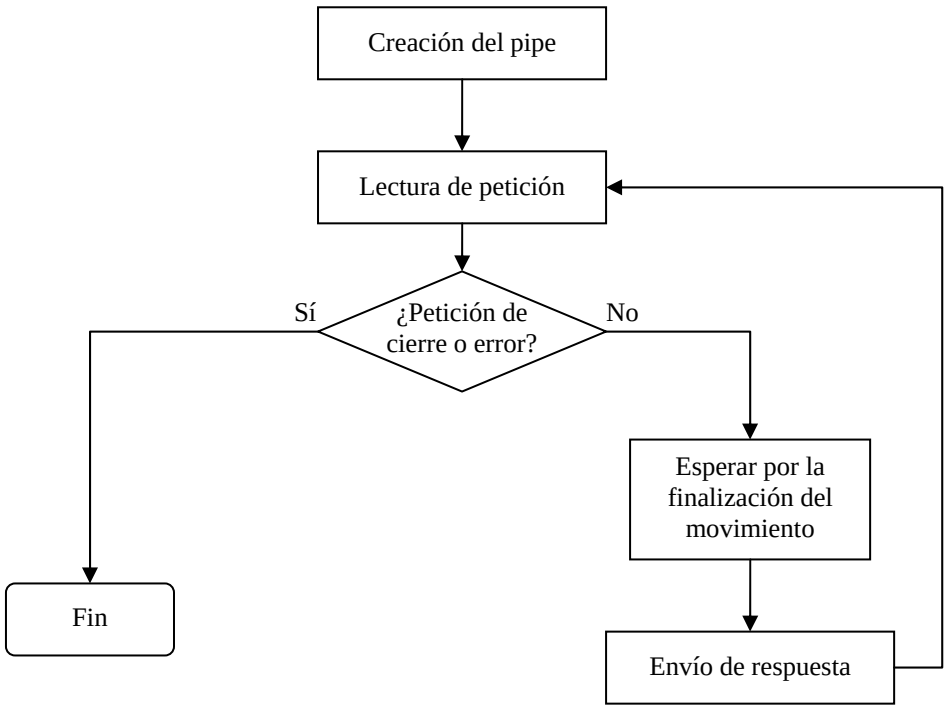


Figura 6-37 : Control del Pipe

- Control de USB e interfaz gráfica

Estas operaciones se realizan en la misma clase. La figura 6-38 muestra el aspecto de la aplicación.

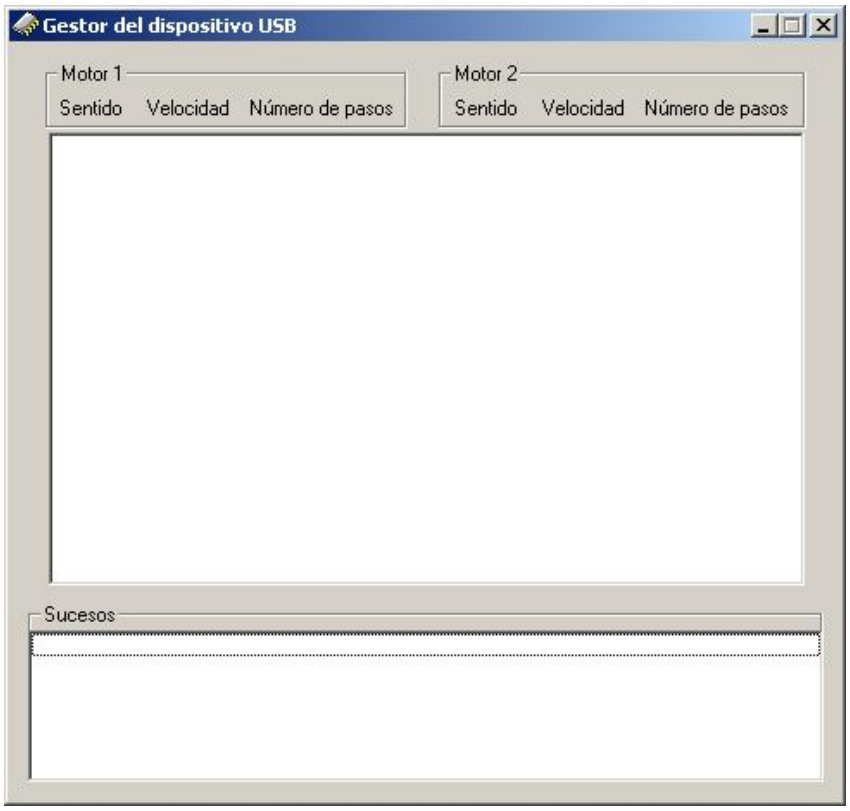


Figura 6-38 : Ventana principal del gestor de USB

- **Control de la interfaz gráfica**

  - **bool OnInitDialog()**

    Método utilizado para inicializar los elementos de la interfaz gráfica, así como el hilo de ejecución que controla el pipe.

  - **LRESULT WindowProc(UINT message,WPARAM wParam,LPARAM lParam)**

    Método principal de toda aplicación Windows, en la que se gestionan todos los mensajes de la misma.

```
LRESULT CUsbhidiocDlg::WindowProc(UINT message,
        WPARAM wParam,LPARAM lParam )
{
```

```
switch (message)
{
case WM_PIPE_ACTIVO:
        {
                if (vector[0] != Salir)
                {
                        WriteReport(); //enviamos la petición
                        ReadReport(); //leemos del dispositivo
                }
                else
                { //cerramos
                        …
                }
        break;
        }

case WM_DESTROY: //cerrar la aplicación
        {
        …
        }
case WM_CLOSE:
case WM_CERRAR:
        {
        …
        }
case WM_PAINT: //dibujar la ventana
        {
        …
        }
case WM_QUERYDRAGICON:
        {
        …
        }
case WM_SYSCOMMAND: //acceso a la barra del sistema
        {
        …
        }
default: //las demás peticiones las egstiona el sistema
return DefWindowProc (message, wParam, lParam);
}
```

```
        return 0;
}
```

- **void OnClose()**

Método utilizado para abandonar la aplicación; en ella se liberan los recursos utilizados.

```
void CUsbhidiocDlg::OnClose()
{
        CloseHandle(DeviceHandle);
        CloseHandle(ReadHandle);
        …
}
```

- **void DisplayData(CString cstrDataToDisplay)**

Muestra datos en el cuadro inferior de la aplicación.

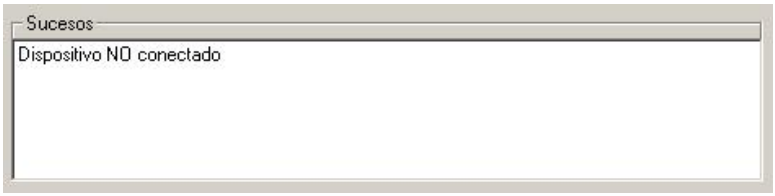


Figura 6-39 : Cuadro inferior mostrando un error de conexión

- **void DisplayReceivedData()**

Muestra los datos de las peticiones enviadas al dispositivo USB (referente a los pasos dados). La figura 6-40 refleja esta situación.

Figura 6-40 : Información del movimiento realizado

- **Control del USB**

  - **bool FindTheHID(unsigned int VendorID,unsigned int ProductID)**

Método utilizado para localizar el dispositivo HID que corresponda con la identificación pasada por parámetros.

Para ello debe seguir una serie de pasos:

```
HidD_GetHidGuid(&HidGuid); //obtener el Globally unique identifier para
//los dispositivos HID

hDevInfo=SetupDiGetClassDevs //obtener el manejador de la información
        (&HidGuid,     //de los dispositivos instalados
        NULL,
        NULL,
        DIGCF_PRESENT|DIGCF_INTERFACEDEVICE);

//obtener información a cerca de la interfaz del dispositivo
Result=SetupDiEnumDeviceInterfaces
        (hDevInfo,
        0,
        &HidGuid,
        MemberIndex,
        &devInfoData);

//obtenemos información más detallada
Result = SetupDiGetDeviceInterfaceDetail
        (hDevInfo,
        &devInfoData,
        detailData,
        Length,
        &Required,
        NULL);

//obtenemos el Manejador del dispositivo, a partir de toda la información obtenida
        DeviceHandle=CreateFile
                (detailData->DevicePath,
                GENERIC_READ|GENERIC_WRITE,
                FILE_SHARE_READ|FILE_SHARE_WRITE,
                (LPSECURITY_ATTRIBUTES)NULL,
```

```
                OPEN_EXISTING,
                0,
                NULL);

    //una vez tenemos el manejador, podremos comparar los valores dados por
    //parámetros para saber si es el dispositivo que buscamos, si no lo es, repetimos
    //el proceso
        Result = HidD_GetAttributes
                (DeviceHandle,
                &Attributes);

        if (Attributes.VendorID == VendorID) //comprobamos si coinciden
                {
                if (Attributes.ProductID == ProductID)
                {
                        ...
                }
                }
```

- **void PrepareForOverlappedTransfer()**

Una vez detectado el dispositivo, lo configuramos para que pueda realizar operaciones (enfocado principalmente a la lectura) en modo solapado, es decir, que sea posible enviar la petición de la operación, y esta se realice en un segundo plano; una vez finalice, avisa con un evento.

- **void ReadReport()**

Lee la respuesta del dispositivo USB.

```
    void CUsbhidiocDlg::ReadReport()
    {
        …
        if (DeviceDetected==FALSE) //la última vez no lo detectó
                DeviceDetected=FindTheHID(0,0);

        if (DeviceDetected==TRUE)
                {
                Result = ReadFile
                        (ReadHandle,
                        InputReport,
```

```
                    Capabilities.InputReportByteLength,
                    &NumberOfBytesRead,
                    (LPOVERLAPPED) &HIDOverlapped);

        //llamamos a un thread para que se quede a la espera de la respuesta del
        //dispositivo
            AfxBeginThread(ControlLecturaUSB,
                                NULL,
                                THREAD_PRIORITY_NORMAL,
                                0,
                                0,
                                NULL);
        }
        else{
            InputReport[0]=5;
            SetEvent(hEventoUSB);
        }
    }
```

- **void WriteReport()**

Envía la petición de movimiento al dispositivo USB.

```
        void CUsbhidiocDlg::WriteReport()
        {
        ...
        if (DeviceDetected==FALSE) //la ultima vez no se detectó
            DeviceDetected=FindTheHID(0,0);

        if (DeviceDetected==TRUE)
            {
            Result = WriteFile
                (DeviceHandle,
                OutputReport,
                Capabilities.OutputReportByteLength,
                &BytesWritten,
                NULL);
                ...
            }
        }
```

- **UINT ControlLecturaUSB(LPVOID pParam)**

Función que espera por la respuesta del dispositivo USB, una vez iniciada en *ReadReport()*. Una vez realizada la misma, o ha sido detectado un error, se indica mediante un evento al hilo que controla el pipe.

```
UINT ControlLecturaUSB( LPVOID pParam )
{
int Result;

Result = WaitForSingleObject(hEventObject, tiempo); //esperamos

switch (Result)
{
case WAIT_OBJECT_0:
   {
   … //recibido correctamente
   }
case WAIT_TIMEOUT:
   {
   …//se ha acabado el tiempo
   }
default:
   {
   …
   }
}
ResetEvent(hEventObject); //lectura (para la próxima)
return 0;
}
```

7. Manual de usuario

En este capítulo se expone el manual de uso de la aplicación desarrollada en este proyecto fin de carrera, el cual permite al usuario conocer su funcionamiento. Proporciona al usuario las nociones necesarias para la utilización satisfactoria de la aplicación.

7.1 Introducción

El conjunto de programas desarrollados en el presente proyecto permiten el control remoto de motores paso a paso conectados al sistema principal utilizando el interfaz USB, mediante el uso de una página Web como interfaz de usuario.

Esta herramienta proporciona un interfaz gráfico amigable y fácil de utilizar, que permite al usuario interactuar de forma eficaz con la aplicación, formada por:

- **Servidor :** Se trata del gestor de los recursos propios de la herramienta (motores y conexiones)
- **Gestor USB :** Intermediario entre el USB y el sistema principal.
- **Cliente:** Utilizada por los usuarios remotos para acceder a la herramienta.

7.2 La aplicación Servidor

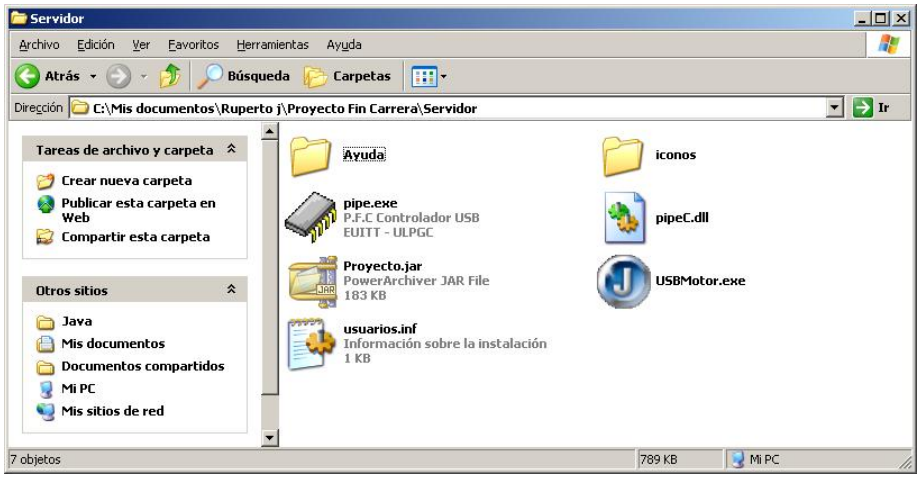


Figura 7-1 : Contenido de la carpeta del servidor

Para ejecutar el programa que gestiona el servidor de la aplicación, se debe localizar el archivo denominado *"USBMotor.exe"*, en el directorio escogido para su instalación. Una

vez abierto (haciendo doble clic sobre el icono que lo representa, ver figura 7-1), se abrirá la pantalla principal de la aplicación, además del gestor USB (la figura 7-2 muestra esta situación).

Centrándonos en el programa servidor de la herramienta, las opciones disponibles inicialmente son las de acceder a la ayuda y cerrar la aplicación (barra superior de botones) o bien identificarse como usuario administrador en el panel central. Para ello, se emplean los cuadros de texto *login* y *password*. Si los datos indicados en estos campos son correctos y corresponden a un usuario con permisos de administrador, se indicará tal incidencia con un mensaje, tras el cual se habilitarán los demás botones de las barras superior e inferior, las cuales se detallan más adelante.

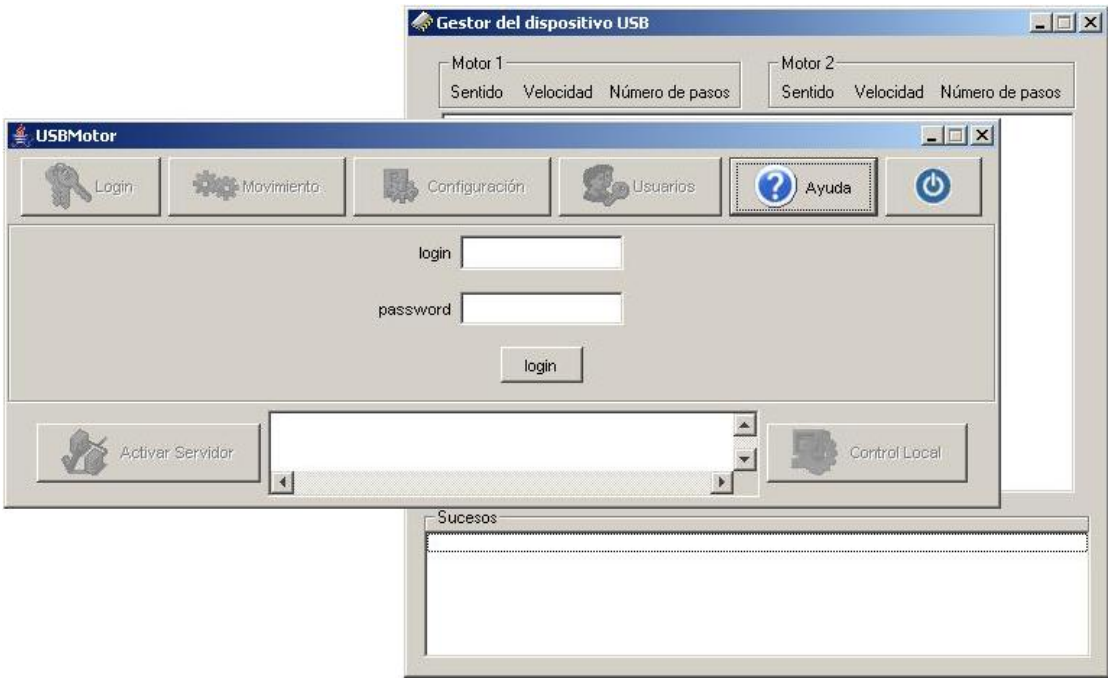


Figura 7-2 : Situación inicial, tras invocar al *servidor*

Si los datos de identificación introducidos no son correctos, o los permisos asignados no son de administrador, el programa lanzará un mensaje informando de estas situaciones (ver figura 7-3) sin alterar el estado de ningún botón de las barras.

Figura 7-3 : Mensaje lanzado cuando la información de identificación no es correcta

Si es la primera vez que se accede a la herramienta tras su instalación, el único usuario conocido es el administrador por defecto, cuya identificación es:

login: ***ADMINISTRADOR***

password: ***ADMIN***

Es recomendable cambiar, al menos, la contraseña de este usuario por defecto, con el objetivo de evitar problemas.

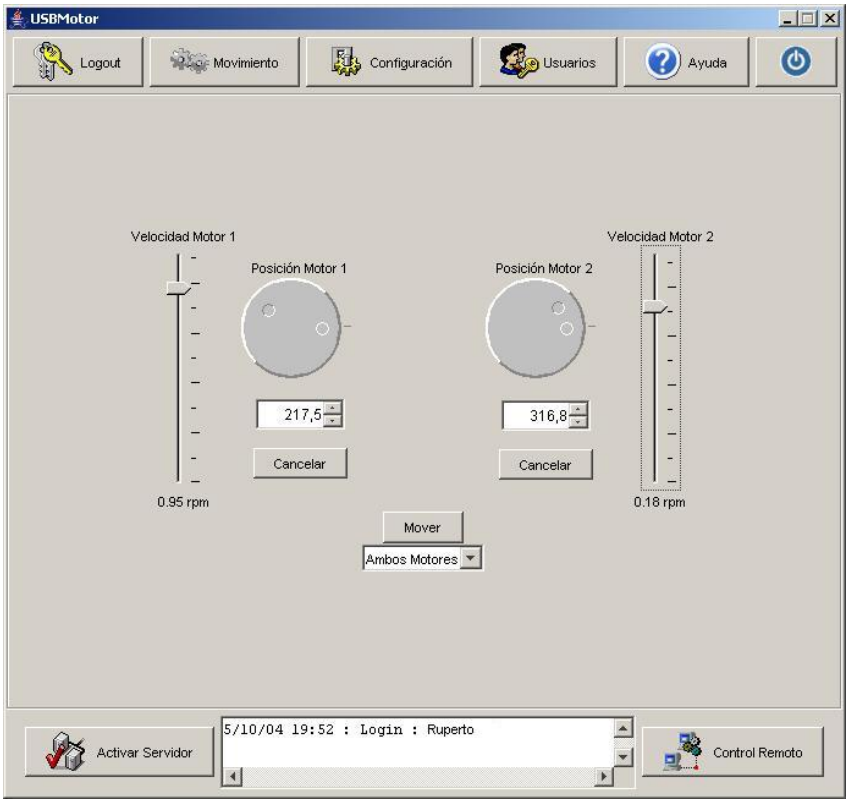


Figura 7-4 : Aspecto general de la aplicación

Dentro del programa *servidor* se pueden diferenciar tres partes principales, como se aprecia en la figura 7-4.

- **Barra superior de botones**

  Se encuentra bajo la barra de título de la aplicación. Contiene los botones que permiten acceder los diferentes paneles que ofrece la aplicación en el cuadro central, además de acceder a la ayuda y cerrar la aplicación.

- **Cuadro central**

Se trata del componente que ocupa la mayor extensión dentro de la aplicación; como su propio nombre indica, se encuentra en el centro de la misma.

Es en este cuadro en el que se desarrollan las actividades principales de la aplicación, ya que es el encargado de alojar los diferentes paneles (identificación, movimiento, configuración y usuarios).

- **Barra inferior de botones**

Se encuentra bajo el cuadro central. Contiene los botones que permiten activar/desactivar las escuchas de la red y el permiso para que los clientes puedan acceder a los motores; además ofrece un cuadro de texto en el que se advierte de las incidencias ocurridas.

7.2.1 Barra superior de botones

Se trata de una barra que contiene los accesos necesarios (botones) para cambiar el panel contenido en el cuadro central, acceder a la ayuda (que utiliza una ventana independiente) y cerrar la aplicación. La figura 7-5 muestra el aspecto de esta barra.

Figura 7-5 : Barra superior de botones

Los cuatro primeros botones (de izquierda a derecha), es decir, *login/logout*, *movimiento*, *configuración* y *usuarios* son los que producen el cambio de panel en el cuadro central. Si se encuentran activos (sólo lo estarán durante la sesión de un administrador), al accionarlos se dará paso al panel indicado por su texto descriptivo, en el cuadro central.

Entre este grupo merece especial atención el primer botón, el que corresponde a la *identificación* (login/logout), ya que según haya o no un usuario administrador activo, cambia su texto descriptivo y estado. Inicialmente, permanece inactivo y con el texto "login", invitando al usuario a identificarse; en esta situación todos los botones que pueden cambiar de panel se encontrarán inactivos, ya que no hay ningún usuario con permisos de administrador. Una vez que un usuario se identifique como administrador,

su texto cambiará por el de “logout” indicando cuál es la posible acción a realizar accionando el botón. En cualquier caso, el estado del botón permanecerá inactivo hasta que se decida cambiar de panel, ya que tras la identificación, el usuario permanecerá en el panel de login, desde el cuál podrá efectuar de igual forma la salida de la sesión. Una vez que esté activo, accionarlo supondrá la aparición de un diálogo de confirmación de abandono de la sesión. La figura 7-6 muestra el estado inicial del botón.

Figura 7-6 : Botón de identificación desactivado

Luego encontramos el botón de ayuda, que desplegará la ventana correspondiente a dicha acción.

7.2.2 Cuadro Central

Se trata de la parte más importante de la aplicación, ya que su función es la de dar cabida a cada uno de los paneles que componen la aplicación.

7.2.2.1 Panel de identificación

Este panel es el primero que encuentra el usuario cuando inicia la aplicación, ya que en él se realiza la identificación del usuario que da acceso a los recursos ofrecidos por la aplicación. La figura 7-7 muestra el aspecto de este panel.

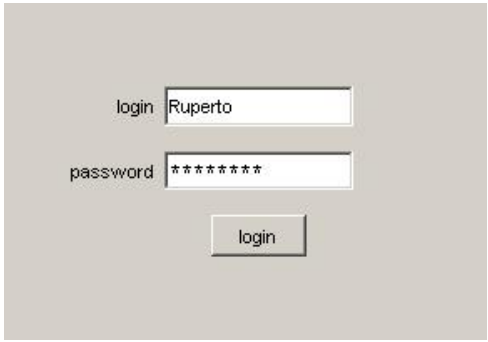


Figura 7-7 : Panel de identificación

Cuenta con dos entradas de texto (login y password) y un botón que procede a la identificación (identificado con el texto “login”). El usuario debe rellenar correctamente ambos campos para poder acceder a la aplicación. Cualquier incidencia será indicada mediante diálogos de mensajes, ya sea para señalar un error en los datos

facilitados (o la ausencia de alguno) como para informar del acceso correcto a la aplicación.

Ya que el servidor permite el control sobre toda la herramienta, además de configurar sus parámetros, sólo se permitirá el acceso a aquellos usuarios registrados como "administradores"; cualquier otro será rechazado por la aplicación (ver figura 7-8).

Figura 7-8 : Mensaje de rechazo de usuario

7.2.2.2 Panel de movimiento

Tal vez se trate del panel más interesante, en cuanto a su atractivo, ya que es el panel que permite actuar sobre los motores (es decir, moverlos). La figura 7-9 muestra el aspecto de este panel.

Sólo es posible acceder a los elementos de este panel si se ha seleccionado el modo *"control local"* en la barra inferior de botones; en caso contrario, los componentes quedarán deshabilitados, por lo que tan sólo se pueden observar los cambios que los clientes vayan produciendo en la posición y velocidad de los motores.

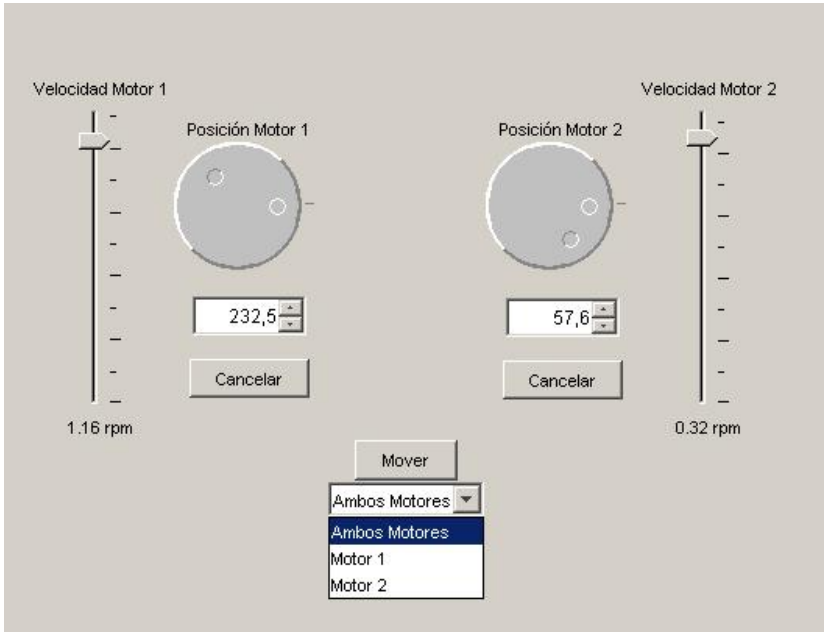


Figura 7-9 : Panel de movimiento

Para controlar cada motor, este panel cuenta con los siguientes componentes:

- **Dial** : Componente situado bajo la etiqueta “Posición Motor 1” o “Posición Motor 2”. Se trata de un círculo con dos cursores. El cursor de color blanco (círculo pequeño dentro del dial) se utiliza para indicar la posición en la que se encuentran actualmente los motores; no puede ser alterado directamente por el usuario (con el ratón, por ejemplo), sino que su posición es asignada como resultado de un movimiento o previa indicación en el panel de configuración.

  El otro círculo (3D) es un cursor móvil (arrastrando el ratón), el cual indica la posición a la que se desea que el motor se desplace en la próxima solicitud de movimiento.

- **Barra deslizante :** Componente situado bajo la etiqueta “Velocidad Motor 1” o “Velocidad Motor 2”. Arrastrando el cursor del que dispone, es posible indicar la velocidad a la que se producirá el movimiento deseado. En la parte inferior del componente se indica la velocidad estimada que supone la posición actual del cursor.

- **Cuadro Numérico :** Permite indicar la posición que se desea alcanzar con el motor, introduciendo directamente el valor numérico de la misma (en grados). Si el usuario introduce un valor que no es posible alcanzar con la configuración actual, el cuadro aproxima al valor más cercano posible; esta situación se da con los motores paso a paso, ya que al tener determinado el ángulo de cada paso, ciertos valores angulares no podrán ser alcanzados.

  Además, el cuadro cuenta con dos botones que se utilizan para aumentar o disminuir el valor almacenado en intervalos de un paso. También es posible mantener pulsados estos botones (uno u otro) durante cierto tiempo, produciéndose una variación progresiva del valor indicado por el cuadro.

  La introducción de un valor en este cuadro supone un cambio en la posición del cursor móvil, así como la variación de este cursor afecta al valor del cuadro; es decir, ambos se encuentran estrechamente relacionados, indicando el cuadro el ángulo en que se encuentra el cursor.

- **Botón Cancelar :** Utilizado cuando se considera que se ha producido un error al situar el cursor móvil ó introducir el ángulo de giro, antes de accionar el botón de movimiento (tras lo cual, la única marcha atrás pasa por retroceder el mismo número de pasos movidos). Una vez accionado el botón de cancelar, el cursor móvil se situará sobre el fijo, indicando que se apunta a la posición actual (no hay movimiento) y el cuadro numérico contendrá el valor angular de dicha posición.

El panel cuenta con dos elementos más:

- **Lista de motores :** Lista desplegable que permite elegir qué motor será movido (motor 1, motor 2 o ambos motores). De esta manera, si se decide mover un único motor, los cambios solicitados sobre el otro no tendrán efecto.

- **Botón Mover :** Se encarga de enviar la petición de movimiento (señalada por los componentes del motor).

  Este botón sólo se habilita cuando se produce algún cambio en la posición del motor indicado en la *lista de motores* (cuando ésta indica sólo uno), o cualquier cambio de orientación si indica *"ambos motores"*.

7.2.2.3 Panel de configuración

En este panel es posible definir todos los parámetros utilizados por la herramienta, es decir, las características de los motores y las peculiaridades de las conexiones de red. Este panel en realidad es el contenedor de otros dos paneles (que se verán a continuación), pero incluyendo dos botones para poder elegir cuál de ellos activar (ver figura 7-10).

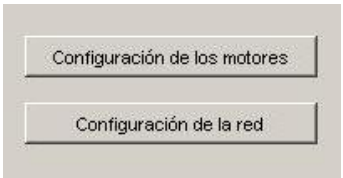


Figura 7-10 : Botones del panel de configuración

7.2.2.3.1 Panel de configuración de los motores

Permite la configuración de los parámetros de los motores, así como el tamaño del archivo de registro de movimientos. La figura 7-11 muestra su aspecto, enmarcado en el entorno del panel de configuración.

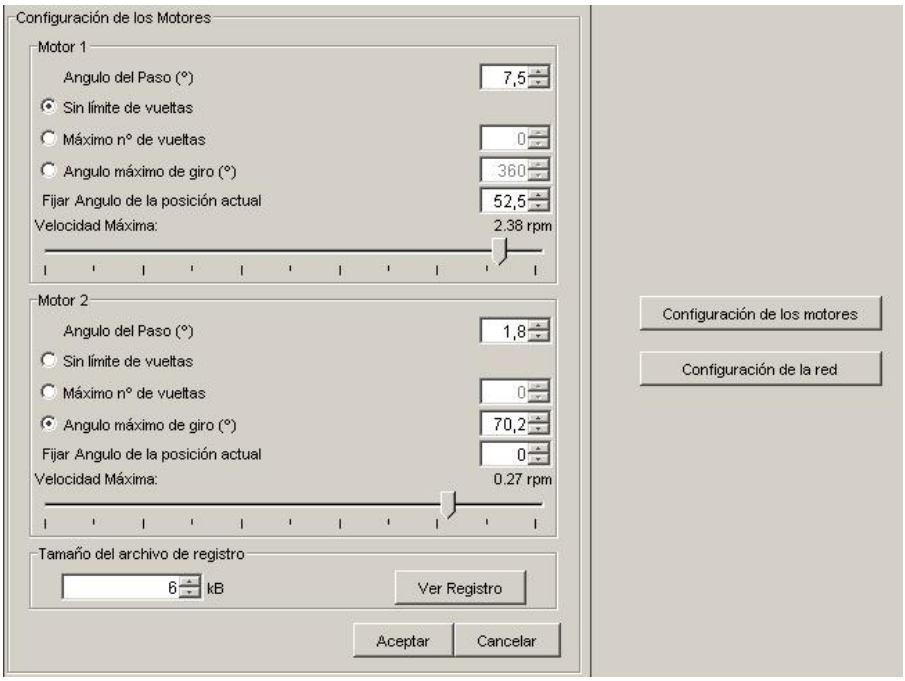


Figura 7-11 : Panel de configuración de los motores

En primer lugar encontramos dos paneles en los que configurar los parámetros de cada motor. En cada uno de ellos podemos encontrar componentes que definan:

- **Angulo de cada paso :** Se trata de una de las características fundamentales de todo motor paso a paso, ya que define la resolución del mismo y por tanto, qué posiciones es capaz de alcanzar.

- **Restricciones :** Esta herramienta posibilita restringir el movimiento de los motores a un rango (sólo una zona) o a un determinado número de vueltas (por ejemplo, si hay peligro de que se enrolle un cable). Estas limitaciones son indicadas mediante la elección de uno de los siguientes botones:

  - **Sin límite de vueltas :** Se trata de la opción sin restricciones, es decir, no hay limitaciones al movimiento.

  - **Máximo número de vueltas :** Esta restricción no permite que el motor gire más de un determinado número de vueltas; este valor deberá ser especificado, como un número entero, en el cuadro numérico que se encuentra a su derecha.

- **Angulo máximo de giro :** Seleccionando esta opción es posible indicar el ángulo que es capaz de describir el motor, a partir del "0" de referencia, en el cuadro numérico que se encuentra a su derecha.

- **Angulo de la posición actual :** Con esta opción es posible indicar la orientación a la que corresponde la posición actual del motor.

- **Velocidad máxima :** Barra deslizante que permite limitar la velocidad máxima a la que podrán ser movidos los motores. En su parte derecha se indica la velocidad correspondiente a la posición del cursor; esta velocidad no sólo depende del propio cursor, sino también del ángulo de cada paso; un motor cuyo ángulo de paso sea grande, será capaz de dar una vuelta mucho más rápido que uno con un ángulo de paso menor, asumiendo para ambos el mismo tiempo entre paso y paso.

En tercer lugar encontramos el panel del registro de movimientos, en el que es posible definir el tamaño máximo del archivo (en kB) utilizando el cuadro numérico, o ver el contenido del mismo accionando el botón *"Ver Registro"*; la figura 7-12 muestra el cuadro en el que se visualiza este registro.

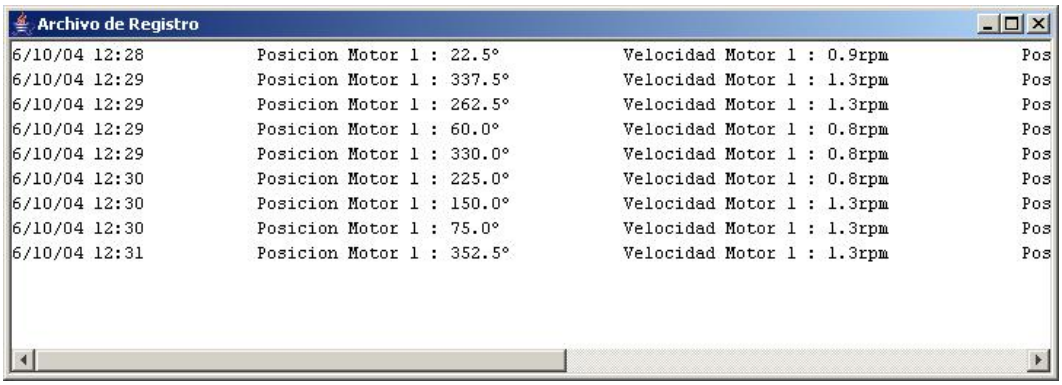


Figura 7-12 : Cuadro del registro de usuarios

Durante la ejecución de la aplicación servidora debe accederse al registro utilizando este método, ya que de esta manera es posible acceder a la versión más actualizada del archivo (pues éste realmente es almacenado en ficheros temporales). Una vez terminada la aplicación, el registro permanecerá almacenado en el archivo "log.txt", accesible mediante cualquier editor de textos.

7.2.2.3.2 Panel de configuración de la red

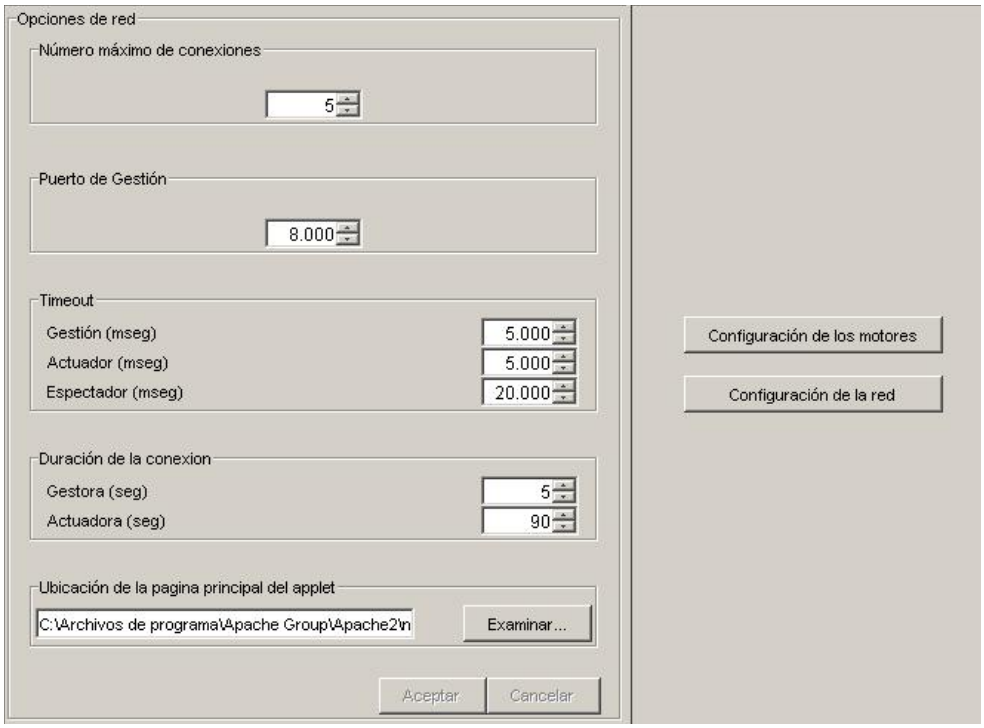


Figura 7-13 : Panel de configuración de la red

Permite la configuración de todos los aspectos relacionados con la red y la interconexión con los clientes. La figura 7-13 muestra el aspecto de este panel, dentro del panel de opciones.

El panel de divide claramente en cinco secciones:

- **Número máximo de conexiones :** Permite, mediante un cuadro numérico, indicar cuántos clientes podrán acceder a la aplicación como máximo.
- **Puerto de gestión :** Permite establecer el puerto a través del que los clientes realizarán las peticiones de acceso a la herramienta.
- **Timeout :** Define, para cada tipo de conexión posible, el tiempo máximo de espera de paquetes. Este tiempo se utiliza para determinar tras cuánto tiempo una conexión se da por perdida (el cliente remoto se ha desconectado sin avisar), permitiendo el acceso a un nuevo cliente.
- **Duración de la conexión :** Con el objetivo de que un usuario no pueda monopolizar las conexiones más importantes (gestión y actuadora), se da la

posibilidad de definir un tiempo máximo para cada una. En cualquier caso, si el valor del cuadro numérico asociado a cada tipo de sesión es *cero*, se indicaría que no hay límite de tiempo en la conexión.

Cuando se cumpla el tiempo establecido, pueden darse diferentes situaciones, dependiendo del tipo de conexión asociado:

- **Gestión :** El usuario será expulsado si no se ha identificado antes de que se cumpla el tiempo indicado, y el sistema volverá a quedar a la espera de las peticiones de un nuevo cliente.

- **Actuadora :** En este caso, el usuario cambia su rol en la aplicación, es decir, pasará a ser un espectador.

- **Ubicación de la página principal del applet :** En este apartado se define el directorio en el que reside el applet. Esta ruta se utiliza para colocar el archivo de configuración de la red y pueda ser obtenido de forma sencilla por los clientes, de tal forma que puedan conocer los parámetros de las conexiones antes de iniciarlas.

  Dispone de dos componentes, una entrada de texto, en la que es posible indicar la ruta manualmente y un botón *"examinar"* que abre un diálogo de selección de archivos como el mostrado en la figura 7-14; utilizando este cuadro, lo que se hace es localizar el archivo HTML en el que se presentará el applet y a partir de esta información se obtendrá la ruta del mismo.

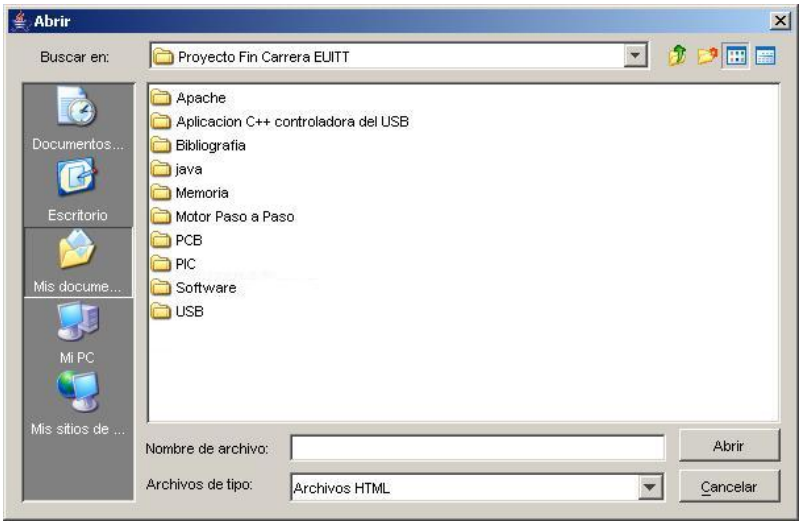


Figura 7-14 : diálogo de selección de archivos

7.2.2.4 Panel de usuarios

Este panel permite tener el control sobre la información de los usuarios y su estado; es el mostrado en la figura 7-15.

El panel se divide verticalmente en dos partes bien diferenciadas. A la izquierda encontramos una tabla con la información del usuario, como es su *login, nombre, permiso, conexión activa, tipo de conexión actual, dirección y último acceso*. A la derecha encontramos un cuadro de botones, cuyas funciones se indican a continuación:

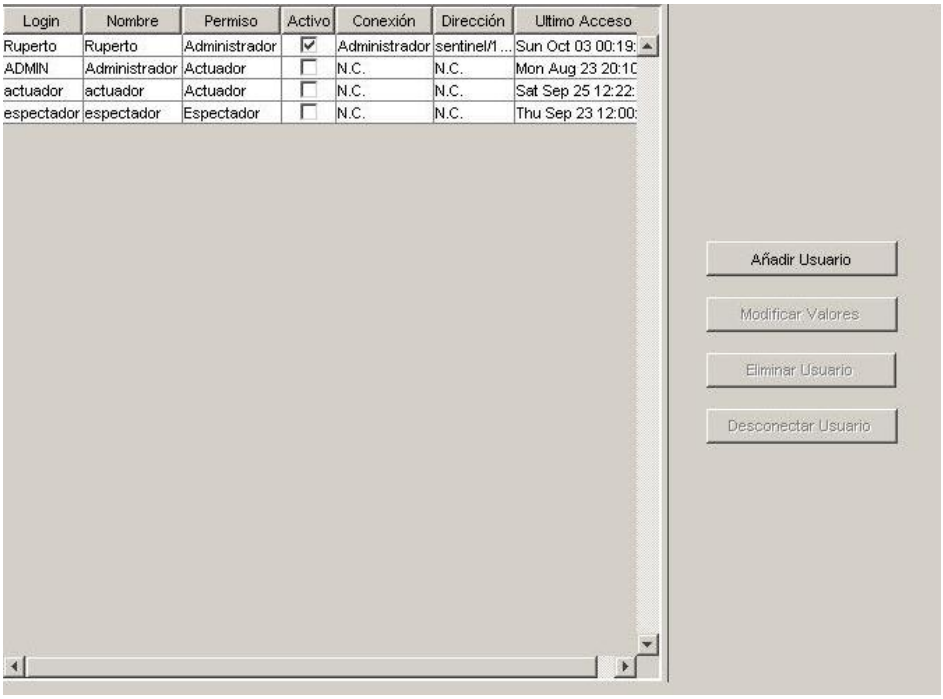


Figura 7-15 : Panel de usuarios

- **Añadir Usuario :** Esta opción siempre se encuentra activa, y permite registrar un nuevo usuario a la aplicación; para ello abre una nueva ventana, como la mostrada en la figura 7-16, en la que indicar los datos del usuario.

  En esta ventana es obligatorio rellenar todos los campos excepto el del nombre. El campo *login* debe cumplir el requisito de no coincidir con el identificador de ningún otro usuario de la aplicación. Además, los dos campos destinados a la contraseña (uno de ellos utilizado como confirmación) deben coincidir en su valor; con el objetivo de hacer más seguro el proceso de introducción de la contraseña, los caracteres en los cuadros de texto destinados al efecto, serán cambiados por asteriscos (*).

Figura 7-16 : Ventana para añadir usuario

- **Modificar Usuario :** Este botón sólo aparecerá habilitado cuando se haya seleccionado un usuario en la tabla. Al accionarlo, se abrirá una nueva ventana, de características similares a las vistas en "añadir usuario", pero con la salvedad de que en este caso los campos ya están inicializados con los valores actuales del usuario seleccionado. De esta manera es posible cambiar los datos, basándose en las mismas reglas descritas en "añadir usuario".

- **Eliminar usuario :** Este botón sólo aparecerá habilitado cuando se haya seleccionado un usuario en la tabla. Despliega un diálogo de confirmación de eliminación; si se contesta afirmativamente, el usuario será eliminado de la tabla y por tanto de la aplicación.

- **Desconectar Usuario :** Este botón sólo aparecerá habilitado cuando se haya seleccionado un usuario en la tabla y éste se encuentre activo. Despliega un diálogo de confirmación de desconexión; si se contesta afirmativamente, el usuario será desconectado, mostrando además esta incidencia en la tabla.

7.2.3 Barra Inferior de Botones

Como muestra la figura 7-17, contiene dos botones, uno a cada lado y un cuadro de texto en el que se muestran las posibles incidencias. Esta barra sólo permanecerá activa durante la sesión de un usuario administrador.

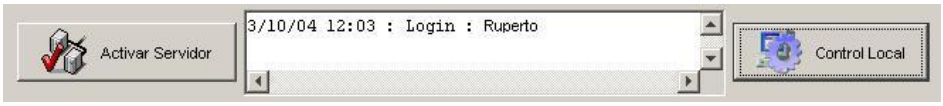


Figura 7-17 : Barra Inferior de botones

El botón de la izquierda es utilizado para activar/desactivar la escucha de peticiones realizadas por los clientes (*Activar/Desactivar Servidor*). Mientras el programa permanece a la espera de peticiones remotas, las opciones de configuración (motores y red) quedan bloqueadas a los valores asignados, los cuales podrán ser modificados cuando no hayan conexiones de red activas (no se espere por peticiones del cliente).

Por otro lado, el botón de la derecha es empleado para indicar si el control de los motores será local a la máquina que ejerce de servidor (es decir, por parte del usuario administrador) o bien se posibilita el acceso a los usuarios remotos.

La figura 7-18 muestra los posibles estados que pueden presentar ambos botones. El texto descriptivo de cada botón indica la acción a realizar (por tanto, actualmente se encuentra activa la contraria)

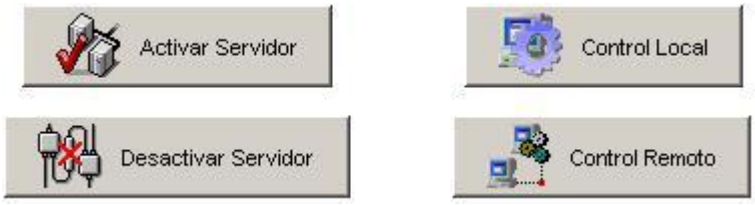


Figura 7-18 : Posibles estados de los botones de la barra inferior

7.2.4 Ayuda

Figura 7-19 : Ventana de selección de ayuda

El acceso a la ayuda se realiza a través del botón etiquetado como *"Ayuda"* en la barra superior de botones; al accionarlo se abrirá una nueva ventana, como la mostrada en la figura 7-19, que permite elegir entre el acceso a la ayuda propiamente dicha, o a la información *acerca de* la aplicación. Una vez seleccionada una opción, se cerrará la ventana de la ayuda.

Figura 7-20 : Acerca de... programa servidor

La ventana *Acerca de...* es la mostrada en la figura 7-20, mientras que la *Ayuda de la aplicación* muestra una nueva ventana como la mostrada en la figura 7-21.

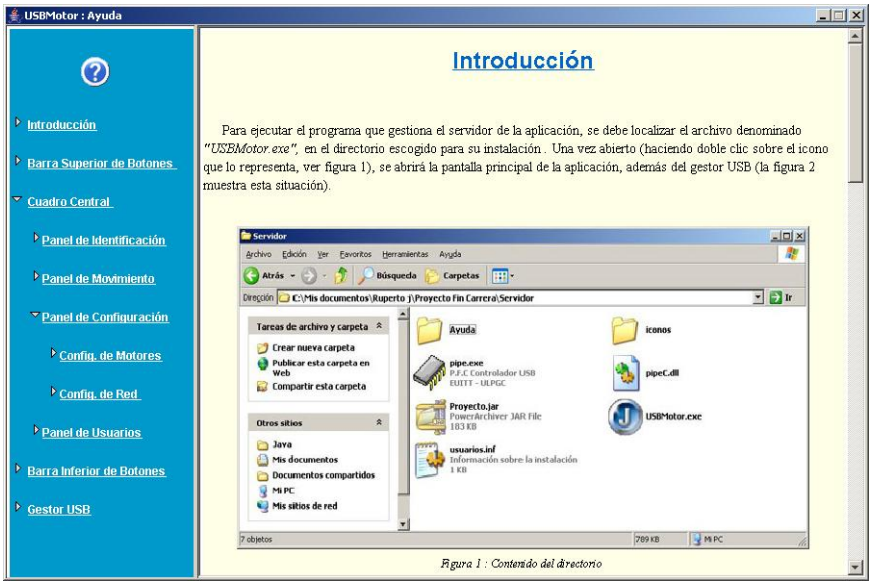


Figura 7-21 : Ventana de ayuda del servidor

7.3 La aplicación Gestor USB

Esta aplicación es la que hace de intermediario entre el servidor y el sistema USB. De cara al usuario se trata de una aplicación pasiva, ya que se limita a mostrar la información enviada al USB. La figura 7-22 muestra la ventana principal de la aplicación, tras realizar un movimiento sobre ambos motores.

En la aplicación se diferencian claramente dos cuadros:

- **Cuadro Superior :** Se utiliza para mostrar al usuario los datos enviados al dispositivo, en un formato más manejable que la verdadera trama transmitida. Para cada motor se indica el sentido del giro (izquierda/derecha), la velocidad (valor entre 0 y 255) y finalmente el número de pasos dados.

- **Cuadro Inferior :** Se trata de un cuadro de estado. Si el dispositivo se encuentra conectado indica esta incidencia, así como toda la información relativa al mismo. Si el dispositivo estuviera desconectado, indicaría esta situación mediante el correspondiente mensaje textual.

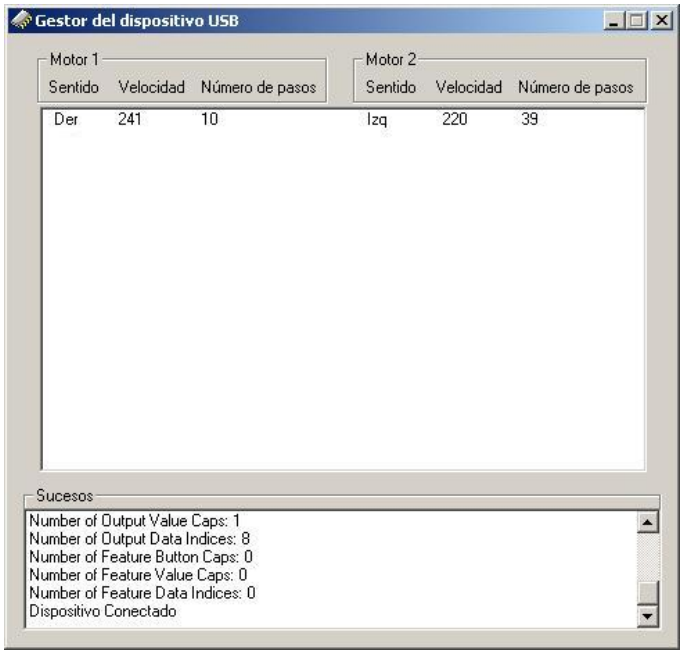


Figura 7-22 : Gestor del dispositivo USB

No es necesario iniciar ni finalizar la aplicación de forma manual, ya que de esto se encarga el propio servidor. Si de forma accidental el gestor de USB fuera cerrado por el usuario, la aplicación servidora se encargaría de iniciarlo cuando lo necesitase.

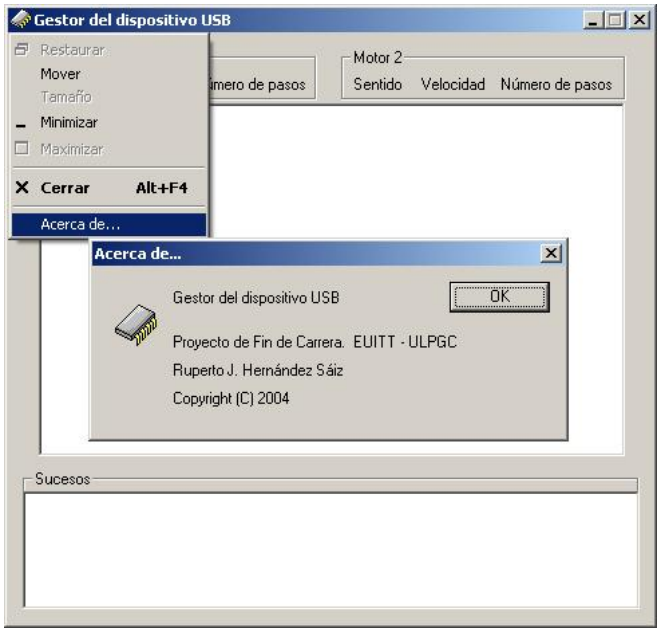


Figura 7-23 : Acerca de... gestor USB

El único menú accesible es el de la barra de estado, en el que además de realizar las operaciones estándar de todas las ventanas Windows, se podrá acceder a información acerca de la aplicación (ver figura 7-23)

7.4 La aplicación Cliente

Una vez accedemos a la pagina en la que reside el applet, lo primero que aparece es lo mostrado en la figura 7-24.

Las opciones disponibles inicialmente son las de acceder a la ayuda (barra superior de botones) e identificarse como usuario en el panel central, empleando para ello los cuadros de texto *login* y *password*. Tanto si los datos indicados en estos campos son correctos y corresponden a un usuario registrado como si no, se indicará la incidencia mediante un mensaje, según el caso.

Como muestra la figura 7-24, al igual que ocurría en el servidor, esta aplicación se encuentra dividida básicamente en tres zonas:

- **Barra superior de botones**

  Se encuentra en la zona superior del área ocupada por el applet. Básicamente contiene dos botones: identificación *(login/logout)* y ayuda.

- **Cuadro central**

  Se trata del componente que ocupa la mayor extensión dentro de la aplicación; como su propio nombre indica, se encuentra en el centro de la misma.

  Es en este cuadro en el que se desarrollan las actividades principales de la aplicación, ya que es el encargado de alojar los diferentes paneles (identificación *(login/logout)* y panel de movimiento).

- **Barra inferior de botones**

  Se encuentra bajo el cuadro central. Contiene el botón que permite el cambio de modo y un cuadro de texto en el que se advierte de las incidencias ocurridas.

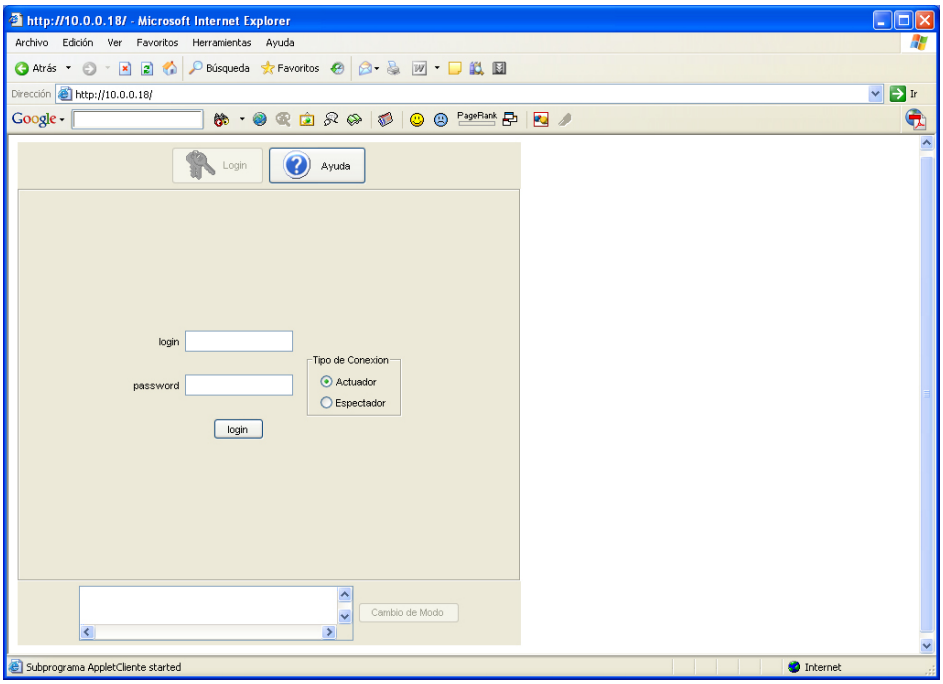


Figura 7-24 : Aspecto general del applet, dentro de la página web

7.4.1 Barra superior de botones

Esta barra cuenta con tan solo dos botones, los cuales se detallan a continuación (ver figura 7-25):

- **Botón de identificación :** Inicialmente se encuentra inactivo y con el texto *"login"*, indicando la invitación al usuario a identificarse. Una vez que el usuario se ha identificado, este botón pasa a estar activo, y con el texto *"logout"*; de esta manera, si es accionado, se desplegará un mensaje de confirmación, al que si se contesta afirmativamente, supondrá el abandono de la sesión.

- **Botón de ayuda :** Muestra la ventana correspondiente a la selección de temas de ayuda.

Figura 7-25 : Barra superior de botones del applet

7.4.2 Cuadro Central

Se trata de la parte más importante de la aplicación, ya que su función es la de dar cabida a los dos paneles que componen la aplicación.

7.4.2.1 Panel de identificación

Este panel es el primero que se encuentra el usuario cuando inicia la aplicación, ya que en él se realiza la identificación de los usuarios. La figura 7-26 muestra el aspecto de este panel.

Cuenta con dos entradas de texto (login y password) y un botón que procede a la identificación. El usuario debe rellenar correctamente ambos campos para poder acceder a la aplicación. Cualquier incidencia será indicada mediante diálogos de mensajes, ya sea para señalar un error en los datos facilitados (o la ausencia de alguno) como para informar del acceso correcto a la aplicación.

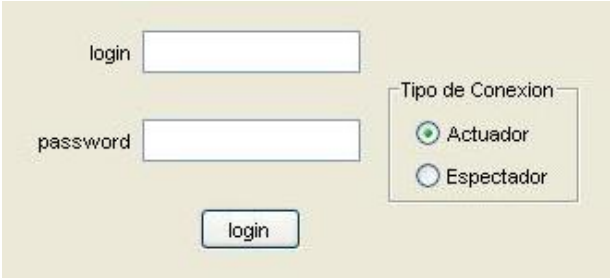


Figura 7-26 : Panel de identificación del applet

Además, el panel cuenta con dos botones adicionales que permiten especificar el tipo de conexión inicial. Si el usuario está registrado en la aplicación, pero no tiene permisos suficientes o el recurso actuador está ocupado, le será asignado el modo espectador; por otro lado, si la conexión actuadora está disponible, es la deseada y el usuario está registrado como actuador, será esta la sesión asignada.

7.4.2.2 Panel de movimiento

Tal vez se trate del panel más interesante, en cuanto a su atractivo, ya que es el panel que permite actuar sobre los motores (es decir, moverlos). La figura 7-27 muestra el aspecto de este panel.

Sólo es posible acceder a los elementos de este panel si actualmente la sesión asignada es la actuadora; en caso contrario, los componentes quedarán deshabilitados, por lo que tan sólo se pueden observar los cambios que el usuario actuador vaya produciendo en la posición y velocidad de los motores.

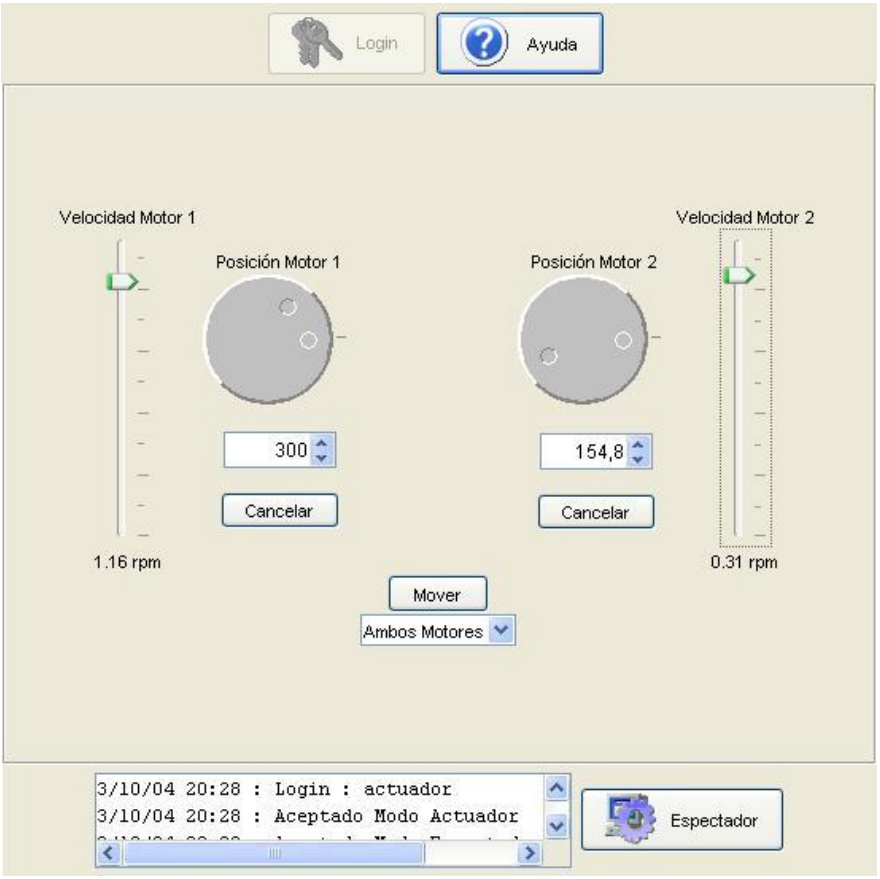


Figura 7-27 : Panel de movimiento

Para controlar cada motor, este panel cuenta con los siguientes componentes:

- **Dial** : Componente situado bajo la etiqueta “Posición Motor X”. Se trata de un círculo con dos cursores. El cursor de color blanco (circulo pequeño dentro del dial) se utiliza para indicar la posición en la que se encuentran actualmente los motores; no puede ser alterado directamente por el usuario (con el ratón, por ejemplo), sino que su posición es asignada como resultado de un movimiento o previa indicación en el panel de configuración.

  El otro círculo (3D) es un cursor móvil (arrastrando el ratón), el cual indica la posición a la que se desea que el motor se desplace.

- **Barra deslizante :** Componente situado bajo la etiqueta “Velocidad Motor X”. Arrastrando el cursor del que dispone, es posible indicar la velocidad a la que se producirá el movimiento deseado. En la parte inferior del componente se indica la velocidad estimada que supone la posición actual del cursor.

- **Cuadro Numérico :** Permite indicar la posición que se desea alcanzar con el motor, introduciendo directamente el valor numérico de la misma (en grados). Si el usuario introduce un valor que no es posible alcanzar con la configuración actual, el cuadro aproxima al valor más cercano posible; esta situación se da con los motores paso a paso, ya que al tener determinado el ángulo de cada paso, ciertos valores angulares no podrán ser alcanzados.

  Además, el cuadro cuenta con dos botones que se utilizan para aumentar o disminuir el valor almacenado en intervalos de un paso. También es posible mantener pulsados estos botones (uno u otro) durante cierto tiempo, produciéndose una variación progresiva del valor indicado por el cuadro.

  La introducción de un valor en este cuadro supone un cambio en la posición del cursor móvil, así como la variación de este cursor afecta al valor del cuadro; es decir, ambos se encuentran estrechamente relacionados, indicando el cuadro el ángulo en que se encuentra el cursor.

- **Botón Cancelar :** Utilizado cuando se considera que se ha producido un error al situar el cursor móvil ó introducir el ángulo de giro, antes de accionar el botón de movimiento (tras lo cual, la única marcha atrás pasa por retroceder el mismo número de pasos movidos). Una vez accionado el botón de cancelar, el cursor móvil se situará sobre el fijo, indicando que se apunta a la posición actual (no hay movimiento) y el cuadro numérico contendrá el valor angular de dicha posición.

El panel cuenta con dos elementos más:

- **Lista de motores :** Lista desplegable que permite elegir qué motor será movido (motor 1, motor 2 o ambos motores). De esta manera, si se decide mover un único motor, los cambios solicitados sobre el otro no tendrán efecto.

- **Botón Mover :** Se encarga de enviar la petición de movimiento (señalada por los componentes del motor).

  Este botón sólo se habilita cuando se produce algún cambio en la posición del motor indicado en la *lista de motores* (cuando ésta indica sólo uno), o cualquier cambio de orientación si indica *"ambos motores"*.

7.4.3 Barra Inferior de Botones

Esta barra cuenta con un cuadro de texto en el que se registrarán las incidencias del cliente y un botón que permite el cambio en el modo de conexión. La figura 7-28 muestra el aspecto de esta barra.

Figura 7-28 : Barra inferior de botones

Cuando un usuario se ha identificado como actuador (y ha sido aceptado como tal), el botón de esta barra le permite, en todo momento, cambiar su modo de conexión al de espectador. Si por el contrario la sesión actual es de espectador, éste botón sólo permanecerá activo y con el texto *"Actuador"*, cuando no haya ningún usuario identificado como tal en la herramienta; de esta manera, si es accionado, el cliente envía una petición de cambio de modo (para ser actuador) al servidor, la cual podrá ser aceptada o no (en cualquier caso, el cliente lanzará un mensaje textual informando del resultado).

7.4.4 Ayuda

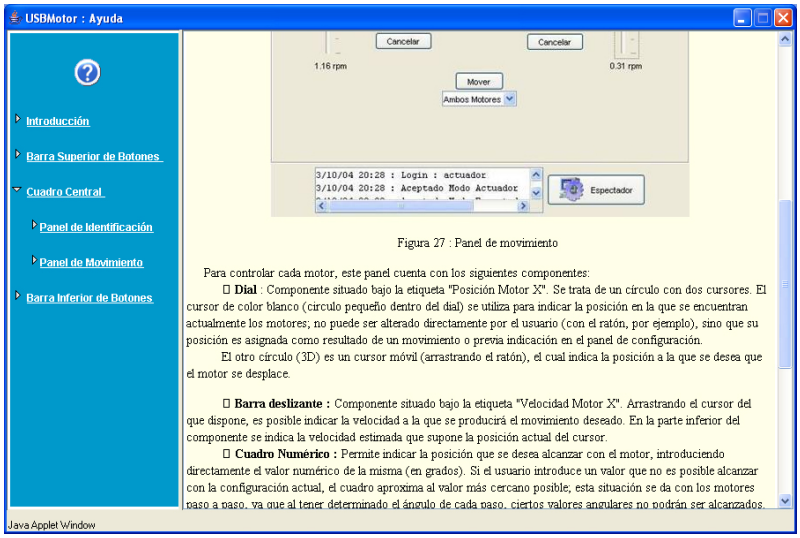


Figura 27 : Panel de movimiento

Para controlar cada motor, este panel cuenta con los siguientes componentes:

- **Dial** : Componente situado bajo la etiqueta "Posición Motor X". Se trata de un círculo con dos cursores. El cursor de color blanco (circulo pequeño dentro del dial) se utiliza para indicar la posición en la que se encuentran actualmente los motores; no puede ser alterado directamente por el usuario (con el ratón, por ejemplo), sino que su posición es asignada como resultado de un movimiento o previa indicación en el panel de configuración.
  El otro círculo (3D) es un cursor móvil (arrastrando el ratón), el cual indica la posición a la que se desea que el motor se desplace.
- **Barra deslizante :** Componente situado bajo la etiqueta "Velocidad Motor X". Arrastrando el cursor del que dispone, es posible indicar la velocidad a la que se producirá el movimiento deseado. En la parte inferior del componente se indica la velocidad estimada que supone la posición actual del cursor.
- **Cuadro Numérico :** Permite indicar la posición que se desea alcanzar con el motor, introduciendo directamente el valor numérico de la misma (en grados). Si el usuario introduce un valor que no es posible alcanzar con la configuración actual, el cuadro aproxima al valor más cercano posible, esta situación se da con los motores paso a paso, ya que al tener determinado el ángulo de cada paso, ciertos valores angulares no podrán ser alcanzados.

Figura 7-29 : aspecto de la ventana de *ayuda de la aplicación*

El acceso a la ayuda se realiza a través del botón etiquetado como *"Ayuda"* en la barra superior de botones; al accionarlo se abrirá una nueva ventana, que permite elegir entre el acceso a la *ayuda de la aplicación* propiamente dicha, o a la información *acerca de* la aplicación. La figura 7-29 muestra el aspecto de la ventana de *ayuda de la aplicación.*

8. Conclusiones

En el presente Proyecto Fin de Carrera se ha diseñado y realizado un sistema hardware/software capaz de controlar remotamente vía Web la orientación de dos motores paso a paso unipolares. Este sistema se ha dividido en cuatro grandes partes: la aplicación del cliente, la del servidor, el gestor del dispositivo USB y el prototipo hardware.

En cuanto a la aplicación *cliente* se ha conseguido crear una interfaz amigable, fácil de utilizar y capaz de ejecutarse en un navegador Web. Para ello ha sido implementado utilizando el lenguaje de programación Java, que permite incorporar de forma sencilla potentes interfaces gráficas utilizando sus componentes Swing, además de ofrecer la posibilidad de incorporar la aplicación a una página Web como un applet Java. Esta interfaz permite a los usuarios registrados acceder a los recursos de la herramienta, previa identificación mediante un nombre de usuario y una clave.

Como requisito para la ejecución de la aplicación *cliente*, encontramos la necesidad de contar con el plug-in de Java versión 1.4 o superior.

La aplicación *servidor* permite dar servicio a varios clientes simultáneos, pero sólo uno de ellos puede actuar sobre la posición de los motores a la vez, de forma que los demás sólo tendrán acceso a la información referente a la orientación de los motores en cada momento. Cuenta con varios archivos de los que obtener abundante información: la configuración de los motores, las características de las conexiones de red, el listado de los usuarios con sus atributos y un registro en el que se almacena cada uno de los movimientos realizados, detallando hora, número de pasos, velocidad y usuario.

Para el envío de peticiones de movimientos a los motores, el servidor intercambia datos con la aplicación *gestor de USB*, la cual hace de puente entre el *servidor* y el módulo hardware. Esta aplicación no interactúa con el usuario más que para mostrarle información acerca de los datos enviados al dispositivo y al estado del mismo. Esta aplicación sólo puede ser utilizada en sistemas Windows, ya que para lograr el acceso al sistema USB ha sido necesario utilizar la API de dicho entorno.

El módulo hardware hace de interfaz entre el ordenador del servidor y los motores paso a paso a telecomandar; la comunicación entre ambos se realiza a través del sistema USB. En

esta comunicación, el ordenador servidor envía los datos del movimiento deseado y espera por la confirmación de los mismos por parte del dispositivo USB una vez finalizada la operación.

Por todo esto se puede concluir que los objetivos planteados al comienzo de este Proyecto Fin de Carrera, que consistían en la creación de una herramienta capaz de controlar motores remotamente vía Web utilizando el sistema USB para la conexión con la máquina servidora, han sido cubiertos con creces.

En cuanto a conocimientos personales, he aprendido a utilizar el lenguaje de programación multiplataforma Java, con el que hacer interfaces gráficas amigables empleando los componentes de Swing, programar concurrentemente o realizar llamadas a código nativo, además de ampliar conocimientos en cuanto a la programación de aplicaciones cliente/servidor utilizando *sockets*. Por otro lado, he aprendido el funcionamiento del sistema USB y cómo utilizarlo (a nivel de diseñador) desde ambos extremos de la comunicación; mediante llamadas al sistema operativo en el lado del ordenador principal y mediante rutinas específicas en el lado del dispositivo (junto con sus requerimientos eléctricos). A su vez, he reforzado y ampliado mis conocimientos en electrónica digital al diseñar y montar el módulo hardware, con el que he aprendido a diseñar sistemas utilizando microcontroladores como el PIC16C745 y programarlos utilizando un lenguaje ensamblador específico.

8.1 Líneas Futuras

Las líneas futuras que se plantean como posibles ampliaciones o líneas paralelas a este Proyecto Fin de Carrera son las siguientes:

- Hacer que los datos entre el servidor y el cliente viajen cifrados para evitar que algún usuario malintencionado pueda analizar o alterar esa información, máxime cuando se trata de la información de identificación de usuario. Podría hacerse programando *sockets* cifrados o bajo un servidor seguro (*https*).

- Añadir un módulo hardware que registre los pasos dados por cada uno de los motores y la velocidad a la que se producen los mismos, de tal forma que la confirmación del dispositivo incluya la información de lo que los motores hicieron

realmente. Tanto el software del sistema principal como el del microcontrolador están preparados para esta situación, cuando se cuente con el hardware necesario.

- Añadir otros actuadores al microcontrolador, tales como motores de continua, relés,… con los que expandir las capacidades de la herramienta.

- Hacer unas librerías que permitieran acceder al sistema USB independientemente del sistema operativo utilizado.

- Hacer que el servidor Web y la aplicación *servidor* (junto al gestor USB), y por tanto el módulo hardware no tengan por qué estar necesariamente en el mismo ordenador. Para ello se requeriría hacer una pequeña aplicación, que se ejecutara en el ordenador del servidor Web, que hiciera de puente entre el applet y la aplicación *servidor*. Esto permitiría utilizar cualquier servidor Web, incluso aquellos que se ejecutan en otros sistemas operativos que no son Windows.

9. Bibliografía

Libros de Java

- [1] Curso de Java. Autores: Patrick Niemeyer, Jonathan Knudsen. Editorial: Anaya Multimedia. 2000.

- [2] La Biblia de Java. Autor: Laurence Vanhelsuwé. Editorial: Anaya Multimedia. 1997.

Webs de Java

- [3] Tutorial de Java http://www.cica.es/formacion/JavaTut

- [4] The Java Tutorial http://java.sun.com/docs/books/tutorial/

- [5] JavaTM 2 Platform, Standard Edition, v 1.4.2 API Specification http://java.sun.com/j2se/1.4.2/docs/api/

Libros de C/C++

- [6] Aprenda lenguaje ANSI C como si estuviera en primero. Autores: Javier García de Jalón, José Ignacio Rodríguez, José María Sarriegui, Rufino Goñi, Alfonso Brazales, Patxi Funes, Alberto Larzabal, Rubén Rodríguez. Universidad de Navarra. 1998.

- [7] Aprenda C++ como si estuviera en primero. Autores: Javier García de Jalón, José Ignacio Rodríguez, José María Sarriegui, Rufino Goñi, Alfonso Brazales, Patxi Funes, Alberto Larzabal, Rubén Rodríguez. Universidad de Navarra.1998.

Web de la API de Windows

- [8] Microsoft Developer Network msdn.microsoft.com

Libros de USB

- [9] Universal Serial Bus System Architecture. Autores: Don Anderson, Dave Dzatko. 2ª edición. Editorial: Addison-Wesley, 2001.

- [10] USB design by example : a practical guide to building I/O devices. Autor: John Hyde. Editorial: Wiley. 1999.
- [11] Especificación USB 1.1. DEC, IBM, Intel, Microsoft y Compaq.
- [12] Especificación USB 2.0. Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Phillips
- [13] Device Class Definition for Human Interface Devices (HID). Autor: USB Implementers' Forum.
- [14] USB Complete: everything you need to develop custom USB peripherals. Autor: Jan Axelson. Editorial: Lakeview Research. 2001.

Libros de PIC

- [15] Microcontroladores PIC, La solución en un chip. Autores: E. Martín Cuenca, J. Mª Angulo Usategui, I. Angulo Martínez. Editorial: Paraninfo. 1998.
- [16] Datasheet PIC 16C745. Microchip
- [17] USB Firmware User's Guide. Microchip

Webs de Motores Paso a Paso

- [18] Introducción a los motores paso a paso. http://www.redeya.com/electronica/tutoriales/mpp/mpp.htm
- [19] Robotica & µControladores Pic http://www.x-robotics.com/motorizacion.htm#MOTORES%20PaP

Pliego de Condiciones

En el pliego de condiciones se define la manera correcta de instalar las aplicaciones que componen este proyecto, además de explicar la forma de instalar el módulo.

Por otro lado se comenta el software que será necesario tener instalado tanto en el ordenador cliente como en el utilizado para el servidor.

Por último también se indican los requisitos mínimos necesarios, en cuanto a equipos, para que tanto la aplicación del servidor como el applet del cliente puedan ejecutarse de forma aceptable.

1 Instalación del Software

1.1 Instalación de la Máquina Virtual de Java

En el ordenador que vaya a actuar como servidor será necesario instalar el Java Runtime Environment (JRE), o si se desea el paquete completo, es posible instalar el SDK de Java; en ambos casos se incluye la máquina virtual de Java, necesaria para ejecutar las aplicaciones escritas en este lenguaje. La versión 1.4.1 para el sistema Windows 32 se puede descargar de forma gratuita de la página de Sun:

http://java.sun.com/j2se/index.jsp

Para su instalación, siga las instrucciones mostradas por el propio programa de instalación.

Para actuar como servidor, antes de poder ejecutar la aplicación, debe colocar en el PATH del sistema (rutas en las que el sistema debe buscar ejecutables, si no los encuentra en el directorio actual) la ruta donde están los archivos del SDK. En un sistema Windows esto se puede hacer modificando el archivo *autoexec.bat*, añadiendo a la línea que empieza con *SET PATH=*, la ruta de los ejecutables del SDK, separando por ";" las demás rutas incluidas en dicha línea. Por defecto, la ruta donde se instala es *C:\j2sdk1.4.1\bin*, con lo que un *PATH* típico podría quedar como:

```
SET PATH=C:\j2sdk1.4.1\bin;C:\WINDOWS;C:\WINDOWS\COMMAND
```

En el caso de los ordenadores que vayan a utilizar la aplicación *cliente*, se requiere la instalación del plug-in para Java o JRE (Java Runtime Environment) versión 1.4 o superior; para ello, de forma automática, tras abrir la página que incluye el applet, se le invitará a instalarlo desde internet, si es que no se encuentra presente en el sistema.

1.2 Copia de los archivos del servidor

Para poder ejecutar las aplicaciones del servidor, se debe copiar el directorio llamado *Servidor*, incluido en el CD adjunto al proyecto, al disco local del ordenador destinado al servidor. En este directorio se incluye el ejecutable que inicia el programa servidor *USBMotor.exe*, junto a sus archivos de configuración, y el gestor de USB.

1.3 Copia de los archivos del cliente

Los archivos del cliente tienen que copiarse en el ordenador que actúa como servidor, asociando este directorio a los controlados por el servidor Web instalado. El directorio a copiar se llama Cliente y está incluido en el CD adjunto al proyecto, el cual incluye las clases necesarias para la aplicación cliente, incluidas en un archivo *.jar*, la página Web de ejemplo y la ayuda; también contiene un directorio denominado *iconos*, que contiene una serie de imágenes necesarias para el applet.

1.4 Servidor Web

En el ordenador que actúe como servidor de la herramienta, será necesario instalar un servidor Web, de manera que ante la petición de un usuario (a través de un browser), el servidor Web entregue la página en la que reside el applet (programa cliente).

Existe un gran número de servidores Web. Entre los más famosos encontramos el servidor *Apache*, el cual ha sido empleado en este proyecto; se trata de una aplicación escrita en código abierto, que es posible descargar de forma gratuita desde su página Web.

http://httpd.apache.org/

Para instalarlo, basta con seguir las instrucciones mostradas por el propio programa de instalación.

2 Instalación del módulo Hardware

El esquema que se sigue a la hora de conectar el módulo hardware al ordenador que hace de servidor es el que muestra la figura 1-1.

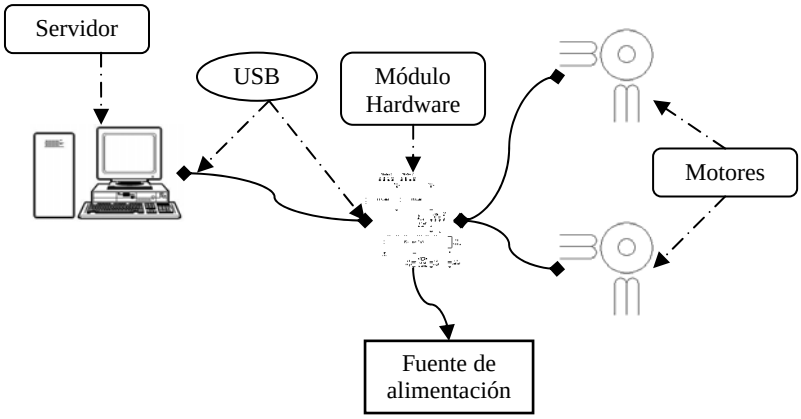


Figura 1-1 : Esquema de conexión del módulo hardware

El módulo dispone de dos conectores:

- Conector USB, para el intercambio de datos con el sistema principal, además de obtener la alimentación del microcontrolador
- Conector de alimentación, utilizado para alimentar los motores paso a paso utilizados en este proyecto.

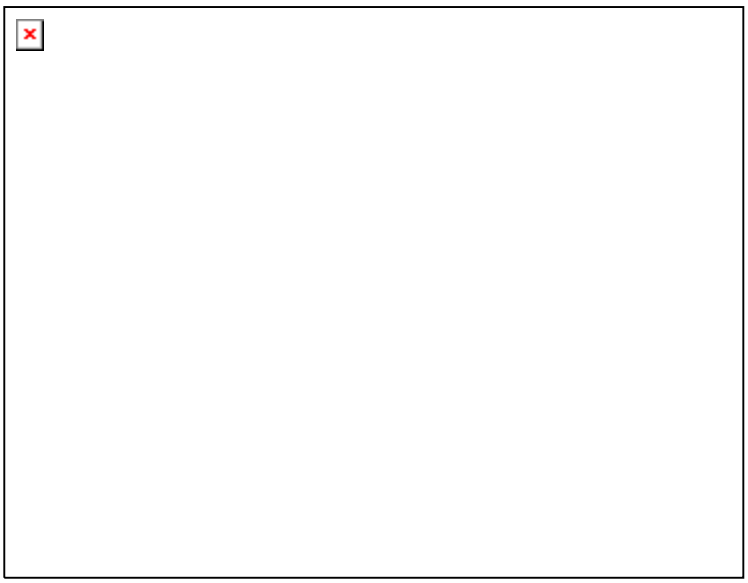


Figura 1-2 : Localización de un puerto USB libre

3 Requerimientos de los ordenadores

Los requerimientos hardware mínimos en el ordenador que actuará como *servidor* son:

- Procesador Intel Pentium a 166MHz
- 64MB de memoria RAM
- Espacio libre en disco 120 MB
- Acceso a Internet
- Unidad de CD-ROM
- Puerto USB libre

A estos requerimientos hay que añadir los necesarios para instalar y ejecutar el servidor Web elegido, como puede ser el Apache Web Server, Xitami, o cualquier otro.

Los requerimientos mínimos de hardware, en el ordenador *cliente* son:

- Procesador Intel Pentium a 166MHz
- 64MB de memoria RAM
- Espacio libre en disco 120 MB
- Acceso a Internet
- Tarjeta gráfica con 2 MB de memoria y una resolución de 800 x 600 píxels.

Para poder ejecutar el applet cliente, es necesario contar con un navegador Web que soporte Java; para ello, el sistema operativo debe ser uno de los que puedan instalar el plug-in de Java versión 1.4.1, como Linux, Solaris o Windows.

4 Contenido del CD

En el directorio raíz del CD adjunto al proyecto encontrará los directorios que se comentan a continuación:

- **Documentos:** Directorio que contiene los archivos en formato "*pdf*" correspondientes al presente proyecto.

- **Servidor:** Este directorio contiene todos los archivos necesarios para que la herramienta funcione correctamente del lado del servidor. Los archivos y directorios que contiene deben ser copiados en su totalidad a una unidad propia del ordenador que hará las veces de servidor.

  Encontramos el ejecutable que abre el programa *servidor* y el gestor USB, los archivos de la ayuda, los iconos utilizados y el fichero inicial de usuarios.

- **Cliente:** Contiene los archivos necesarios para ejecutar la aplicación cliente. Su contenido debe ser copiado en un directorio conocido por el servidor Web que tenga instalado.

- **Datasheets:** Contiene las hojas de características de los componentes utilizados para realizar el módulo hardware.

- **MPLAB:** Programa utilizado para compilar y simular programas escritos en el ensamblador de los microcontroladores PIC de Microchip.

- **IC-PROG:** Programa utilizado para grabar el código ensamblador generado, en la memoria del PIC16C745.

- **SDK 1.4.2:** Paquete de desarrollo de Java, junto a su plug-in para navegadores Web. Es necesario tenerlo instalado tanto en el cliente como en el servidor.

- **Apache:** Servidor Web para Windows.

Presupuesto

En este apartado se hace un estudio del coste económico que ha representado realizar este Proyecto Fin Carrera. Se incluyen en estos costes los correspondientes a lo componentes utilizados para la realización de módulo hardware, el de la mano de obra necesaria para realizar tanto el módulo como las aplicaciones del servidor y del cliente y de costes varios que incluyen folios, tinta, CD-R...; también se incluye la amortización de todo el material inventariable y del software utilizados.

1 Material utilizado

A continuación se muestran los componentes que han sido necesarios para la realización del módulo hardware junto a sus costes.

Componente	Cantidad	Precio Unitario (€)	Total (€)
Placa C.I Positiva	1	8,65	8,65
Zócalo 28 pines	1	0,90	0,90
Zócalo 28 pines	2	0,57	1,14
Condensador 0,1uF	1	0,15	0,15
Condensador 200 nF	1	0,15	0,15
Condensador 33pF	2	0,27	0,54
Resistencia 1,5 kΩ 1/4 W	1	0,05	0,05
Cristal 6MHz	1	1,85	1,85
Driver ULN-2003	2	0,87	1,74
PIC 16C745	1	46,67	46,67
Tornillería	6	0,12	0,71
Motor paso a paso RS440-458 1,8deg.	1	62,41	62,41
Motor paso a paso UBD20N08RNZxx 7,5deg.	1	23,93	23,93
Conector alimentación	1	0,60	0,60
Cable conexión USB	1	17,62	17,62
Total Componentes			167,11

2 Mano de obra

Para calcular el coste de la mano de obra empleada para realizar este proyecto, se ha contabilizado el tiempo que se ha necesitado para su realización, teniendo en cuenta que el trabajo se tarifica según el tiempo empleado.

Las tarifas honorarias se han calculado teniendo en cuenta la publicación de “Baremos orientativos para el cálculo de honorarios”, editada por el COITT (Colegio Oficial de Ingenieros Técnicos de Telecomunicación), con fecha de 1 de Enero de 2004. El citado

documento señala que para calcular los honorarios *(H)* hay que aplicar la siguiente fórmula:

$$H=Hn \times 65+He \times 78$$

Donde *Hn* son las horas trabajadas en horario normal y *He* son las horas trabajadas fuera del horario normal.

El tiempo empleado en la realización de este Proyecto Fin de Carrera ha sido de 7 meses. Tomando de estos 7 meses 154 días hábiles de los cuales se trabajó 4 horas diarias en horario normal y 2 horas diarias fuera del horario normal. Se tiene que las horas trabajadas en horario normal *(Hn)* fueron:

$$154 \text{ días} \times 4 \text{ horas/día} = 616 \text{ horas}$$

Y las horas trabajadas fuera del horario normal *(He)* fueron:

$$154 \text{ días} \times 2 \text{ horas/día} = 308 \text{ horas}$$

Al aplicar la fórmula para el cálculo de los honorarios se obtiene el siguiente resultado:

$$616 \times 65 + 308 \times 78 = 64064 \text{ €}$$

A esta cantidad hay que aplicarle un coeficiente corrector que depende del número de horas totales trabajadas. Este coeficiente lo establece el COITT y se muestra en la tabla 11-1.

Horas	Coeficiente
De 1 a 36	1
De 36 a 72	0,90
De 72 a 108	0,80
De 108 a 144	0,70
De 144 a 180	0,65
De 180 a 360	0,60
De 360 a 510	0,55
De 510 a 720	0,50
De 720 a 1080	0,45
De 1080 en adelante	0,40

Tabla 11-1 : Coeficientes correctores

Dado que la cantidad de horas empleadas para realizar este proyecto asciende a 924 horas, el coeficiente que se debe aplicar es de 0.45. Con todo esto, el coste total por mano de obra asciende a:

Concepto	Horas trabajadas	Coste (€) sin coef.	Coeficiente	Total (€)
Mano de obra	924	64064	0,45	28828,8

3 Amortización del material inventariable

Para la realización de este proyecto ha sido necesario un equipo informático con las características relacionadas a continuación:

- Microprocesador: Pentium IV 2,53 GHz
- Memoria RAM 512 MB
- Disco Duro 40 GB
- Monitor LG TFT 17''
- Tarjeta gráfica NVIDIA GForce4 MX 440

También ha sido necesario el siguiente material de laboratorio que se describe a continuación:

- Multímetro
- Fuente de alimentación
- Taladro
- Brocas
- Soldador

Se ha establecido un periodo de amortización para todos los elementos de 5 años, durante aproximadamente 154 días al año y trabajando 5 horas diarias. Esto supone un total de 3850 horas de uso.

Para calcular el gasto a amortizar del material, partimos del coste de dicho material calculando la cantidad amortizada en una hora de trabajo (dividiendo el coste total entre

3850). Por último, multiplicamos el coste por hora de trabajo por el número de horas trabajadas. De esta manera obtendremos los siguientes gastos:

Elemento	Coste (€)/unidad	Coste(€)/hora	Tiempo de uso (horas)	Amortización (€)
Ordenador	1000	0,26	790	205,19
Multímetro	115	0,02	5	0,10
Fuente de alimentación	235	0,06	20	1,22
Demás elementos de laboratorio	70	0,01	6	0,06
Total amortización material inventariable				206,58

4 Amortización del software utilizado

Para los programas informáticos se ha establecido un período de amortización de 3 años, durante aproximadamente 154 días al año y trabajando 5 horas diarias. Se ha elegido un período de amortización menor al de los elementos hardware, ya que se considera que el software se devalúa antes. En total, las horas de uso serían 2310.

A continuación se detallan los programas utilizados en la elaboración de este proyecto, junto a los costes de cada uno. No se incluyen aquellos programas que son gratuitos o que sólo han sido utilizados en periodos de evaluación sin coste.

Software	Coste(€)/unidad	Coste(€)/hora	Tiempo de uso (horas)	Amortización
S.O. Windows XP	152,19	0,07	790	52,05
Borland Jbuilder X	500	0,22	365	79
Microsoft Visual Studio	400	0,17	270	46,75
Microsoft Office XP	328,74	0,14	150	21,35
Adobe Acrobat 6	428,04	0,19	2	0,38
Dreamweaver 4	439	0,19	5	0,95
Protel 1,5	20	0,01	15	0,13
Total amortización del software utilizado				201,03

5 Varios

Se deben añadir los costes correspondientes a materiales diversos, no incluidos en los apartados anteriores y que corresponde a folios, tinta de impresión, encuadernación, CD-R incluidos en el proyecto, transporte, etc. El gasto de este material es de:

180 €

6 Presupuesto total

Para obtener el presupuesto total se debe sumar todas las cantidades parciales obtenidas en los apartados anteriores, con lo que se obtiene:

Concepto	Total (€)
Material empleado	167,11
Mano de obra	28828,8
Amortización material	206,58
Amortización software	201,03
Varios	180
Total	29583,5

A este importe total se le debe aplicar el IGIC (Impuesto General Indirecto Canario) que para esta actividad grava un 4,5%, con lo que el presupuesto total de este Proyecto Fin de Carrera asciende a:

Presupuesto total	**30914,75 €**

Treinta mil novecientos catorce euros con setenta y cinco céntimos

El alumno autor del proyecto:

Fdo. Ruperto J. Hernández Saiz

A. Herramientas empleadas

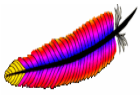
En este apéndice se nombran las herramientas software y hardware empleadas en la realización de este proyecto fin de carrera.

A.1 Elementos software

A.1.1 Servidor

	**Sistema Operativo Microsoft Windows XP** Sistema Operativo multitarea que dispone de un entorno de ventanas, lo que permite al usuario trabajar de forma cómoda en un entorno amigable.
	**Borland JBuilder X** Herramienta de desarrollo de aplicaciones Java. Cuenta con un entorno de ventanas que simplifica la realización de las tareas que se producen de comúnmente en la programación de aplicaciones.
	**Microsoft Paint** Aplicación incluida en Windows, utilizada para el retoque de las imágenes incluidas en la aplicación.
	**DreamWeaver 4** Aplicación utilizada para elaborar páginas Web. Se ha utilizado para confeccionar la ayuda.

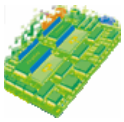
A.1.2 *Cliente*

	**Sistema Operativo Microsoft Windows XP** Sistema Operativo multitarea que dispone de un entorno de ventanas, lo que permite al usuario trabajar de forma cómoda en un entorno amigable.
	**Borland JBuilder X** Herramienta de desarrollo de aplicaciones Java. Cuenta con un entorno de ventanas que simplifica la realización de las tareas que se producen de comúnmente en la programación de aplicaciones.
	**Microsoft Paint** Aplicación incluida en Windows, utilizada para el retoque de las imágenes incluidas en la aplicación.
	**DreamWeaver 4** Aplicación utilizada para elaborar páginas Web. Se ha utilizado para confeccionar la ayuda.
	**Apache HTTP Web Server 2.0.51** Servidor Web altamente configurable y ampliamente utilizado.
	**Microsoft Internet Explorer** Navegador Web. Empleado para probar el funcionamiento del applet

A.1.3 *Gestor USB*

	**Sistema Operativo Microsoft Windows XP** Sistema Operativo multitarea que dispone de un entorno de ventanas, lo que permite al usuario trabajar de forma cómoda en un entorno amigable.
	**Microsoft Visual C++ 6.0** Herramienta de desarrollo de aplicaciones C/C++. Cuenta con un entorno de ventanas que simplifica la realización de las tareas que se producen de comúnmente en la programación de aplicaciones.

A.1.4 Módulo Hardware

	**Sistema Operativo Microsoft Windows XP** Sistema Operativo multitarea que dispone de un entorno de ventanas, lo que permite al usuario trabajar de forma cómoda en un entorno amigable.
	**MPLAB IDE v6.61** Herramienta de desarrollo de código ensamblador para µC PIC. Ofrece una interfaz gráfica sencilla. El fabricante, Microchip, lo distribuye de forma gratuita.
	**ICProg v1.05C** Programa que se distribuye de forma gratuita, utilizado para enviar al programador de microcontroladores (elemento hardware), la información a grabar en la memoria del circuito integrado.
	**Protel 1.5** Aplicación empleada para el diseño de del circuito impreso del módulo Hardware

A.1.5 Realización de la memoria

	**Sistema Operativo Microsoft Windows XP** Sistema Operativo multitarea que dispone de un entorno de ventanas, lo que permite al usuario trabajar de forma cómoda en un entorno amigable.
	**Microsoft Word XP** Procesador de textos de Microsoft Office, versión XP, utilizado para la realización de la presente memoria.
	**Microsoft Paint** Aplicación incluida en Windows, utilizada para el retoque de las imágenes incluidas en la memoria.
	**Microsoft Power Point XP** Programa utilizado para realizar la presentación en que se muestran los conceptos generales de este proyecto; esta presentación será mostrada durante la evaluación del PFC.

A.2 Elementos Hardware

	**Ordenador Personal** PC con procesador Intel Pentium IV 2.53 GHz, 40 GB de disco duro, 512 MB de memoria RAM
	**Programador de Microcontroladores** Elemento hardware utilizado para grabar en la memoria de los microcontroladores (PICs en este caso), el código generado.
	**Borrador de EPROMs** Aparato que cuenta con un tubo de luz ultravioleta, necesario para borrar las memorias de tipo EPROM (implementada en el PIC16C745)

	**Impresora** HP LaserJet 5100

**********GESTION DE RED**********

red.java

```
import java.io.*;
import java.util.*;

//clase que define la estructura de los datos a enviar
 class PDU {
  protected Vector datos;

  public PDU(){
    datos = new Vector();
  }

  public void setTipo(int tipo, int subTipo){
    datos.removeAllElements();
    datos.add((new String()).valueOf(tipo));
    datos.add((new String()).valueOf(subTipo));
  }

  public void borrar(){
    datos.removeAllElements();
  }

  public void addDato(String dato){
    datos.add(dato);
  }

  public void addDato(int dato){
    datos.add((new Integer(dato)).toString()); //
  }

  public void addDato(double dato){
    datos.add((new Double(dato)).toString());
  }
```

```
 public short getTipo(){
   try{
     return (new Short( (String) datos.elementAt(0))).shortValue();
   }
   catch(NullPointerException e){return 0xFF;}
   catch(ArrayIndexOutOfBoundsException e){return 0xFF;}
 }

 public short getSubtipo(){
   try{
     return (new Short( (String) datos.elementAt(1))).shortValue();
   }
   catch(NullPointerException e){return 0xFF;}
   catch(ArrayIndexOutOfBoundsException e){return 0xFF;}
 }

 public int sizeDatos(){
  return (datos.size()-2); //no tipo ni subtipo
 }

 public String getCampo(int campoDATO){//sin tener en cuenta tipo ni subtipo
  try{
   return (String) datos.elementAt(campoDATO+2);
  }
  catch(ArrayIndexOutOfBoundsException e){return null;}
 }

public int getIntcampo(int campoDATO) throws NumberFormatException{
 try{
  return (int) (new Integer((String)(datos.elementAt(campoDATO+2)))).intValue();
 }
 catch(ArrayIndexOutOfBoundsException e){return 0xFFFF;}
 catch(NumberFormatException e){return 0xFFFF;}
}
```

```
 public double getDoubleCampo(int campoDATO) throws
NumberFormatException{
   try{
       return (double)(new Double((String)(datos.
                    elementAt(campoDATO+2)))).doubleValue();
     }
     catch(ArrayIndexOutOfBoundsException e){return 0xFFFF;}
     catch(NumberFormatException e){return 0xFFFF;}
 }
}

class PDU_TCP extends PDU{

   public PDU_TCP() {
      super();
    }

   public PDU_TCP(PDU_TCP pdu){
     super();

     this.setTipo(pdu.getTipo(),pdu.getSubtipo());
     for(int i=0;i < pdu.sizeDatos(); i++){
       this.addDato(pdu.getCampo(i));
     }
   }

  public synchronized Vector getInfo(){ //para serializar
     return datos;
   }

   public synchronized void setInfo(Vector datos){ //para deserializar
        this.datos = datos;
   }
```

```
}

class PDU_UDP extends PDU{
  public int size;

  public PDU_UDP() {
    super();
    size = 0;
  }

  public PDU_UDP(PDU_UDP pdu){
    super();

    this.setTipo(pdu.getTipo(),pdu.getSubtipo());
    for(int i=0;i < pdu.sizeDatos(); i++){
      this.addDato(pdu.getCampo(i));
    }
  }

  public synchronized byte[] getBytes() { //obtenemos la secuencia de bytes
    ByteArrayInputStream Bin;
    ByteArrayOutputStream Bout;
    ObjectOutputStream out;

    try {
      Bout = new ByteArrayOutputStream();
      out = new ObjectOutputStream(Bout);

      out.writeObject(datos);
      size = Bout.size();

      byte[] array = new byte[size + 2];
      array[0] = (byte) ( (size & 0x3F80) / 0x80);
      array[1] = (byte) (size & 0x007F);
```

```
      Bin = new ByteArrayInputStream(Bout.toByteArray());
      Bin.read(array, 2, size);
      Bout.reset();
      out.reset();
      return array;
    }
    catch (IOException e) {
      return null;
    }
  }

  public synchronized void setBytes(byte[] array) throws IOException { //a partir de
los bytes,
    try { //obtendremos el Vector
      ObjectInputStream in = new ObjectInputStream(new ByteArrayInputStream(
          array));
      datos = (Vector) in.readObject();
    }
    catch (IOException e) {
      throw e;
    }
    catch (ClassNotFoundException e) {
      throw new IOException("Error en la recepcion del vector");
    }
  }
}
```

paqueteTCP.java

```
import java.io.*;
import java.net.*;
import java.util.*;

class PaqueteTCPrecepcion extends Thread{
  private ObjectInputStream in;
```

```
 private Vector listeners;
 private PDU_TCP pdu;
 private short id;

 public PaqueteTCPrecepcion(){
  pdu = new PDU_TCP();
 }

 public void setParam(short id, Socket cliente)
    throws IOException{
  if (cliente != null)
    in = new ObjectInputStream(cliente.getInputStream());
  this.id = id;
 }

 private synchronized void close() {
  try{
    in.close();
  }
  catch(IOException e){}
 }

 private short recibir() throws SocketException{
  try{
    if (pdu == null)
      pdu = new PDU_TCP();
    pdu.setInfo((Vector)in.readObject());
    return 0;
  }
  catch(SocketTimeoutException e){
    pdu = null;
    return 1;
  }
  catch(SocketException e){
```

```
      avisaRecepcion((short) 3, id, null);
      throw e;
    }
    catch(ClassNotFoundException e){
      pdu = null;
      return 2;
    }
    catch(IOException e){
      pdu = null;
      return 2;
    }
  }

  public void run(){
    try{
      avisaRecepcion((short)7,id,null);

      while (true){
        avisaRecepcion(recibir(),id,pdu);
        }
      }
    catch(SocketException e){close();}
  }

//*****************************Control del evento*******************************
    public void addTCPListener(PaqueteTCPListener l) {//recepcion
      if (listeners == null)
        listeners = new Vector();

      listeners.addElement(l);
    }

    public void removeTCPListener(PaqueteTCPListener l) {
      if (listeners == null)
```

```
      listeners = new Vector();

    listeners.removeElement(l);
  }

  protected synchronized void avisaRecepcion(short suceso, short id, PDU_TCP
pdu) {
    EventoPaqueteTCP event;

    if (listeners != null && !listeners.isEmpty()) {
      if (pdu != null)
      {
        event = new EventoPaqueteTCP(this, suceso, id,
            new PDU_TCP(pdu));
      }
      else
      {
        event = new EventoPaqueteTCP(this, suceso, id, null);
      }
      Vector targets;
      synchronized (this) {
        targets = (Vector) listeners.clone();
      }

      Enumeration enum = targets.elements();
      while (enum.hasMoreElements()) {
        PaqueteTCPListener l = (PaqueteTCPListener) enum.nextElement();
        l.recibido(event);
      }
    }
  }
}
 class PaqueteTCPtransmision{
  private ObjectOutputStream out;
```

```
  public PaqueteTCPtransmision(Socket cliente) throws IOException {
    out = new ObjectOutputStream(cliente.getOutputStream());
   }

   public synchronized void close() {
    try{
      out.close();
    }
    catch(IOException e){}
   }

   public synchronized boolean enviar(PDU_TCP pdu) {
    try{
      out.flush();
      out.writeObject(new Vector(pdu.getInfo()));
      return true;
       }
    catch(NullPointerException e) {return false;}
    catch(SocketTimeoutException e){return false;}
    catch(IOException e){return false;}
    }
  }
```

paqueteUDP.java

```
import java.io.*;
import java.net.*;
import java.util.*;

class PaqueteUDPtransmision {
  private InetAddress addr;
  private DatagramSocket socket;
  private int puerto;
```

```
public PaqueteUDPtransmision(int puerto, DatagramSocket socket) {
  this.puerto = puerto;
  this.addr = null;
  this.socket=socket;
}

public short setAddress(StringBuffer host) {
  try {
    this.addr = InetAddress.getByName(host.toString());
    return 0;
  }
  catch (UnknownHostException e) { //ha habido error
    this.addr = null;
    return 1;
  }

}

public short setAddress(InetAddress host) {
    this.addr = host;
    return 0;
  }

public short setAddress(String host) {
  try {
    this.addr = InetAddress.getByName(host);
    return 0;
  }
  catch (UnknownHostException e) { //ha habido error
    this.addr = null;
    return 1;
  }
}
```

```
  public short enviar(PDU_UDP pdu) {
    if (addr!=null) {
      try {
        DatagramPacket pack = new DatagramPacket(pdu.getBytes(),(pdu.size+2),
                                        addr, puerto);
         socket.send(pack);
        return 0;
      }
      catch (IOException e) {
        return 2;}
      catch (NullPointerException e){
        return 1;                    //faltan datos
      }
    }
    else {
      return 1;                      //faltan datos
    }
  }
}

class PaqueteUDPrecepcion extends Thread{
  private Vector listeners;
  private DatagramPacket packet;
  private PDU_UDP pdu;
  DatagramSocket socket;
  private final int MAX_POS = 500;
  private byte[] longitud;

  public PaqueteUDPrecepcion(DatagramSocket socket) throws IOException{
    packet = new DatagramPacket(new byte[MAX_POS],MAX_POS);
    this.socket = socket;
    pdu = new PDU_UDP();
  }
```

```
 private short recibir() throws SocketException{
  try{
   socket.receive(packet);
   return 0;
  }
  catch(SocketTimeoutException e){return 1;}
  catch(SocketException e){throw e;}
  catch(IOException e){
   return 2;}
 }


 public void run(){
  short suceso;
  try{
   while (true){
    suceso = recibir();
    if (suceso == 0){
     pdu.setBytes(conformar(packet.getData()));
     avisaRecepcion(suceso,packet.getAddress(),pdu);
    }
    else{
     avisaRecepcion(suceso,null,null);
    }
   }
  }
  catch(SocketException e){/*close();*/}            //salimos y cerramos el pipe
  catch(IOException e){//System.out.println(e);
   avisaRecepcion((short)2,packet.getAddress(),null);}

 }

 private byte[] conformar(byte[] dato){
  ByteArrayOutputStream stream = new ByteArrayOutputStream();
```

```
    stream.write(dato,2,(dato[0]*0x80)+dato[1]);
    return stream.toByteArray();
  }
//*****************************Control del evento*******************************

  /** Registra un "listener" para avisar cuando llega un paquete ó aparece un
   * timeout */
    synchronized public void addUDPListener(PaqueteUDPListener l) {//recepcion
      if (listeners == null)
        listeners = new Vector();

      listeners.addElement(l);
    }

    /** Elimina un "listener" de la lista*/
    synchronized public void removeUDPListener(PaqueteUDPListener l) {
      if (listeners == null)
        listeners = new Vector();

      listeners.removeElement(l);
    }

    /** Indica el evento a todos los "listeners" registrados */
    protected void avisaRecepcion(short suceso, InetAddress addr, PDU_UDP pdu)
{
      // si no hay listeners, no hacemos nada
      EventoPaqueteUDP event;

      if (listeners != null && !listeners.isEmpty()) {
        // creamos el objeto (evento) a enviar
        if (pdu != null)
        {
          event = new EventoPaqueteUDP(this, suceso, addr, new PDU_UDP(pdu));
```

```
        }
        else
        {
            event = new EventoPaqueteUDP(this, suceso, addr, null);
        }

        Vector targets;                    // hacemos una copia de la lista,
        synchronized (this) {              //por si acaso alquien
          targets = (Vector) listeners.clone(); //añade ó elimina otro evento
        }

        Enumeration enum = targets.elements();  //recorremos la lista, llamando
        while (enum.hasMoreElements()) {        //al método recibido de cada uno
          PaqueteUDPListener l = (PaqueteUDPListener) enum.nextElement();
          l.recibido(event);               //de los "listener" registrados
        }
      }
    }
}

import java.util.EventListener;
import java.util.EventObject;
import java.net.InetAddress;

/*************************EVENTOS DE RED*********************************/

class EventoPaqueteTCP extends EventObject { //clase del evento
  short suceso;

  PDU_TCP pdu;
  short id; //distingue entre conexiones (gestion/actuador)
```

```
  public EventoPaqueteTCP(Object source,short suceso, short id, PDU_TCP pdu)
{
    super(source);
    this.suceso = suceso;
    this.pdu = pdu;
    this.id = id;
    }
}

class EventoPaqueteUDP extends EventObject { //clase del evento
    short suceso;

  InetAddress addr;
  PDU_UDP pdu;

  public EventoPaqueteUDP(Object source, short suceso,InetAddress addr,
PDU_UDP pdu) {
    super(source);
    this.suceso = suceso;
    this.addr = addr;
    this.pdu = pdu;
    }
}

interface PaqueteTCPListener extends EventListener { //listener del TCP
//llamado cuando llegue un paquete TCP, o se produzca un Timeout
    public void recibido(EventoPaqueteTCP evento);
  }

interface PaqueteUDPListener extends EventListener { //listener del UDP
//llamado cuando llegue un paquete UDP, o se produzca un Timeout
    public void recibido(EventoPaqueteUDP evento);
  }
```

```
/*************************EVENTOS USUARIOS*********************************/
class EventoUsuario extends EventObject { //clase del evento
   short suceso;  // 0:Nueva entrada   1:Entrada eliminada   2:Entrada Modificada
   String login;

   public EventoUsuario(Object source,short suceso, String login) {
     super(source);
     this.suceso = suceso;

     this.login = login;
     }
 }

interface UsuarioListener extends EventListener { //listener del TCP
//llamado cuando haya alguna modificacion en la tabla de usuarios
     public void recibido(EventoUsuario evento);
   }

/*************************EVENTOS CONEXIONES*********************************/
class EventoConexion extends EventObject { //clase del evento
       short suceso;  // 0:Nueva entrada   1:Entrada eliminada   2:Entrada
Modificada
       entradaConexion entrada;

       public EventoConexion(Object source,short suceso,
                        entradaConexion entrada) {
         super(source);
         this.suceso = suceso;
         this.entrada = entrada;
       }
     }

interface ConexionListener extends EventListener { //listener del TCP
//llamado cuando haya alguna modificacion en la tabla de conexiones
```

```
    public void recibido(EventoConexion evento);
  }
```

**********AYUDA**********

PanelAyuda.java

```
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.html.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.beans.*;
import java.net.URL;

public class PanelAyuda extends JFrame implements HyperlinkListener{
  private GridBagConstraints constraints;
  private final boolean applet;
  JEditorPane derecha;
  URL CodeBase;

  public PanelAyuda(boolean esApplet, URL codeBase) {
    super("USBMotor : Ayuda");
    this.applet = esApplet;
    this.CodeBase = codeBase;

    constraints = new GridBagConstraints();

    JButton ayuda = new JButton("Ayuda de la aplicacion");
    JButton acerca = new JButton("Acerca de...");

    Container container = this.getContentPane();
    container.setLayout(new GridBagLayout());
```

```
addGB(container, ayuda, 0, 0);
addGB(container, new JLabel(" "), 0, 1);
constraints.ipadx = 45;
addGB(container, acerca, 0, 2);

setResizable(false); //no se puede modificar su tamaño
setSize(200, 125);
setVisible(true);

/**listeners*/
ayuda.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ev) {
    try {
      JEditorPane izquierda;
      JSplitPane divisor;

      if(applet){
        URL url = new URL(CodeBase,"Ayuda/menu.htm");
        izquierda = new JEditorPane(url);
        url = new URL(CodeBase,"Ayuda/inicio.htm");
        derecha = new JEditorPane(url);
      }
      else{
        File fichero = new File("Ayuda/menu.htm");
        izquierda = new JEditorPane(fichero.toURL());

        fichero = new File("Ayuda/inicio.htm");
        derecha = new JEditorPane(fichero.toURL());
      }
      izquierda.setEditable(false);
      derecha.setEditable(false);
      izquierda.addHyperlinkListener(listener());
      derecha.setVisible(true);
```

```
        divisor = new JSplitPane();
        divisor.addPropertyChangeListener(new PropertyChangeListener(){
          public void propertyChange(PropertyChangeEvent ev){
            ((JSplitPane)ev.getSource()).setDividerLocation(195);
          }
        });

        divisor.setLeftComponent(izquierda);

        JScrollPane scroll = new JScrollPane(derecha);

scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS)
;

scroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NE
VER);
        divisor.setRightComponent(scroll);
        divisor.setDividerLocation(195);


        JFrame help = new JFrame("USBMotor : Ayuda");
        help.getContentPane().add(divisor);
        help.setSize(900, 600);
        help.setResizable(false);
        help.setVisible(true);

        foo().setVisible(false);
      }
      catch (IOException e) {
        JOptionPane.showMessageDialog(foo(), "No ha sido posible localizar" +
                                " el archivo de ayuda");
      }
    }
  });
```

```
    acerca.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
        foo().setVisible(false);
        Acerca acerca = new Acerca();
      }
    });

  }

  public void hyperlinkUpdate(HyperlinkEvent e) {
    if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
      JEditorPane pane = (JEditorPane) e.getSource();
      if (e instanceof HTMLFrameHyperlinkEvent) {
        HTMLFrameHyperlinkEvent  evt = (HTMLFrameHyperlinkEvent)e;
        HTMLDocument doc = (HTMLDocument)pane.getDocument();
        doc.processHTMLFrameHyperlinkEvent(evt);
      } else {
        try {
          derecha.setPage(e.getURL());
        } catch (Throwable t) {
          t.printStackTrace();
        }
      }
    }
  }

  private JFrame foo() {
    return this;
  }

  private HyperlinkListener listener() {
  return this;
}
```

```
  private void addGB(Container panel, Component component, int x, int y) {
    constraints.gridx = x;
    constraints.gridy = y;
    panel.add(component, constraints);
  }
}

class Acerca extends JFrame{
  private GridBagConstraints constraints;

  public Acerca(){
    super("USBMotor : Acerca de...");
    constraints = new GridBagConstraints();

    Container container = this.getContentPane();
    container.setLayout(new GridBagLayout());

    JButton aceptar = new JButton("Aceptar");

    addGB(container,new JLabel(crearIcono("iconos/USBMotorpeque.png"),
                          JLabel.CENTER),0,0);
    constraints.anchor = GridBagConstraints.EAST;
    addGB(container,aceptar,2,4);
    constraints.anchor = GridBagConstraints.WEST;
    addGB(container,new JLabel(" "),0,1);
    addGB(container,new JLabel("Proyecto de Fin de Carrera EUITT -
ULPGC"),0,2);
    constraints.anchor = GridBagConstraints.CENTER;
    addGB(container,new JLabel(" "),0,3);
    addGB(container,new JLabel("Autor : Ruperto J. Hernández Saiz"),0,4);
    addGB(container,new JLabel("Tutor : Domingo Marrero Marrero"),0,5);

    /**listener*/
    aceptar.addActionListener(new ActionListener() {
```

```
      public void actionPerformed(ActionEvent ev) {
        foo().setVisible(false);
      }
    });
    setResizable(false);
    setSize(310,160);
    setVisible(true);
  }

  private ImageIcon crearIcono(String path) {
    return new ImageIcon(path);
  }

  private JFrame foo(){
    return this;
  }

  private void addGB(Container panel,Component component,int x, int y){
  constraints.gridx = x;
  constraints.gridy = y;
  panel.add(component,constraints);
  }
}
```

**********INTERFAZ GRÁFICA**********

Dial.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Dial extends JComponent implements
MouseListener,MouseMotionListener{
  final int valorMinimo;
  int valor, valorReal,radio,valorMaximo;
```

```
  double anguloMaximo;

  public Dial(int n_pasos, int valor_inicial, boolean permite_movimiento){
    this.valorMinimo = 0;
    this.valorMaximo = n_pasos;
    this.valor = valor_inicial;
    this.valorReal = valor_inicial;
    this.anguloMaximo = 0; //inicialmente desactivado

    setForeground(Color.lightGray);

    activaListeners(permite_movimiento);
  }
//interfaz de MouseListener
  public void mousePressed(MouseEvent e) {}
  public void mouseClicked(MouseEvent e) {}
  public void mouseEntered (MouseEvent e){}
  public void mouseExited (MouseEvent e){}
  public void mouseReleased (MouseEvent e){}

//interfaz de MouseMotionListener
  public void mouseDragged(MouseEvent e) {
    spin(e);
    firePropertyChange("valor",this.valor,valor); //indicamos movimiento
  }
  public void mouseMoved (MouseEvent e){}

  public void activaListeners(boolean activar){
    if (activar == true){
      addMouseListener(this);
      addMouseMotionListener(this);
    }
    else {
      removeMouseListener(this);
```

```
    removeMouseMotionListener(this);
  }
 }

 protected void spin(MouseEvent e){
  int y = e.getY();
  int x = e.getX();
  int temp;

  double th = Math.atan((1.0 * y - radio) / (x - radio));
  int valor = ((int)(th / (2 * Math.PI) * (valorMaximo - valorMinimo)));

  if (x < radio){
   temp = valor + (valorMaximo / 2);
   if (temp == valorMaximo)
    setValor(valorMinimo);
   else
    setValor(temp);
  }
  else{
   if (y < radio) {
    temp = valor + valorMaximo;
    if (temp == valorMaximo)
     setValor(valorMinimo);
    else
     setValor(temp);
   }
   else{
    if (valor == valorMaximo)
    setValor(valorMinimo);
   else
    setValor(valor);
   }
  }
```

```
}

public void paintComponent(Graphics g){
  Graphics2D g2 = (Graphics2D)g;
  int tick = 10;
  radio = getSize().width / 2 - tick;

  g2.setPaint(getForeground().darker());
  g2.drawLine(radio * 2 + tick / 2, radio, radio * 2 + tick, radio);
  g2.setStroke(new BasicStroke(2));
  dibujaCirculo3D(g2,2,2,radio,true);
  int knobRadio = radio / 7;

  double th = (valorReal * (2 * Math.PI) / (valorMaximo - valorMinimo));
  int x = (int)(Math.cos(th) * (radio - knobRadio * 3));
  int y = (int)(Math.sin(th) * (radio - knobRadio * 3));
  g2.setStroke(new BasicStroke(1));

  dibujaCirculo2D(g2, x + radio - knobRadio + 2, y + radio - knobRadio + 2,
                  knobRadio);

  th = (valor * (2 * Math.PI) / (valorMaximo - valorMinimo));
  x = (int)(Math.cos(th) * (radio - knobRadio * 3));
  y = (int)(Math.sin(th) * (radio - knobRadio * 3));
  g2.setStroke(new BasicStroke(1));

  dibujaCirculo3D(g2, x + radio - knobRadio +2, y + radio - knobRadio +2,
                  knobRadio , false);
}

private void dibujaCirculo3D(Graphics g, int x, int y,
                             int radio, boolean elevado){
  Color foreground = getForeground();
  Color light = foreground.brighter();
```

```
    Color dark = foreground.darker();
    g.setColor(foreground);
    g.fillOval(x,y,radio * 2, radio * 2);
    g.setColor(elevado ? light : dark);
    g.drawArc(x,y,radio * 2,radio *2,45,180);
    g.setColor(elevado ? dark : light);
    g.drawArc(x,y,radio * 2,radio * 2,225,180);
  }

  private void dibujaCirculo2D(Graphics g, int x, int y,
                               int radio){
    Color foreground = getForeground();
    Color light = foreground.brighter();
    g.setColor(foreground);
    g.fillOval(x,y,2 * radio, 2 * radio);
    g.setColor(foreground.brighter());
    g.drawOval(x,y,2 * radio, 2 * radio);
  }

  public Dimension getPreferredSize(){
    return new Dimension(100,100);
  }

  public void setValor(int paso){
    int temp = paso;

    if (anguloMaximo > 0){ //está activado el ángulo máximo
      if ((anguloMaximo/(360.0 / valorMaximo)) < paso) //se nos pasa del máximo
        temp = (int)(anguloMaximo/(360.0 / valorMaximo)); //lo colocamos en el
ángulo maximo
    }
    else{
      while (temp >= valorMaximo) { //pues lo que queremos es orientarlo, no dar
vueltas
```

```
      temp = temp - valorMaximo;
    }
  }

  this.valor = temp;
  repaint();
  avisaDial();
}

public void setValor(double angulo){
  this.valor = pasoCorrecto(angulo);
  repaint();
  avisaDial();
}

public void setValorReal(int n_pasos){
  this.valorReal = n_pasos;
  repaint();
}

public void setAnguloMáximo(double angulo){
  anguloMaximo = angulo;
  if (valor > (angulo / (360.0 / valorMaximo)))
    setValor(valorMaximo);

  if (valorReal > (angulo / (360.0 / valorMaximo)))
    setValorReal((int)(angulo / (360.0 / valorMaximo)));
}

public int getValorReal(){
  return valorReal;
}

public int getValor() {
```

```
  return valor;
 }

 public void setNpasos(int n_pasos){
  this.valorMaximo = n_pasos;
 }

 public int getNpasos(){
  return valorMaximo;
 }

 public boolean isAnguloActual(double angulo){
  if (angulo == (valor * (360.0/valorMaximo)))
    return true;
  else
    return false;
 }

 private int pasoCorrecto(double angulo){
  double temp = (double)(angulo / (360.0/valorMaximo));

  if (Math.floor(temp) == temp)
    return (int)Math.floor(temp);

  if (Math.floor(temp) + 0.5 > temp)
    return (int)(Math.floor(temp));
  else
    return (int)(Math.floor(temp)+1);
 }

 public void addDialListener(DialListener listener){
  listenerList.add(DialListener.class, listener);
 }
```

```
  public void removeDialListener(DialListener listener) {
    listenerList.remove(DialListener.class, listener);
  }

  void avisaDial(){
    double valor = (double)(this.valor) *360/(valorMaximo);
    Object[] listeners = listenerList.getListenerList();
    for(int i = 0;i < listeners.length; i += 2 )
      if (listeners[i] == DialListener.class)
        ((DialListener)listeners[i+1]).dialAjustado(new DialEvent(this,valor));
  }
}
```

DialEvent.java

```
public class DialEvent extends java.util.EventObject{
    private double valor;

    DialEvent (Dial source, double valor){
      super(source);
      this.valor = valor;
    }

    public Double getValor(){
      return new Double(valor);
      //Double numero = new Double(valor);
//      String temp = numero.toString();

  //    return temp;
      /*if (temp.length() >= temp.indexOf('.') + 3) //2 o mas decimales
        return temp.substring(0, temp.indexOf('.') + 3); //cogemos 2 decimales
      else
        return temp.substring(0, temp.indexOf('.') + 2); //cogemos 1 decimal*/
```

```
    }
}

 interface DialListener extends java.util.EventListener {
   void dialAjustado(DialEvent e);
 }
```

JSpinnerDial.java

```
import javax.swing.*;
import java.awt.*;

public class JSpinnerDial extends JSpinner{
  double angulo_paso;

  public JSpinnerDial(int n_pasos, int pos_inicial){
    super();
    angulo_paso = 360.0/n_pasos;
    SpinnerNumberModel model = new SpinnerNumberModel(
      pos_inicial * angulo_paso, 0.0, 360.0, angulo_paso);
    this.setModel(model);
  }

  public JSpinnerDial(int n_pasos, int pos_inicial, double anguloMaximo){
    super();
    SpinnerNumberModel model;
    angulo_paso = 360.0/n_pasos;

    if ((pos_inicial*angulo_paso) > anguloMaximo)
      model = new SpinnerNumberModel(
    anguloMaximo, 0.0, anguloMaximo, angulo_paso);
    else
      model = new SpinnerNumberModel(
    pos_inicial * angulo_paso, 0.0, anguloMaximo, angulo_paso);
```

```
    this.setModel(model);
  }

  public JSpinnerDial(double angulo_paso, double angulo_inicial){
    super();
    this.angulo_paso = angulo_paso;
    SpinnerNumberModel model = new SpinnerNumberModel(
      anguloCorrecto(angulo_inicial), 0.0, 360.0, angulo_paso);
    this.setModel(model);
  }

  public Object getNextValue(){
    Double temp = new Double((new
Double(getValue().toString())).doubleValue()+angulo_paso);

    if (temp.doubleValue() >= 360.0)
      temp = new Double(360.0-temp.doubleValue());
    firePropertyChange("angulo_paso",getValue(),temp);
    setValue(temp);
    return temp;
  }

  public Object getPreviousValue(){
    Double valor = new Double(getValue().toString());
    Double temp;
    if (valor.doubleValue() == 0.0)
      temp = new Double(360-angulo_paso);
    else
      temp = new Double(valor.doubleValue()-angulo_paso);

    firePropertyChange("angulo_paso",getValue(),temp);
    setValue(temp);
    return temp;
  }
```

```
  public void setValue(Object o){
    Double temp = new Double(o.toString());
    Double valor = new Double(anguloCorrecto(temp.doubleValue()));

    super.setValue(valor);
  }

  public void setNpasos(int n_pasos){
    double temp = (new Double(this.getValue().toString())).doubleValue() /
angulo_paso;
  //calculamos el paso en el que estabamos, para asi preservar donde estamos

    this.angulo_paso = 360.0/n_pasos;
    SpinnerNumberModel model = new SpinnerNumberModel(
      (new Double(getValue().toString())).doubleValue(),0.0, 360.0, angulo_paso);
    this.setModel(model);

    setValue(new Double(temp * angulo_paso));
  }

  private double anguloCorrecto(double angulo){
    double temp = (double)(angulo / (angulo_paso));

    if (Math.floor(temp) == temp)
      return angulo;

    if (Math.floor(temp) + 0.5 > temp)
      return (Math.floor(temp))*(angulo_paso);
    else
      return (Math.floor(temp)+1)*(angulo_paso);
  }
}
```

PanelMovimiento.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class PanelMovimiento extends JPanel{
  GridBagConstraints constraints;
  boolean permiso_movimiento;
  boolean quieto;
  double anguloPaso1;
  double anguloPaso2;

  public PanelMovimiento(int n_pasos1, int n_pasos2,
                int paso_inicial1, int paso_inicial2,
                int velocidad_inicial1,int velocidad_inicial2,
                double anguloMaximo1,
                double anguloMaximo2,
                short velocidadMaxima1, short velocidadMaxima2,
                boolean permisos_movimiento){

    super();

    setName("panel central");

    this.permiso_movimiento = permisos_movimiento;
    this.quieto = true;
    this.anguloPaso1 = 360.0 / n_pasos1;
    this.anguloPaso2 = 360.0 / n_pasos2;

    constraints = new GridBagConstraints();
    setLayout(new GridBagLayout());
```

```
    JPanel Ptemp = new JPanel();
    Ptemp.setLayout(new GridBagLayout());

    final Dial dial1;
    final Dial dial2;
    final JButton moverButton;
    final JComboBox listaMovimientos;
    final JSpinnerDial angulo_motor1;
    final JSpinnerDial angulo_motor2;
    final JButton reiniciarButton1;
    final JButton reiniciarButton2;

    final JSlider velocidadSlider1;
    final JSlider velocidadSlider2;

    final JLabel velocidadLabel1;
    final JLabel velocidadLabel2;


//inicializamos los objetos
    dial1 = new Dial(n_pasos1,paso_inicial1,permiso_movimiento);
    dial1.setName("dial1");

    dial2 = new Dial(n_pasos2,paso_inicial2,permiso_movimiento);
    dial2.setName("dial2");

    moverButton = new JButton("Mover");
    moverButton.setEnabled(false);
    moverButton.setName("mover");

    listaMovimientos = new JComboBox();
    listaMovimientos.setEnabled(permisos_movimiento);
    listaMovimientos.setEditable(false);
```

```
listaMovimientos.setName("lista");
listaMovimientos.addItem("Ambos Motores");
listaMovimientos.addItem("Motor 1");
listaMovimientos.addItem("Motor 2");

if (anguloMaximo1 > 0)
  angulo_motor1 = new JSpinnerDial(n_pasos1,paso_inicial1,anguloMaximo1);
else
  angulo_motor1 = new JSpinnerDial(n_pasos1,paso_inicial1);

angulo_motor1.setEnabled(permiso_movimiento);
angulo_motor1.setName("angulo_motor1");

if (anguloMaximo2 > 0)
  angulo_motor2 = new JSpinnerDial(n_pasos2,paso_inicial2, anguloMaximo2);
else
  angulo_motor2 = new JSpinnerDial(n_pasos2,paso_inicial2);

angulo_motor2.setEnabled(permiso_movimiento);
angulo_motor2.setName("angulo_motor2");

reiniciarButton1 = new JButton("Cancelar");
reiniciarButton1.setEnabled(false);
reiniciarButton1.setName("cancelar1");
reiniciarButton1.setToolTipText("Coloca el cursor en la posición del " +
                        "último movimiento");
reiniciarButton2 = new JButton("Cancelar");
reiniciarButton2.setName("cancelar2");
reiniciarButton2.setToolTipText("Coloca el cursor en la posición del " +
                      "último movimiento");
reiniciarButton2.setEnabled(false);

if (velocidad_inicial1 < velocidadMaxima1)
  velocidadSlider1 = new JSlider(JSlider.VERTICAL,1,
```

```
                    velocidadMaxima1,velocidad_inicial1);
  else
    velocidadSlider1 = new JSlider(JSlider.VERTICAL,1,
                    velocidadMaxima1,velocidadMaxima1);

  velocidadSlider1.setMajorTickSpacing(50);
  velocidadSlider1.setMinorTickSpacing(25);
  velocidadSlider1.setPaintTicks(true);
  velocidadSlider1.setEnabled(permiso_movimiento);
  velocidadSlider1.setName("velocidad1");

  if (velocidad_inicial2 < velocidadMaxima2)
    velocidadSlider2 = new JSlider(JSlider.VERTICAL,1,
                    velocidadMaxima2,velocidad_inicial2);
  else
    velocidadSlider2 = new JSlider(JSlider.VERTICAL,1,
                    velocidadMaxima2,velocidadMaxima2);

  velocidadSlider2.setMajorTickSpacing(50);
  velocidadSlider2.setMinorTickSpacing(25);
  velocidadSlider2.setPaintTicks(true);
  velocidadSlider2.setEnabled(permiso_movimiento);
  velocidadSlider2.setName("velocidad2");

  velocidadLabel1 =  new JLabel(cuentaRevoluciones(
                    velocidadSlider1.getValue(),n_pasos1));
  velocidadLabel2 =  new JLabel(cuentaRevoluciones(
                    velocidadSlider2.getValue(),n_pasos2));

  if (anguloMaximo1 > 0){
    dial1.setAnguloMáximo(anguloMaximo1);
  }
  if (anguloMaximo2 > 0){
    dial2.setAnguloMáximo(anguloMaximo2);
```

```
    }

/**colocamos los componentes en el panel*/
    Ptemp.setName("panel dial1");
    constraints.gridx = 0;
    constraints.gridy = 0;
    Ptemp.add(new JLabel("Posición Motor 1"),constraints);
    constraints.gridy = GridBagConstraints.RELATIVE;
    Ptemp.add(dial1,constraints);
    constraints.ipadx = 20;
    constraints.ipady = 5;
    constraints.anchor = GridBagConstraints.CENTER;
    constraints.fill = GridBagConstraints.NONE;
    Ptemp.add(angulo_motor1,constraints);
    constraints.ipadx = 0;
    constraints.ipady = 0;
    Ptemp.add(new JLabel(" "),constraints);
    Ptemp.add(reiniciarButton1,constraints);

    addGB(Ptemp,1,0);

    constraints.ipadx = 1;
    constraints.ipady = 1;
    Ptemp = new JPanel();
    Ptemp.setLayout(new GridBagLayout());

    Ptemp.setName("panel dial2");
    constraints.gridy = 0;
    Ptemp.add(new JLabel("Posición Motor 2"),constraints);
    constraints.gridy = GridBagConstraints.RELATIVE;
    Ptemp.add(dial2,constraints);
    constraints.ipadx = 20;
    constraints.ipady = 5;
    constraints.anchor = GridBagConstraints.CENTER;
```

```
constraints.fill = GridBagConstraints.NONE;
Ptemp.add(angulo_motor2,constraints);
constraints.ipadx = 0;
constraints.ipady = 0;
Ptemp.add(new JLabel(" "),constraints);
Ptemp.add(reiniciarButton2,constraints);

addGB(Ptemp,3,0);

constraints.ipadx = 1;
constraints.ipady = 1;
Ptemp = new JPanel();
Ptemp.setLayout(new GridBagLayout());

Ptemp.setName("panel velocidad1");
constraints.gridy = 0;
Ptemp.add(new JLabel("Velocidad Motor 1"),constraints);
constraints.gridy = GridBagConstraints.RELATIVE;
Ptemp.add(velocidadSlider1,constraints);
Ptemp.add(velocidadLabel1,constraints);
addGB(Ptemp,0,0);

Ptemp = new JPanel();
Ptemp.setLayout(new GridBagLayout());

Ptemp.setName("panel velocidad2");
constraints.gridy = 0;
Ptemp.add(new JLabel("Velocidad Motor 2"),constraints);
constraints.gridy = GridBagConstraints.RELATIVE;
Ptemp.add(velocidadSlider2,constraints);
Ptemp.add(velocidadLabel2,constraints);
addGB(Ptemp,4,0);

constraints.fill = GridBagConstraints.NONE;
```

```
    addGB(moverButton,2,1);
    addGB(new JLabel(""),2,2);
    addGB(listaMovimientos,2,3);

/**Listeners:*/

    moverButton.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
            equals("Ambos Motores")){
          avisaMovimiento(dial1.getValor(),
                          dial2.getValor(),
                          velocidadSlider1.getValue(),
                          velocidadSlider2.getValue());
          reiniciarButton1.setEnabled(false);
          reiniciarButton2.setEnabled(false);
        }
        else
        if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
            equals("Motor 1")){
          avisaMovimiento(dial1.getValor(),
                          velocidadSlider1.getValue(),
                          (short) 1);
          reiniciarButton1.setEnabled(false);
        }
        else
          if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
            equals("Motor 2")){
          avisaMovimiento(dial2.getValor(),
                          velocidadSlider2.getValue(),
                          (short) 2);
          reiniciarButton2.setEnabled(false);
        }
```

```
        moverButton.setEnabled(false);
        listaMovimientos.setEnabled(false);
        quieto = false;
       }
      });

  listaMovimientos.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
      if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
            equals("Ambos Motores")){
        if (((dial1.getValor() != dial1.getValorReal()) ||
             (dial2.getValor() != dial2.getValorReal()) && (quieto)))
          moverButton.setEnabled(true);
        else
          moverButton.setEnabled(false);
      }
      else
        if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
            equals("Motor 1")){
          if ((dial1.getValor() != dial1.getValorReal()) && (quieto))
            moverButton.setEnabled(true);
          else
            moverButton.setEnabled(false);
        }
        else
          if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
              equals("Motor 2")){
            if ((dial2.getValor() != dial2.getValorReal()) && (quieto))
              moverButton.setEnabled(true);
            else
              moverButton.setEnabled(false);
          }
    }
```

```
});

reiniciarButton1.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
      dial1.setValor(dial1.getValorReal());

      if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
          equals("Motor 1") || ((listaMovimientos.getItemAt(listaMovimientos.
          getSelectedIndex()).equals("Ambos Motores")) &&
          (dial2.getValorReal() == dial2.getValor()))){
        moverButton.setEnabled(false);
        reiniciarButton1.setEnabled(false);
      }
    }
  });

reiniciarButton2.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
      dial2.setValor(dial2.getValorReal());

      if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
          equals("Motor 2") || ((listaMovimientos.getItemAt(listaMovimientos.
          getSelectedIndex()).equals("Ambos Motores")) &&
          (dial1.getValorReal() == dial1.getValor()))){
        moverButton.setEnabled(false);
        reiniciarButton2.setEnabled(false);
      }

    }
  });

  dial1.addDialListener(new DialListener(){
    public void dialAjustado(DialEvent ev){
     if (!angulo_motor1.getValue().toString().equals(ev.getValor()))
```

```
        angulo_motor1.setValue(ev.getValor());

        if ((permiso_movimiento) && (quieto))
          if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
              equals("Ambos Motores") ||
             listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
              equals("Motor 1")){

             moverButton.setEnabled(permiso_movimiento);
             reiniciarButton1.setEnabled(true);
           }
      }
    });

    dial2.addDialListener(new DialListener(){
      public void dialAjustado(DialEvent ev){
       if (!angulo_motor2.getValue().toString().equals(ev.getValor()))
        angulo_motor2.setValue(ev.getValor());

       if ((permiso_movimiento) && (quieto))
        if ((listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
             equals("Ambos Motores") ||
           listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
             equals("Motor 2"))){
          moverButton.setEnabled(permiso_movimiento);
          reiniciarButton2.setEnabled(true);
         }
      }
    });

    angulo_motor1.addChangeListener(new ChangeListener(){
      public void stateChanged(ChangeEvent ev){
       double temp = (new Double
        (((JSpinnerDial)ev.getSource()).getValue().toString())).doubleValue();
```

```
      if (!dial1.isAnguloActual(temp))
        dial1.setValor(temp);
    }
  });

  angulo_motor2.addChangeListener(new ChangeListener(){
    public void stateChanged(ChangeEvent ev){
      double temp = (new Double
        (((JSpinnerDial)ev.getSource()).getValue().toString())).doubleValue();
      if (!dial2.isAnguloActual(temp))
        dial2.setValor(temp);
    }
  });

  velocidadSlider1.addChangeListener(new ChangeListener(){
    public void  stateChanged(ChangeEvent ev){
      velocidadLabel1.setText(cuentaRevoluciones(
                    velocidadSlider1.getValue(),dial1.getNpasos()));
    }
  });

  velocidadSlider2.addChangeListener(new ChangeListener(){
    public void  stateChanged(ChangeEvent ev){
      velocidadLabel2.setText(cuentaRevoluciones(
                    velocidadSlider2.getValue(),dial2.getNpasos()));
    }
  });
}

 public void setDatos(int paso1,int paso2, int velocidad1, int velocidad2,
                boolean actuador){
  JPanel panel;
  JButton boton;
  JComboBox listaMovimientos;
```

```
short tipo;

tipo = setPosicion((short)1,paso1,actuador);
if ((tipo == 0) && (permiso_movimiento)){
  panel = (JPanel)buscaComponente(this,"panel dial1");
  buscaComponente(panel,"cancelar1").setEnabled(false);
  listaMovimientos = (JComboBox)buscaComponente(this,"lista");
  listaMovimientos.setEnabled(true);
}
else{
  if ((tipo == 1) && (permiso_movimiento)){
    panel = (JPanel)buscaComponente(this,"panel dial1");
    buscaComponente(panel,"cancelar1").setEnabled(true);
    boton = (JButton)buscaComponente(this,"mover");
    boton.setEnabled(true);
    listaMovimientos = (JComboBox)buscaComponente(this,"lista");
    listaMovimientos.setEnabled(true);
  }
}

tipo = setPosicion((short)2,paso2,actuador);
if ((tipo == 0) && (permiso_movimiento)){
  panel = (JPanel)buscaComponente(this,"panel dial2");
  buscaComponente(panel,"cancelar2").setEnabled(false);
  listaMovimientos = (JComboBox)buscaComponente(this,"lista");
  listaMovimientos.setEnabled(true);

}
else{
  if ((tipo == 1) && (permiso_movimiento)){
    panel = (JPanel)buscaComponente(this,"panel dial2");
    buscaComponente(panel,"cancelar2").setEnabled(true);
    boton = (JButton)buscaComponente(this,"mover");
    boton.setEnabled(true);
```

```
        listaMovimientos = (JComboBox)buscaComponente(this,"lista");
        listaMovimientos.setEnabled(true);
      }
    }

    quieto = true;

    setVelocidad((short)1,velocidad1);
    setVelocidad((short)2,velocidad2);
  }

//se ha producido un fallo en el movimiento solicitado
  public void falloMover(){
    JComboBox listaMovimientos;
    JButton boton;
    JPanel panel;

    quieto = true;

    listaMovimientos = (JComboBox)buscaComponente(this,"lista");
    if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
              equals("Motor 1")){
      if (permiso_movimiento){
        panel = (JPanel)buscaComponente(this,"panel dial1");
        buscaComponente(panel,"cancelar1").setEnabled(true);
      }
    }
    else{
      if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
          equals("Motor 2")) {
        if (permiso_movimiento){
          panel = (JPanel)buscaComponente(this,"panel dial2");
          buscaComponente(panel,"cancelar2").setEnabled(true);
        }
```

```
      }

      else{
        if (listaMovimientos.getItemAt(listaMovimientos.getSelectedIndex()).
            equals("Ambos Motores")) {
          if (permiso_movimiento){
            panel = (JPanel)buscaComponente(this,"panel dial1");
            buscaComponente(panel,"cancelar1").setEnabled(true);
            panel = (JPanel)buscaComponente(this,"panel dial2");
            buscaComponente(panel,"cancelar2").setEnabled(true);
          }
        }
      }
    }

    if (permiso_movimiento){
      boton = (JButton) buscaComponente(this, "mover");
      boton.setEnabled(true);
      listaMovimientos.setEnabled(true);
    }
  }

//modifica la posicion a la que apunta el dial (posicion real)
  private short setPosicion(short ID, int n_pasos,boolean actuador){
    /*valor de retorno:  0: El valor actual es el mismo que el real
                         1: El valor actual es distinto del real
                         2: Error*/
    Component componente;

    try{
      if (ID == 1) { //dial1
        componente = buscaComponente(this, "panel dial1"); //buscamos el panel de
dial solicitado
        componente = (Dial)buscaComponente((JPanel)componente, "dial1");
```

```
      }
      else { //dial2
        componente = buscaComponente(this, "panel dial2"); //buscamos el panel de
dial solicitado
        componente = (Dial)buscaComponente((JPanel)componente, "dial2");
      }
      if (!actuador)
        ((Dial)componente).setValor(n_pasos);

      ((Dial)componente).setValorReal(n_pasos);

      if (((Dial)componente).getValor() == n_pasos)
        return 0; //no coinciden
      else
        return 1; //coinciden
    }
    catch (NullPointerException e){
      return 2;
    }
    catch (ClassCastException e){
      return 2;
    }
  }

//modifica la velocidad a la que apunta el dial
  private boolean setVelocidad(short ID, int velocidad){
    Component componente;

    try{
      if (ID == 1) { //dial1
        componente = buscaComponente(this, "panel velocidad1"); //buscamos el
panel de dial solicitado
        componente = (JSlider)buscaComponente((JPanel)componente,
"velocidad1");
```

```
    }
    else { //dial2
      componente = buscaComponente(this, "panel velocidad2"); //buscamos el panel de dial solicitado
      componente = (JSlider)buscaComponente((JPanel)componente, "velocidad2");
    }
    ((JSlider)componente).setValue(velocidad);
    return true;
  }
  catch (NullPointerException e){
    return false;
  }
  catch (ClassCastException e){
    return false;
  }
 }

 public void setPermisoMovimiento(boolean permiso){
  this.permiso_movimiento = permiso;

  try {
    Component[] componentes = this.getComponents();
    Component[] componente;
    int j;

    for (int i = 0; i < this.getComponentCount(); i++) {
      if (componentes[i].getName() != null) //no tiene nombre
        if (componentes[i] instanceof JPanel){  //miramos dentro de los paneles
          componente = ( (JPanel) componentes[i]).getComponents();
          for (j = 0; j < ( (JPanel) componentes[i]).getComponentCount();
              j++)
            if (componente[j] instanceof Dial) //los diales se activan de otro forma
              ((Dial)componente[j]).activaListeners(permiso);
```

```
            else
              componente[j].setEnabled(permiso);
          }
          else{
           if ((!componentes[i].getName().equals("mover")) ||(permiso == false))
             //solo actuamos sobre el boton "mover" si es para desactivarlo
             componentes[i].setEnabled(permiso);
          }
       }
     }
     catch (NullPointerException e) {}
  }

  public void setMoverEnabled(boolean permiso){ //activa/desactiva el boton "mover"
     JButton boton = (JButton)buscaComponente(this,"mover");
     boton.setEnabled(permiso);
  }

  private void addGB(Component component,int x, int y){
      constraints.gridx = x;
      constraints.gridy = y;
      add(component,constraints);
    }

  private String cuentaRevoluciones(int velocidad,int n_pasos){
    Double rev = new Double(60.0 / (((0xFF - velocidad+1) * 256 * 0x40 *
                                4.0/6000000.0) * (2.0 * n_pasos)));
    String cadena = new String(rev.toString());
    if (cadena.length() >= cadena.indexOf('.') + 2) //1 o mas decimales
       return cadena.substring(0, cadena.indexOf('.') + 3) + " rpm";
    else
      return cadena + " rpm";
  }
```

```
 private Component buscaComponente(JPanel panel,String nombre){
  try {
   Component[] componentes = panel.getComponents();

   for (int i = 0; i < panel.getComponentCount(); i++) {
    if (componentes[i].getName() != null) //sobre todo por las etiquetas sin
nombre
      if (componentes[i].getName().equals(nombre))
       return componentes[i]; //se ha encontrado
   }
   return null; //no se ha encontrado
  }
  catch (NullPointerException e) {
   return null;
  }
 }

 public void addPanelMovimientoListener(PanelMovimientoListener listener){
  listenerList.add(PanelMovimientoListener.class, listener);
 }

 public void removePanelMovimientoListener(PanelMovimientoListener listener) {
  listenerList.remove(PanelMovimientoListener.class, listener);
 }

 void avisaMovimiento(int paso1, int paso2, int velocidad1, int velocidad2){
  Object[] listeners = listenerList.getListenerList();
  for(int i = 0;i < listeners.length; i += 2 )
   if (listeners[i] == PanelMovimientoListener.class)
    ((PanelMovimientoListener)listeners[i+1]).moverMotor(
     new PanelMovimientoEvent(this,paso1 * getAnguloPaso((short)1) * 1.0,
                              paso2 * getAnguloPaso((short)2) * 1.0,
                              velocidad1, velocidad2));
```

```
  }

  void avisaMovimiento(int pasoX, int velocidadX, short ID){
     Object[] listeners = listenerList.getListenerList();
     for(int i = 0;i < listeners.length; i += 2 )
       if (listeners[i] == PanelMovimientoListener.class)
         ((PanelMovimientoListener)listeners[i+1]).moverMotor(
           new PanelMovimientoEvent(this,pasoX * getAnguloPaso(ID) * 1.0,
                                    velocidadX, ID));
    }

  private synchronized double getAnguloPaso(short ID){
    if (ID == 1)
      return anguloPaso1;
    else
      return anguloPaso2;
  }
}
```

PanelMovimientoEvent.java

```
public class PanelMovimientoEvent extends java.util.EventObject{
    public double paso1;       //se cogen directamente
    public double paso2;
    public int velocidad1;
    public int velocidad2;
    public short ID; /**0 : Ambos Motores
                       1 : Motor 1
                       2 : Motor 2
                     */

    public PanelMovimientoEvent (PanelMovimiento source, double paso1,
                                 double paso2, int velocidad1, int velocidad2){
      super(source);
```

```
      this.paso1 = paso1;
      this.paso2 = paso2;
      this.velocidad1 = velocidad1;
      this.velocidad2 = velocidad2;
      this.ID = 0;
    }

    public PanelMovimientoEvent (PanelMovimiento source, double pasoX,
                                 int velocidadX, short ID){
      super(source);
      this.ID = ID;

      if (ID == 1){
        this.paso1 = pasoX;
        this.velocidad1 = velocidadX;
      }
      else{
        this.paso2 = pasoX;
        this.velocidad2 = velocidadX;
      }
    }
}

 interface PanelMovimientoListener extends java.util.EventListener {
   void moverMotor(PanelMovimientoEvent e);
 }
```

PanelLoginEvent

```
import javax.swing.*;

public class PanelLoginEvent extends java.util.EventObject{
  public String login;
```

```
  public char[] password;
  public short permiso;

  public PanelLoginEvent(JPanel source, String login,
                         short permiso) {
    super(source);
    this.login = login;

    this.permiso = permiso;
  }

  public PanelLoginEvent(JPanel source, String login, char[] password,
                         short permiso) {
    super(source);
    this.login = login;

    this.password = password;
    this.permiso = permiso;
}
}

interface PanelLoginListener extends java.util.EventListener {
    void loged(PanelLoginEvent e); //tanto para login como para logout
  }
```

**********GESTION DE RED**********

conexiones.java

```
import java.util.*;
import java.net.*;

class entradaConexion{ //indica cada una de las entradas para cada usuario
  public static final short udp = 0;
  public static final short actuador = 1;
  public static final short admin = 2;
  private String login;
  private InetAddress direccion;
  private short tipo;     //indica si es UDP (0), actuador (1), admin (2)
  private Date ultimoAcceso;

  public entradaConexion(){
    login= null;
    direccion = null;
    ultimoAcceso = null;                    //un valor conocido
  }

  public entradaConexion(entradaConexion conexion){
    try{
      this.login = new String(conexion.login);
      this.direccion = conexion.direccion;
      this.tipo = conexion.tipo;
      this.ultimoAcceso = (Date) conexion.ultimoAcceso.clone();
    }
    catch(NullPointerException e){}
  }

  public entradaConexion(String login,InetAddress direccion, short tipo){
    this.login= new String(login);
    this.direccion = direccion;
```

```
    this.tipo= tipo;
    if (tipo == 0)//udp
      this.ultimoAcceso = null;
    else
     setUltimoAcceso();
  }

  protected void setLogin(String login){
      this.login = new String(login);
  }

  protected void setTipo(short tipo){
      this.tipo = tipo;
  }

  protected void setDireccion(InetAddress direccion){
      this.direccion = direccion;
  }

  protected void setUltimoAcceso(){
  }

  protected void setInfo(String login,InetAddress direccion, short tipo){
    setLogin(login);
    setDireccion(direccion);
    setTipo(tipo);
    setUltimoAcceso();
  }

  protected String getLogin() throws NullPointerException{
    return login.toString();
  }
```

```
  protected InetAddress getDireccion() throws NullPointerException{
    return direccion;
  }

  protected short getTipo(){
    return tipo;
  }

  protected Date getUltimoAcceso() throws NullPointerException{
    return ultimoAcceso;
  }
}

//implementa la tabla de conexiones activas
 class TablaConexiones {
   public static final short udp = 0;
   public static final short actuador = 1;
   public static final short admin = 2;

   public int MAX_CONEXIONES;
   protected Vector tabla;
   private Vector listeners;

   private boolean aumentado;

   public TablaConexiones(int MAX_CONEXIONES){
      tabla = new Vector();
      this.MAX_CONEXIONES = MAX_CONEXIONES;
      aumentado = false;
   }

   protected synchronized int anadirConexion(String login,
                                    InetAddress direccion,short tipo){
     if (tabla.size() < MAX_CONEXIONES && (buscaLogin(login) ==
```

```
                MAX_CONEXIONES)) {
    if (((tipo >= 1) && (buscaTipo(tipo, 0) == MAX_CONEXIONES)
        && (tipo < 3)) || (tipo == udp)) {
      entradaConexion entrada = new entradaConexion(login, direccion, tipo);

      tabla.add(entrada);
      avisaTabla((short)0,entrada);
      return 0;
    }
    else
      return 1;
  }
  else
    return 2;
 }

 /**Valores de retorno: 0: correcto
  *                    1: ya hay una conexión del tipo especificado
  *                    2: se ha alcanzado el número máximo de conexiones
  */
//utilizado para el cambio de modo:
  protected synchronized int anadirConexion(entradaConexion nuevaEntrada){
    if (tabla.size() < MAX_CONEXIONES &&
      (buscaLogin(nuevaEntrada.getLogin()) == MAX_CONEXIONES)) {
      if (((nuevaEntrada.getTipo()>=1) && (nuevaEntrada.getTipo() < 3)
          && (buscaTipo(nuevaEntrada.getTipo(),0)== MAX_CONEXIONES))
          || (nuevaEntrada.getTipo()==udp)){
            tabla.add(nuevaEntrada);
            avisaTabla((short)0,nuevaEntrada);
            return 0;
      }
      else
        return 1;
    }
```

```
  else
    return 2;
}

protected synchronized boolean eliminarConexion(String login){
  int i;
  try{
    i=buscaLogin(login);
    entradaConexion entrada = getEntrada(i);
    tabla.remove(i);
    avisaTabla((short)1,entrada);
    return true;
  }
  catch(ArrayIndexOutOfBoundsException e){
    return false;
  }
}

protected synchronized boolean eliminarConexion(int pos){
  try{
    avisaTabla((short)1,getEntrada(pos));
    tabla.remove(pos);
    return true;
  }
  catch(ArrayIndexOutOfBoundsException e){
    return false;
  }
}

protected synchronized void desconectarTodos(){
  int i = 0;
  try{
    while ( (this.tabla.isEmpty() == false) &&
          ( (! (tabla.size() == 1)))) {
```

```
        if ( (!getEntrada(i).getLogin().equals("ADMIN")) &&
            (! (getEntrada(i).getTipo() == TablaConexiones.admin)))
          eliminarConexion(i);

        i++;
      }
    }
    catch(NullPointerException e){}
  }

  protected synchronized boolean cambioTipo(entradaConexion entrada,
                                    short tipo){
    if (((tipo >=1) && (tipo < 3)
          && (buscaTipo(tipo,0)== MAX_CONEXIONES))
          || (tipo==udp)){
        entrada.setTipo(tipo);
        avisaTabla((short)2,entrada);
        return true;
      }
    return false;
  }

  protected synchronized boolean eliminarConexion(InetAddress addr){
    int i;
    try{
      i=buscaDireccion(addr);
      if (i < tabla.size())
      {
        entradaConexion entrada = getEntrada(i);
        tabla.remove(i);
        avisaTabla( (short) 1, entrada);
        return true;
      }
      else
```

```
        return false;
      }
      catch(ArrayIndexOutOfBoundsException e){
        return false;
      }
    }

    protected synchronized boolean eliminarConexionAdmin(){
      int i;
      try{
        i = buscaTipo(admin,0);

        if (i < tabla.size())
        {
          entradaConexion entrada = getEntrada(i);
          tabla.remove(i);
          avisaTabla( (short) 1, entrada);
          return true;
        }
        else
          return false;
      }
      catch(ArrayIndexOutOfBoundsException e){
        return false;
      }
    }

  protected synchronized boolean eliminarConexionTipo(InetAddress
direccion,short tipo){
    int inicio = 0;

    try{
      do {
        inicio = buscaTipo(tipo, inicio);
```

```
     }
     while ( ( (entradaConexion) tabla.elementAt(inicio)).getDireccion().
           equals(direccion) == false);
    if (inicio < tabla.size())
    {
      entradaConexion entrada = getEntrada(inicio);
      avisaTabla( (short) 1, entrada);
      tabla.remove(inicio);
      return true;
    }
    else
      return false;
   }
   catch(ArrayIndexOutOfBoundsException e){
       return false;
    }
}

   protected int size(){
    return tabla.size();
   }

   protected entradaConexion getEntrada_login(String login){
    try{
      return (entradaConexion) tabla.elementAt(buscaLogin(login));
    }
    catch(ArrayIndexOutOfBoundsException e){
      return null;
    }
   }

    protected entradaConexion getEntrada_direccion(InetAddress direccion){
     try{
       return (entradaConexion) tabla.elementAt(buscaDireccion(direccion));
```

```
    }
    catch(ArrayIndexOutOfBoundsException e){
      return null;
    }
  }

  protected entradaConexion getEntrada(int pos){
    try{
      return (entradaConexion) tabla.elementAt(pos);
    }
    catch(ArrayIndexOutOfBoundsException e){
      return null;
    }
  }

  protected entradaConexion getEntrada_tipo(int tipo,int inicio){
    try{
      return (entradaConexion) tabla.elementAt(buscaTipo(tipo,inicio));
    }
    catch(ArrayIndexOutOfBoundsException e){
      return null;
    }
  }

  protected synchronized boolean setLogin(String oldlogin, String newlogin){
    int indice;

    indice = buscaLogin(oldlogin);
    if ((indice < MAX_CONEXIONES) &&
        (buscaLogin(newlogin) == MAX_CONEXIONES)) {
      entradaConexion entrada = (entradaConexion)tabla.elementAt(indice);

      entrada.setLogin(newlogin);
      avisaTabla((short)2,entrada);
```

```
        return true;
      }
      else
        return false;
    }

    protected synchronized boolean actualizarAcceso_login(String login){
      int indice;

      indice = buscaLogin(login);
      if (indice < MAX_CONEXIONES){
        ((entradaConexion)tabla.elementAt(indice)).setUltimoAcceso();
        return true;
      }
      else
        return false;
    }

    protected synchronized boolean actualizarAcceso_direccion(InetAddress
direccion){
      int indice;

      indice = buscaDireccion(direccion);
      if (indice < MAX_CONEXIONES){
        ((entradaConexion)tabla.elementAt(indice)).setUltimoAcceso();
        return true;
      }
      else
        return false;
    }

   public synchronized int buscaTipo(int tipo, int inicio){
    int i;
      try{
```

```
        for (i=inicio;(((entradaConexion) tabla.elementAt(i)).getTipo() != tipo);i++){}
        return i;
      }
      catch(ArrayIndexOutOfBoundsException e){
        return MAX_CONEXIONES;
      }
  }

   protected synchronized int buscaLogin(String login){
     int i;

     try{
       for (i=0;(((entradaConexion) tabla.elementAt(i)).getLogin().equals(login))
           == false; i++) {}
       return i;
     }
     catch(ArrayIndexOutOfBoundsException e){
       return MAX_CONEXIONES;
     }
   }

   private synchronized int buscaDireccion(InetAddress direccion){
     int i;

     try{
       for                                                    (i=0;(((entradaConexion)
tabla.elementAt(i)).getDireccion().equals(direccion))
           == false; i++) {}
       return i;
     }
     catch(ArrayIndexOutOfBoundsException e){
       return MAX_CONEXIONES;
     }
   }
```

```
  protected synchronized boolean crearActuadorFicticio(){
    entradaConexion entrada = getEntrada(buscaTipo(actuador,0));
    boolean habia_actuador = false;

    if (entrada != null){
      eliminarConexion(buscaTipo(actuador, 0));
      entrada.setTipo(udp);
      anadirConexion(entrada);
      habia_actuador = true;
    }

    if (tabla.size() == MAX_CONEXIONES){
      MAX_CONEXIONES++;
      aumentado = true;
    }

    try{
      anadirConexion("ADMIN", InetAddress.getLocalHost(), actuador);
    }
    catch(UnknownHostException e){
      anadirConexion("ADMIN", null, actuador);
    }

    return habia_actuador;
  }

  protected synchronized void eliminarActuadorFicticio(){
    eliminarConexion("ADMIN");
    if (aumentado == true)
      MAX_CONEXIONES--;
  }

  //*****************************Control del evento*******************************
```

```
synchronized public void addConexionListener(ConexionListener l) {
  if (listeners == null)
    listeners = new Vector();

  listeners.addElement(l);
}


synchronized public void removeConexionListener(ConexionListener l) {
  if (listeners == null)
    listeners = new Vector();

  listeners.removeElement(l);
}



protected synchronized void avisaTabla(short suceso,
                                       entradaConexion entrada) {
  if (listeners != null && !listeners.isEmpty()) {
    EventoConexion evento = new EventoConexion(this,suceso, entrada);

    Vector targets;
    synchronized (this) {
      targets = (Vector) listeners.clone();
    }

    Enumeration enum = targets.elements();
    while (enum.hasMoreElements()) {
      ConexionListener l = (ConexionListener) enum.nextElement();
      l.recibido(evento);
    }
  }
}
```

```
}
```

ServidorRed.java

```
import java.net.*;
import java.io.*;
import java.util.*;

class ServidorRed extends Thread implements PaqueteTCPListener,
                                MotorMovidoListener{
  static final short gestor = 0;
  static final short actuador = 1;
  static final short admin = 2;
  final short ref_puerto_udp = 0;
  private TablaUsuarios usuarios;
  private TablaConexiones conexiones;
  private int TCPtimeout;
  private int UDPtimeout;
  private int tiempo_revisionesUDP;
  private long timeout_actuador;
  private long timeout_gestor;

  private OpcionesMotores tablaMotores;
  private Vector peticiones;

  private ControlEventoServidor eventoServidor;
  private MoverMotor moverMotor;

  private boolean procesandoPaquete;
  private boolean finConexiones;

  private boolean actuador_autenticado;
  private PDU_TCP[] arrayTCP; //array en que se almacena la ultima pdu por
confirmar
```

```
private PaqueteTCPServidor[] tcp;  /**array de tres posiciones:
                        /* 0: conexion gestora
                        /* 1: conexion actuadora
                        /* 2: conexion administrador*/

private PaqueteUDPServidor udp;

private int puerto_gestion;       /** +0 : puerto de gestion
                         * +1 : puerto de UDP
                         * +2 : puerto de actuador*/

public ServidorRed(TablaUsuarios tablaUsuarios,
                   TablaConexiones tablaConexiones,
                   int TCPtimeout, int tiempo_conexionActuador,
                   int tiempo_gestion,int puerto_gestion,
                   int UDPtimeout, int tiempo_revisionesUDP,
                   OpcionesMotores tablaMotores,
                   ServidorListener l) throws IOException {

  this.usuarios = tablaUsuarios;
  this.conexiones = tablaConexiones;

  this.eventoServidor = new ControlEventoServidor();
  this.moverMotor = new MoverMotor();

  this.puerto_gestion = puerto_gestion;
  this.TCPtimeout = TCPtimeout;
  this.UDPtimeout = UDPtimeout;
  this.tiempo_revisionesUDP = tiempo_revisionesUDP;
  this.timeout_actuador = tiempo_conexionActuador;
  this.timeout_gestor = tiempo_gestion;

  this.eventoServidor.addServidorListener(l);
```

```
this.finConexiones = false;
this.procesandoPaquete = false;
peticiones = new Vector();

this.start();

tcp = new PaqueteTCPServidor[3];
arrayTCP = new PDU_TCP[3];

tcp[gestor] = new PaqueteTCPServidor(puerto_gestion,
                        gestor,
                        TCPtimeout,
                        timeout_gestor,
                        this.getPuerto(ref_puerto_udp),
                        this);

tcp[actuador] = null;
tcp[admin] = null;

actuador_autenticado = false;

udp = new PaqueteUDPServidor(puerto_gestion,
                        UDPtimeout,
                        TCPtimeout,
                        timeout_actuador,
                        tiempo_revisionesUDP,
                        tablaUsuarios,
                        conexiones,
                        tcp,
                        this,
                        eventoServidor);

eventoServidor.avisaServidor((short)4,(short)0);
```

```
    this.tablaMotores = tablaMotores;
    }

    public void close() {
      if (tcp[gestor] != null)
       tcp[gestor].close();

      if (tcp[actuador] != null)
       tcp[actuador].close();

      if (tcp[admin] != null)
       tcp[admin].close();

      if (udp != null)
       udp.close();

      setfinConexiones(true);

      conexiones.desconectarTodos();

      eventoServidor.avisaServidor((short)5,(short)0);
    }

 private int getPuerto(short tipo_puerto){//NO para el puerto gestor (tiene su propia
variable)
   return (puerto_gestion+tipo_puerto+1);
   }

private synchronized boolean getfinConexiones(){
   return finConexiones;
}

private synchronized void setfinConexiones(boolean estado){
  finConexiones = estado;
```

```
}

 public void recibido(EventoPaqueteTCP paquete){
  anadePeticion(paquete);
}

private synchronized void anadePeticion(EventoPaqueteTCP paquete){
   if (peticiones.isEmpty() == false)
     peticiones.add(paquete);
   else{
     peticiones.add(paquete);
     if (getProcesando() == false) //no hay nada en el vector, y no está procesando
una peticion : esta parado
       notify(); //despertamos al thread
   }
}

private synchronized void esperar(){
  try{
    if (peticiones.isEmpty() == true){
      setProcesando(false); //asi esperamos por otro paquete
      wait();
    }
  }
  catch (InterruptedException e) {}
}

private synchronized Object getPeticion(){
  try{
    if (peticiones.isEmpty())
      return null;
    else{
      setProcesando(true);
      Object temp = peticiones.elementAt(0); //siempre el primer elemento
```

```
      peticiones.remove(0); //una vez cogido, eliminamos esa posicion
      return temp;
    }
  }
  catch(ArrayIndexOutOfBoundsException e){
    return null;  //en teoria no puede pasar nunca
  }
}

private synchronized void setProcesando(boolean estado){
  procesandoPaquete = estado;
}

private synchronized boolean getProcesando(){
  return procesandoPaquete;
}

  public void run(){
  PDU_TCP pdu = new PDU_TCP();
  EventoPaqueteTCP paquete = null;

  while (getfinConexiones() == false)
  {
    try {
      esperar(); //esperamos a que llegue otro paquete

      paquete = (EventoPaqueteTCP)getPeticion();

      switch (paquete.suceso) { //primero clasificamos segun el suceso
        case 0: //recibido correctamente
          switch (paquete.pdu.getTipo()) { //primero miramos los paquetes comunes
            case 0:
              mensajesActuador(paquete.pdu);
              break;
```

```
        case 3: //Paquetes de información de la conexion
          switch (paquete.pdu.getSubtipo()) { //paquetes de info de conexion
            case 0: //¿estas ahi?
              pdu.setTipo(3, 1); //sigo aqui
              pdu.addDato("Hola, soy el servidor del PFC");
              tcp[paquete.id].enviar(pdu);
              break;
            case 1: //si, sigo aqui
              if ( (arrayTCP[paquete.id].getTipo() == 3) && //ultimo paquete enviado
                  (arrayTCP[paquete.id].getSubtipo() == 0))
                arrayTCP[paquete.id] = null;
              break;
            case 3: //error en el último paquete enviado
              eventoServidor.avisaServidor(paquete.id, (short) 2); //avisamos al
gestor principal
              break;
          }
          break;
        case 5: //paquetes de desconexion
          switch (paquete.pdu.getSubtipo()) {
            case 0: //desconexion
              if (paquete.id != gestor)
                desconectarTCP(paquete.id, null);
              else {
                reiniciarConexionTCP(gestor);
              }
              break;
          }
          break;
        default: //paquetes concretos segun servicio
          switch (paquete.id) {
            case gestor:
              mensajesGestor(paquete.pdu);
              break;
```

```
            case actuador:
              mensajesActuador(paquete.pdu);
              break;
            }
            break;
          }
          break;

        case 1: //timeout
          if (arrayTCP[paquete.id] != null) {
            if ( (arrayTCP[paquete.id].getTipo() == 3) && //ultimo paquete enviado
              (arrayTCP[paquete.id].getSubtipo() == 0)) {
              if (paquete.id == gestor) { //es la conexion gestora
                reiniciarConexionTCP(gestor);
              }
              else { //es cualquiera de las otras conexiones
                pdu.setTipo(5, 2); //pues ya lo habiamos mandado con anterioridad
                desconectarTCP(paquete.id, pdu);//null);
              }
            }
            else { //AUN no se ha enviado el paquete que espera confirmacion
              pdu.setTipo(3, 0); //pdu de timeout
              pdu.addDato("Hola, soy el servidor del PFC");
              tcp[paquete.id].enviar(pdu);
              tcp[paquete.id].enviar(arrayTCP[paquete.id]); //reenviamos la pdu sin
respuesta
              arrayTCP[paquete.id] = pdu; //añadimos al listado la pdu
            }
          }
          else { //aun no se ha enviado el paquete que espera confirmacion
            //Y no hay ninguna pdu sin respuesta
            pdu.setTipo(3, 0);
            pdu.addDato("Hola, soy el servidor del PFC");
            tcp[paquete.id].enviar(pdu);
```

```
          arrayTCP[paquete.id] = pdu; //añadimos al listado
        }
        break;

      case 2: //Error
        if (paquete.id == gestor) { //es la conexion gestora
          reiniciarConexionTCP(gestor);
        }
        else //es cualquiera de las otras conexiones
          desconectarTCP(paquete.id, null); //NO enviamos nada (error en la
conexion)

        eventoServidor.avisaServidor(paquete.id, (short) 1); //Informamos del cierre
        break;

      case 3: //conexion cerrada (socket cerrado: Timeout, caida del extremo
remoto...)
        if (paquete.id == gestor)
          reiniciarConexionTCP(gestor);
        else
          desconectarTCP(paquete.id, null); //no enviamos ningun mensaje, pues el
socket ya esta cerrado
        break;
      case 4: //finaliza la conexion, pues se ha agotado su tiempo (NO a nivel TCP)
        switch(paquete.id){
          case gestor:
            arrayTCP[gestor] = null;
          break;
          case actuador:
            entradaConexion entrada;
            PDU_UDP pdu_udp = new PDU_UDP();

         try{
           //guardamos una copia de la entrada que deja TCP y pasa a UDP
```

```
            entrada = conexiones.getEntrada(conexiones.buscaTipo(paquete.id, 0));
            //eliminamos la conexion
            conexiones.eliminarConexion(conexiones.buscaTipo(paquete.id, 0));
            entrada.setTipo(conexiones.udp); //lo pasamos a UDP
            conexiones.anadirConexion(entrada);

            pdu.setTipo(3, 2); //avisamos de la disponibilidad de la conexion TCP
            pdu.addDato(0);  //indicamos que SI hay disponibilidad
            udp.multicast(pdu_udp);
            actuador_autenticado = false; //para el proximo
           }
           catch (NullPointerException e) {}
           break;
         }
         break;
      }
    }
    catch (NullPointerException e) { //se ha recibido una pdu con un numero
incorrecto
      //de parámetros
      if (paquete != null)
        if (tcp[paquete.id] != null) { //indica que aun no se ha creado la conexion
(avisara por otros métodos)
          pdu.setTipo(3, 3);
          tcp[paquete.id].enviar(pdu);
        }
    }
    catch (NumberFormatException e) { //algun parametro en formato numerico es
incorrecto
      pdu.setTipo(3, 3);
      tcp[paquete.id].enviar(pdu);
    }
    catch (IOException e) {
      eventoServidor.avisaServidor(paquete.id, (short) 1);
```

```
    }
  }
}

protected void permitirActuador(boolean permitir){
  try{
    if (permitir == true) { //sí se permite
      conexiones.eliminarActuadorFicticio();

      PDU_UDP pduUDP = new PDU_UDP();
      pduUDP.setTipo(3, 2);
      pduUDP.addDato(0); //indicamos que SI hay disponibilidad
      udp.multicast(pduUDP);
    }

    else { //NO está permitido
      usuarios.anadirUsuario("ADMIN", "Administrador", "",
                        TablaUsuarios.actuador);

      if (conexiones.crearActuadorFicticio() == true) { //habia un actuador,
//lo despedimos
        PDU_TCP pdu = new PDU_TCP();
        pdu.setTipo(5, 3); //solicitud por orden del administrador
        pdu.addDato(getPuerto(ref_puerto_udp)); //indicamos cual es el puerto UDP
a utilizar
        tcp[ServidorRed.actuador].enviar(pdu); //enviar despedida

        //eliminamos todo rastro del actuador
        actuador_autenticado = false; //para el próximo
        tcp[ServidorRed.actuador].close();
        arrayTCP[ServidorRed.actuador] = null;
      }

      //indicamos que ya se ha cubierto la plaza de actuador
```

```
      PDU_UDP pduUDP = new PDU_UDP();
      pduUDP.setTipo(3, 2);
      pduUDP.addDato(1); //indicamos que NO hay disponibilidad
      udp.multicast(pduUDP);
    }
  }
  catch(NullPointerException e){}
}

protected synchronized void desconectarTCP_GUI(int id){
  PDU_TCP pdu = new PDU_TCP();
  pdu.setTipo(5,0);

  desconectarTCP(id,pdu);
}

protected synchronized void desconectarUDP_GUI(entradaConexion entrada){
  PDU_UDP pdu = new PDU_UDP();
  pdu.setTipo(5,0);

  udp.enviar(pdu,entrada.getDireccion());
  conexiones.eliminarConexion(entrada.getLogin());
}

private synchronized entradaConexion desconectarTCP(int id,PDU_TCP motivo){
  entradaConexion entrada;
  try{
    entrada = conexiones.getEntrada(conexiones.buscaTipo(id, 0));
    conexiones.eliminarConexion(conexiones.buscaTipo(id, 0));

    if ( (id == actuador) && (arrayTCP[actuador] == null)) { //ya se haya autenticado
      PDU_UDP pdu = new PDU_UDP();
      pdu.setTipo(3, 2);
      pdu.addDato(0);
```

```
      udp.multicast(pdu);
      actuador_autenticado = false; //para el nuevo

    }
    else {
      if ( (id == actuador) && ((arrayTCP[actuador].getTipo() != 1)
      || ((arrayTCP[actuador].getSubtipo() != 1)))) {
        PDU_UDP pdu = new PDU_UDP();
        pdu.setTipo(3, 2);
        pdu.addDato(0);
        udp.multicast(pdu);
        actuador_autenticado = false;
      }
      else { //al no estar autenticado, no podemos cerrar la conexion pq alguien
        //se haya metido, seguimos esperando al que nos interesa
        reiniciarConexionTCP(actuador);
        return entrada;
      }
    }

    if (motivo != null)
      tcp[id].enviar(motivo);
    tcp[id].close();
    arrayTCP[id] = null;
    return entrada;
  }
  catch(NullPointerException e){
    return null;
  }
}

private synchronized void reiniciarConexionTCP(short id){
 switch(id){
   case gestor:
```

```
    tcp[gestor].reiniciar();
    arrayTCP[gestor]=null;
    break;

   case actuador:
     tcp[gestor].reiniciar();
     arrayTCP[actuador]=null;
     break;
  }
}

private void mensajesGestor(PDU_TCP pdu_recibida)
   throws NullPointerException,NumberFormatException,IOException{
  PDU_TCP pdu = new PDU_TCP();

  if (pdu_recibida.getTipo() == 1){//gestion de login
     if (pdu_recibida.getSubtipo() == 0) {
       entradaUsuario usuario;
       arrayTCP[gestor] = null;

       usuario = usuarios.getEntrada(pdu_recibida.getCampo(0));
       if (usuario != null) {
         if (pdu_recibida.getCampo(1).equals(usuario.getPassword())) {
           if (pdu_recibida.getIntcampo(2) <= usuario.getPermisos()){
             switch(conexiones.anadirConexion(pdu_recibida.getCampo(0),
                                 tcp[gestor].getDireccion(),
                                 (short)pdu_recibida.getIntcampo(2))){
             case 0:
               tcp[gestor].enviar(getConfiguracionMotor());

               short tipo = (short)pdu_recibida.getIntcampo(2);
               pdu.setTipo(1,1);
               pdu.addDato(tipo);
               pdu.addDato(getPuerto(tipo));
```

```
    usuarios.actualizarAcceso(pdu_recibida.getCampo(0));

    tcp[gestor].enviar(pdu);

    if (tipo > ref_puerto_udp){
         tcp[tipo] = new PaqueteTCPServidor(getPuerto(tipo),
          tipo, TCPtimeout, timeout_actuador,
          this.getPuerto(ref_puerto_udp),this);

       if (tipo == actuador)
       arrayTCP[tipo]=pdu;

       PDU_UDP pduUDP = new PDU_UDP();
       pduUDP.setTipo(3, 2);
       pduUDP.addDato(1);
       udp.multicast(pduUDP);
    }

    reiniciarConexionTCP(gestor);
    break;
  case 1:
    if (conexiones.anadirConexion(pdu_recibida.getCampo(0),
                        tcp[gestor].getDireccion(),
                        conexiones.udp) == 0){
       tcp[gestor].enviar(getConfiguracionMotor());

       pdu.setTipo(1,1);
       pdu.addDato(conexiones.udp);
       pdu.addDato(getPuerto(ref_puerto_udp));
       tcp[gestor].enviar(pdu);
    }
    else{
      pdu.setTipo(1,2);
      pdu.addDato(4);
```

```
      tcp[gestor].enviar(pdu);
    }
    reiniciarConexionTCP(gestor);
    break;
  case 2:
    pdu.setTipo(1,2);
    pdu.addDato(4);
    tcp[gestor].enviar(pdu);
    reiniciarConexionTCP(gestor);
    break;
  }
}
else{
  if (conexiones.anadirConexion(pdu_recibida.getCampo(0),
                        tcp[gestor].getDireccion(),
                        conexiones.udp) == 0){

      tcp[gestor].enviar(getConfiguracionMotor());

      pdu.setTipo(1,1);
      pdu.addDato(conexiones.udp);
      pdu.addDato(getPuerto(ref_puerto_udp));
      tcp[gestor].enviar(pdu);
      reiniciarConexionTCP(gestor); //una vez conectado, reiniciamos el
                                    //gestor
    }
    else{
      pdu.setTipo(1,2);
      pdu.addDato(4);
      tcp[gestor].enviar(pdu);
      reiniciarConexionTCP(gestor);//una vez conectado, reiniciamos el
                                   //gestor
    }
}
```

```
      }
      else { //PASSWORD INCORRECTO
        pdu.setTipo(1,2);
        pdu.addDato(1);
        tcp[gestor].enviar(pdu);
      }
     }
     else{
       pdu.setTipo(1,2);
       pdu.addDato(0);
       tcp[gestor].enviar(pdu);
     }
    }
 }
}

 public PDU_TCP getConfiguracionMotor(){
   PDU_TCP pdu = new PDU_TCP();

   pdu.setTipo(2, 0); //enviamos la configuracion de los motores

   pdu.addDato( (int) (360.0 / tablaMotores.getAnguloPaso(1))); //N_PASOS 1
   pdu.addDato( (int) (360.0 / tablaMotores.getAnguloPaso(2))); //N_PASOS 2
   pdu.addDato(tablaMotores.getAnguloActual(1)); //ANGULO 1 (incial)
   pdu.addDato(tablaMotores.getAnguloActual(2)); //ANGULO 2 (incial)
   pdu.addDato(tablaMotores.getVelocidadMaxima(1));
   pdu.addDato(tablaMotores.getVelocidadMaxima(2));
   pdu.addDato(tablaMotores.getVelocidad(1));
   pdu.addDato(tablaMotores.getVelocidad(2));
   pdu.addDato(tablaMotores.getFlag(1)); //flag 1
   if (tablaMotores.getFlag(1) == 2) {
     //define un ángulo máximo (en número de pasos)
     pdu.addDato(tablaMotores.getAnguloMaximo(1));
   }
```

```
   pdu.addDato(tablaMotores.getFlag(2));
   if (tablaMotores.getFlag(2) == 2) {
     pdu.addDato(tablaMotores.getAnguloMaximo(2));
   }
   return pdu;
 }

private void mensajesActuador(PDU_TCP pdu_recibida)
throws NullPointerException,NumberFormatException,IOException{
  PDU_TCP pdu = new PDU_TCP();

  if ((actuador_autenticado == true) || pdu_recibida.getTipo()==0)
  {//se ha autenticado anteriormente ó pretende autenticarse
    switch (pdu_recibida.getTipo()) {
      case 0:
        entradaConexion conexion =
conexiones.getEntrada_login(pdu_recibida.getCampo(0));
        if (conexion != null) {
          if ( (conexion.getTipo() == actuador) &&
              (conexion.getDireccion().equals(tcp[actuador].getDireccion()))) {
            arrayTCP[actuador] = null;
            actuador_autenticado = true;
          }
          else {
            pdu.setTipo(5, 0);
            pdu.addDato(2);
            desconectarTCP(actuador, pdu);
          }
        }
        else {
          pdu.setTipo(5, 0);
          pdu.addDato(2);
          desconectarTCP(actuador, pdu);
        }
```

```
      break;
    case 2:
      switch (pdu_recibida.getSubtipo()) {
        case 1:
          moverMotor.avisaMotor(pdu_recibida.getDoubleCampo(0),
                                pdu_recibida.getDoubleCampo(1),
                                pdu_recibida.getIntcampo(2),
                                pdu_recibida.getIntcampo(3));
          break;
        case 2:
          moverMotor.avisaMotor(pdu_recibida.getDoubleCampo(0),
                                pdu_recibida.getIntcampo(1),
                                (short)pdu_recibida.getIntcampo(2));
          break;
      }
      break;
    case 4:
      if (pdu_recibida.getSubtipo() == 0) {
        String login = conexiones.getEntrada_tipo(actuador, 0).getLogin();
        InetAddress direccion = tcp[actuador].getDireccion();

        pdu.setTipo(4, 2);
        pdu.addDato(getPuerto(ref_puerto_udp));
        desconectarTCP(actuador, pdu);
        conexiones.anadirConexion(login,
                                  direccion,
                                  (short) 0 /*UDP*/);
        break;
      }
  }
}
else{ //no se ha autenticado, y pretende hacer alguna operacion
  pdu.setTipo(3,3);
  tcp[actuador].enviar(pdu);
```

```
    pdu.setTipo(5,0);//Lo expulsamos
    pdu.addDato(2);
    desconectarTCP(actuador,pdu);
  }
}

  public void addMotorListener(MoverMotorRedListener l){
    moverMotor.addMoverMotorListener(l);
  }

  public void removeMotorListener(MoverMotorRedListener l){
    moverMotor.removeMoverMotorListener(l);
  }

  public void motorMovido(MotorMovidoEvent ev){// el USB ha respondido al
movimiento: Avisamos
    PDU_UDP pduUDP = new PDU_UDP();
    PDU_TCP pduTCP = new PDU_TCP();

    if ((ev.tipo == 1) || (ev.tipo == 2)){
      if (tcp[actuador] != null){//si hay un actuador
        pduTCP.setTipo(2,1);
        pduTCP.addDato(tablaMotores.getAnguloActual(1));//paso 1
        pduTCP.addDato(tablaMotores.getAnguloActual(2));//paso 2
        pduTCP.addDato(tablaMotores.getVelocidad(1)); //velocidad 1
        pduTCP.addDato(tablaMotores.getVelocidad(2)); //velocidad 2

        tcp[actuador].enviar(pduTCP); //avisamos al actuador
      }

      //avisamos a los espectadores
      pduUDP.setTipo(2,1);
      pduUDP.addDato(tablaMotores.getAnguloActual(1));//paso 1
      pduUDP.addDato(tablaMotores.getAnguloActual(2));//paso 2
```

```
      pduUDP.addDato(tablaMotores.getVelocidad(1)); //velocidad 1
      pduUDP.addDato(tablaMotores.getVelocidad(2)); //velocidad 2

      udp.multicast(pduUDP);
    }
    else{//ha habido un error al acceder al dispositivo
      if (tcp[actuador] != null){//si hay un actuador
        pduTCP.setTipo(2, 3);
        tcp[actuador].enviar(pduTCP);
      }
    }
  }
}

//------------------------PAQUETE UDP-----------------------------------

class PaqueteUDPServidor extends Thread implements PaqueteUDPListener{
final short gestor = 0;
final short actuador = 1;
final short admin = 2;
final short ref_puerto_udp = 0;


  private PaqueteUDPtransmision tx;
  private PaqueteUDPrecepcion rx;
  private TablaConexiones conexiones;
  private TablaUsuarios usuarios;
  private int tiempo_revisiones;
  private int UDPtimeout;

  private int TCPtimeout; //utilizado para el paso de UDP a TCP
  private long tiempoConexionTCP; //utilizado para el paso de UDP a TCP
  private PaqueteTCPListener listenerTCP;//utilizado para el paso de UDP a TCP
```

```
private DatagramSocket socket;

private Timer timerComprobacion;
private Timer timerRevision;
private Date ultimaRevision;

private boolean procesandoPaquete;
private boolean finConexiones;
private int puerto_gestion;
private ControlEventoServidor eventoServidor;
private PaqueteTCPServidor[] tcp;

private Vector peticiones; //para el control de los paquetes en el thread

public PaqueteUDPServidor(int puerto_gestion, int UDPtimeout, int TCPtimeout,
                          long tiempoConexionTCP,
                          int tiempo_revisiones,
                          TablaUsuarios usuarios,
                          TablaConexiones conexiones,
                          PaqueteTCPServidor[] tcp,
                          PaqueteTCPListener listenerTCP,
                          ControlEventoServidor eventoServidor)
                          throws IOException{
  try{
    this.eventoServidor = eventoServidor;
    this.tcp = tcp;
    this.tiempo_revisiones = tiempo_revisiones;
    this.UDPtimeout = UDPtimeout;

    this.TCPtimeout = TCPtimeout;
    this.tiempoConexionTCP = tiempoConexionTCP;
    this.listenerTCP = listenerTCP;

    this.puerto_gestion = puerto_gestion;
```

```
    socket = new DatagramSocket(getPuerto(ref_puerto_udp));
    tx = new PaqueteUDPtransmision(getPuerto(ref_puerto_udp),socket);
    rx = new PaqueteUDPrecepcion(socket);
    this.conexiones = conexiones;
    this.usuarios = usuarios;

    this.procesandoPaquete = false;
    this.finConexiones = false;
    peticiones = new Vector();//inicializamos el vector de peticiones

    rx.addUDPListener(this);
    rx.start();                  //comienza la escucha
    ultimaRevision = new Date();
    setTimerTask();
    start();
  }
  catch(IOException e){throw e;}
}

 private void setTimerTask() {
   timerRevision = new Timer();
   TimerTask taskRevision = new TimerTask() {
     public void run() {
       revision();
     }
   };
   timerRevision.scheduleAtFixedRate(taskRevision, (long) tiempo_revisiones,
                   (long) (2*tiempo_revisiones));

   timerComprobacion = new Timer();
   TimerTask taskComprobacion = new TimerTask() {
     public void run() {
       comprobacion();
     }
```

```
      };
      timerRevision.scheduleAtFixedRate(taskComprobacion,
                        (long) (2*tiempo_revisiones)+(UDPtimeout),
                        (long) (2*tiempo_revisiones)+(UDPtimeout));
      }

      private synchronized boolean getfinConexiones(){
        return finConexiones;
      }

      private synchronized void setfinConexiones(boolean estado){
       finConexiones = estado;
      }

 public void recibido(EventoPaqueteUDP paquete){
  anadePeticion(paquete);
}

private synchronized void anadePeticion(EventoPaqueteUDP paquete){
   if (peticiones.isEmpty() == false)
     peticiones.add(paquete);
   else{
     peticiones.add(paquete);
     if (getProcesando() == false) //no hay nada en el vector, y no está procesando
una peticion : esta parado
       notify(); //despertamos al thread
   }
}

private synchronized void esperar(){
  try{
    if (peticiones.isEmpty() == true){
      setProcesando(false); //asi esperamos por otro paquete
      wait();
```

```
    }
  }
  catch (InterruptedException e) {}
}

private synchronized Object getPeticion(){
  try{
    if (peticiones.isEmpty())
      return null;
    else{
      setProcesando(true);
      Object temp = peticiones.elementAt(0); //siempre el primer elemento
      peticiones.remove(0); //una vez cogido, eliminamos esa posicion
      return temp;
    }
  }
  catch(ArrayIndexOutOfBoundsException e){
    return null;  //en teoria no puede pasar nunca
  }
}

private synchronized void setProcesando(boolean estado){
  procesandoPaquete = estado;
}

private synchronized boolean getProcesando(){
  return procesandoPaquete;
}

  public void run(){
    EventoPaqueteUDP paquete = null;
    PDU_UDP pdu = new PDU_UDP();

    while (getfinConexiones() == false)
```

```
{
  try {
    esperar(); //esperamos a que llegue otro paquete

    paquete = (EventoPaqueteUDP)getPeticion();

    switch (paquete.suceso) {
       case 0:
         switch (paquete.pdu.getTipo()) {
           case 3:
             switch (paquete.pdu.getSubtipo()) {
               case 0:
                 pdu.setTipo(3, 1);
                 pdu.addDato("Hola, soy el servidor del PFC");
                 enviar(pdu, paquete.addr);
                break;
               case 1:
                 //Nos limitamos a actualizar el último acceso (al final de la funcion)
                 break;
               case 3:
                 eventoServidor.avisaServidor( (short) 3, (short) 2);
                 conexiones.eliminarConexion(paquete.addr);
                 break;
             }
             break;
             case 4:
               if (paquete.pdu.getSubtipo() == 1) {
                 entradaConexion conexion = conexiones.getEntrada_direccion(
                    paquete.addr);
                 if (conexion != null) {
                   if ((usuarios.getEntrada(conexion.getLogin()).
                       getPermisos() > TablaUsuarios.udp) &&
                         conexiones.cambioTipo(conexion,
                           TablaConexiones.actuador)){
```

```
            try{
              tcp[actuador] = new PaqueteTCPServidor(getPuerto(actuador),
                  actuador,
                  TCPtimeout,
                  tiempoConexionTCP,
                  getPuerto(ref_puerto_udp),
                  listenerTCP);
            }
            catch(IOException e){
              pdu.setTipo(4, 3);
              pdu.addDato(2);
              enviar(pdu, conexion.getDireccion());
              break;
            }
            pdu.setTipo(4, 2);
            pdu.addDato(getPuerto(actuador));
            enviar(pdu, conexion.getDireccion());
          }
          else{
            pdu.setTipo(4, 3);
            pdu.addDato(2);
            enviar(pdu, conexion.getDireccion());
            break;
          }
        }
        else{
          pdu.setTipo(4, 3);
          pdu.addDato(2);
          enviar(pdu, conexion.getDireccion());
          break;
        }
      }
      break;
    case 5:
```

```
                if (paquete.pdu.getSubtipo() == 0)
                  conexiones.eliminarConexion(paquete.addr);
                break;
              }
              break;
            case 1: //Timeout
              eventoServidor.avisaServidor( (short) 3, (short) 0);
              break;
            case 2: //Error I/O
              eventoServidor.avisaServidor( (short) 3, (short) 1);
              break;
          }
            conexiones.actualizarAcceso_direccion(paquete.addr); //sea cual sea el
paquete,
        }
        catch (NullPointerException e) { //se ha recibido una pdu con un numero
incorrecto
          //de parámetros
          pdu.setTipo(3,3);
          enviar(pdu,
          conexiones.getEntrada_direccion(paquete.addr).getDireccion());
        }
        catch (NumberFormatException e) { //algun parametro en formato numerico
es incorrecto
          pdu.setTipo(3,3);
          enviar(pdu,
          conexiones.getEntrada_direccion(paquete.addr).getDireccion());
        }
      }
    }

  private int getPuerto(short tipo_puerto){
   return (puerto_gestion+tipo_puerto+1);
  }
```

```
public synchronized boolean enviar(PDU_UDP pdu, InetAddress addr){
  if(tx.setAddress(addr) == 0){
    tx.enviar(pdu);
    return true;
  }
  else
    return false;
}

public synchronized boolean multicast(PDU_UDP pdu){
  int i = 0;
 entradaConexion entrada;
 boolean correcto = true;

  try{
    while (correcto == true) {
      i = conexiones.buscaTipo(conexiones.udp, i);
      entrada = conexiones.getEntrada(i);
      correcto=enviar(pdu, entrada.getDireccion());
      i++;
    }
    return correcto;
  }
  catch(NullPointerException e){
    return correcto;
  }
}

private void revision(){
  PDU_UDP pdu = new PDU_UDP();

  pdu.setTipo(3,0);
  pdu.addDato("Hola, soy el servidor del PFC");
```

```
  multicast(pdu);

  if (conexiones.buscaTipo(TablaConexiones.actuador,0) ==
      conexiones.MAX_CONEXIONES){
    pdu.setTipo(3, 2);
    pdu.addDato(0);
    multicast(pdu);
  }
}

private void comprobacion(){
  PDU_UDP pdu = new PDU_UDP();
  int j = 0;

  try {
    pdu.setTipo(5, 2);

    while (true) {
      j = conexiones.buscaTipo(conexiones.udp, j);
   if ((conexiones.getEntrada(j).getUltimoAcceso() != null) &&
       ((conexiones.getEntrada(j).getUltimoAcceso()).before(
          ultimaRevision))){
        enviar(pdu, conexiones.getEntrada(j).getDireccion());
        conexiones.eliminarConexion(j);
      }
      else
        j++;
    }

  }
  catch(NullPointerException e){}
    ultimaRevision = new Date();
  }
```

```
  public synchronized void close(){
    try{
      PDU_UDP pdu = new PDU_UDP();
      pdu.setTipo(5,0);
      multicast(pdu);

      socket.close();
      if (timerRevision != null)
        timerRevision.cancel();
      if (timerComprobacion != null)
        timerComprobacion.cancel();
      setfinConexiones(true);
      if (socket != null)
        socket.close();
    }
    catch (Exception e){}
  }
}

//------------------------PAQUETE TCP-------------------------------------

class PaqueteTCPServidor extends Thread{
  final short gestor = 0;
  final short actuador = 1;
  final short admin = 2;

  private PaqueteTCPtransmision tx;
  private PaqueteTCPrecepcion rx;
  private PaqueteTCPrecepcion avisos;
  private PaqueteTCPListener listener;

  private Socket socket;

  private int puerto_udp;
```

```
  private int socketTimeout;
  private ServerSocket serverSocket;
  private long tiempo_conexion;     //para asi poder contar segundos
  //private PDU_TCP pdu;  //utilizada en labores de mantenimiento
  private short id;

  private boolean finalizar;
  private boolean reiniciar;   //indicará si está en estado de conexion ó está
dormido
  private PDU_TCP pdu;

    public PaqueteTCPServidor (int puerto, short id, int socketTimeout,
                        long tiempo_conexion, int puerto_udp,
                        PaqueteTCPListener l)
                        throws IOException{
      this.socketTimeout = socketTimeout;
      this.id = id;
      //this.puerto = puerto;
      this.puerto_udp = puerto_udp;
      this.tiempo_conexion = tiempo_conexion;

      this.finalizar = false;
      this.reiniciar = false;
      this.pdu = new PDU_TCP();

      this.serverSocket = new ServerSocket(puerto);
      this.serverSocket.setReuseAddress(true);

      if (id == gestor) //conexion gestora
        serverSocket.setSoTimeout(0); /**No hay timeout por la espera de
conexiones*/
      else
        serverSocket.setSoTimeout(socketTimeout);
```

```
    this.listener=l;
    rx = new PaqueteTCPrecepcion();
    avisos = new PaqueteTCPrecepcion();
    avisos.setParam(id,null);
    avisos.addTCPListener(listener);

    start();//inicia la comunicacion
  }

  public void run (){
    try{
      do
      {
        if (getReiniciar() == true){
          prepararAntesReiniciar();
          setReiniciar(false);
          rx = new PaqueteTCPrecepcion();
        }

        rx.addTCPListener(listener);

        socket = serverSocket.accept();

        rx.setParam(id, socket);
        socket.setSoTimeout(socketTimeout);
        tx = new PaqueteTCPtransmision(socket);
        rx.start();

        esperar();

        if ((getFinalizar() == false) && (getReiniciar() == false))
        {
        switch (id){
          case gestor:
```

```
            pdu.setTipo(5, 0); //solicitud de desconexion
            enviar(pdu);
          break;

          case actuador:
            pdu.setTipo(5, 1); //solicitud de cambio por tiempo agotado
            pdu.addDato(puerto_udp);
            enviar(pdu);
           break;
          }
          avisos.avisaRecepcion((short) 4,id,null);
        }
      }while ((id == gestor) && getFinalizar() == false);
      close();  //al salir definitivamente
    }
    catch(SocketTimeoutException e){
      if (getFinalizar() == false)
        avisos.avisaRecepcion((short) 1,id,null);  //NO se ha establecido la conexion
    }
    catch(SocketException e){
      if (getFinalizar() == false)
        avisos.avisaRecepcion((short) 3,id,null);
    }
    catch(IOException e){
      if (getFinalizar() == false)
        avisos.avisaRecepcion((short) 2,id,null);
    }
    catch(NullPointerException e){
      if (getFinalizar() == false)
      {
        avisos = new PaqueteTCPrecepcion();
        avisos.addTCPListener(listener);
        avisos.avisaRecepcion( (short) 2, id, null);
      }
```

```
    }
  }

  private synchronized void esperar(){
   try{
    wait(tiempo_conexion);  //si es 0, esperará indefinidamente
   }
   catch(InterruptedException e){}
  }


  public InetAddress getDireccion(){
   return socket.getInetAddress();
  }

  public synchronized boolean enviar(PDU_TCP pdu){
   if ((socket != null) && (socket.isClosed()==false))
     return tx.enviar(pdu);
   else
     return false;
  }

  private synchronized void setFinalizar(boolean estado){
   this.finalizar = estado;
  }

  private synchronized boolean getFinalizar(){
   return finalizar;
  }

  private synchronized void setReiniciar(boolean estado){
   this.reiniciar = estado;
  }
```

```
private synchronized boolean getReiniciar(){
  return reiniciar;
}

public synchronized void close(){
  try {
    PDU_TCP pdu = new PDU_TCP();
    pdu.setTipo(5,0);

    enviar(pdu);    //nos despedimos

    setFinalizar(true);     //indicamos que lo vamos a cerrar
    rx.removeTCPListener(listener); //asi no seguirá recibiendo los mensajes
    serverSocket.close(); //rx.close() se ejecutara implicitamente
    if (socket != null) //en caso de que no se haya recibido ninguna conexion
      socket.close(); //cerramos el socket

    if (tx != null) //en caso de que no se haya recibido ninguna conexion
      tx.close();
    //rx.close();   lo llama implícitamente al cerrar el socket
    notify(); //si estaba esperando, no se ha enterado de que lo hemos finalizado
  }
  catch (Exception e) {}
}

public synchronized void reiniciar(){ //sólo utilizado por la conexion gestora
  try {
      setReiniciar(true);   //indicamos que lo que pretendemos es reiniciar
      notify(); //si estaba esperando, no se ha enterado de que lo hemos reiniciado
  }
  catch (Exception e) {
  }
}
```

```
  private void prepararAntesReiniciar(){
    try{
      rx.removeTCPListener(listener); //para que no pueda avisar a nadie
      socket.close();
      tx.close();
    }
    catch (IOException e){}
  }
}
```

eventosServidorRed.java

```
import java.util.*;


/**Usado por el servidor para notificar al proceso principal alguna ocurrencia
*/
class EventoServidor extends EventObject { //clase del evento
        short ID;
        short suceso; /** 0:Desconexion por Timeout
                        * 1:Desconexion por error I/O
                        * 2:Desconexion por error en el protocolo (pdu)*/

        public EventoServidor(Object source,short ID,short suceso) {
          super(source);
          this.ID = ID;
          this.suceso = suceso;
          }
       }

interface ServidorListener extends EventListener { //listener del TCP
        public void recibido(EventoServidor evento);
       }
```

```
class ControlEventoServidor {
  Vector listeners;

  public ControlEventoServidor(){
    listeners = new Vector();
  }

  /** Registra un "listener" para avisar cuando se produzca un error*/
  synchronized public void addServidorListener(ServidorListener l) {//recepcion
    if (listeners == null)
      listeners = new Vector();
      listeners.addElement(l);
    }

  /** Elimina un "listener" de la lista*/
    synchronized public void removeServidorListener(ServidorListener l) {
      if (listeners == null)
        listeners = new Vector();
        listeners.removeElement(l);
      }

    /** Indica el evento a todos los "listeners" registrados */
    protected synchronized void avisaServidor(short ID,short suceso) {
      if (listeners != null && !listeners.isEmpty()) {
        EventoServidor evento = new EventoServidor(this,ID,suceso);

        Vector targets;
        synchronized (this) {
          targets = (Vector) listeners.clone();
        }

        Enumeration enum = targets.elements();
        while (enum.hasMoreElements()) {
          ServidorListener l = (ServidorListener) enum.nextElement();
```

```
        l.recibido(evento);
      }
    }
  }
}
```

**********GESTION DE USUARIOS**********

usuarios.java

```
import java.text.DateFormat;
import java.util.*;
import java.io.*;

class entradaUsuario implements Serializable{
  public static final short udp = 0;
  public static final short actuador = 1;
  public static final short admin = 2;
  private String nombre;
  private String password;
  private short tipo;              //indica si TCP, UDP ó Admin
  private Date ultimoAcceso;

  public entradaUsuario(){
    nombre = null;
    password = null;
    ultimoAcceso = null;                    //valor conocido
  }

  public entradaUsuario(entradaUsuario usuario){
    try{
      this.nombre = new String(usuario.nombre);
      this.password = new String(usuario.password);
      this.tipo = usuario.tipo;
      this.ultimoAcceso = usuario.ultimoAcceso;
```

```
  }
  catch(NullPointerException e){}
}

public entradaUsuario(String nombre, String password, short tipo){
  this.nombre = new String(nombre);
  this.password = new String(password);
  this.tipo = tipo;
  setUltimoAcceso();
}


protected void setInfo(String nombre, String password, short tipo){
  setNombre(nombre);
  setPassword(password);
  setTipo(tipo);
  setUltimoAcceso();
}

protected void setPassword(String password){  //fija el password
  this.password = new String(password);
}

protected void setNombre(String nombre){//fija el nombre completo
  this.nombre = new String(nombre);
}

protected void setTipo(short tipo){//fija el tipo
  this.tipo = tipo;
}

protected void setUltimoAcceso(){           //fija la fecha del ultimo acceso
  ultimoAcceso = new Date();
}
```

```
  protected String getPassword(){
      return password;
  }

  protected String getNombre(){
    return nombre;
  }

  protected short getPermisos(){
    return tipo;
  }

  protected Date getUltimoAcceso(){
    return ultimoAcceso;
  }

  protected String getUltimoAccesoString(){
    DateFormat fecha = DateFormat.getInstance();
      return fecha.format(ultimoAcceso);//ultimoAcceso.format(new Date());
    }
}
//-----------------------------TABLA DE USUARIOS-----------------------------

class TablaUsuarios {
  public static final short udp = 0;
  public static final short actuador = 1;
  public static final short admin = 2;
  private Vector listeners;                  //gestor de eventos
  private HashMap tabla;

  public TablaUsuarios(){
    tabla = new HashMap();
  }
```

```
public TablaUsuarios(String nombreArchivo){
  if (!cargarTabla(nombreArchivo))
    tabla = new HashMap();
}

protected synchronized boolean cargarTabla(String nombreArchivo){
  try {
    FileInputStream archivo = new FileInputStream(nombreArchivo);
    ObjectInputStream in = new ObjectInputStream(archivo);
    tabla = (HashMap)in.readObject();
    in.close();
  }
  catch (Exception e) {
    return false;
  }
  return true;
}

protected synchronized boolean guardarTabla(String nombreArchivo){
  try{
    FileOutputStream archivo = new FileOutputStream(nombreArchivo);
    ObjectOutputStream out = new ObjectOutputStream(archivo);
    out.writeObject(tabla);
    out.close();
    }
  catch (Exception e){
    return false;                    //NO se guardó correctamente
  }
  return true;                       //se guardó correctamente
}

protected synchronized int size(){
  return tabla.size();
}
```

```
 protected synchronized boolean anadirUsuario(String login,String nombre,String password,
                          short tipo){
   entradaUsuario usuario;
   if (tabla.containsKey(login) == false){
     usuario = new entradaUsuario(nombre, password, tipo);
     tabla.put(login, usuario);
     avisaTabla((short)0,login);
     return true;
   }
   else
     return false;
 }

 protected synchronized boolean anadirUsuario(String login, entradaUsuario nuevaEntrada){
   if (tabla.containsKey(login) == false){
     tabla.put(login, nuevaEntrada);
     avisaTabla((short)0,login);
     return true;
   }
   else
     return false;
 }


 protected synchronized short getPermiso(String login, char[]password){
   try{
     entradaUsuario entrada = getEntrada(login);

     if (entrada.getPassword().equals(new String(password)))
       return entrada.getPermisos();
     else
```

```
        return (short)(admin + actuador + udp);
      }
      catch(NullPointerException e){
        return (short)(admin + actuador + udp);
      }
    }

    protected synchronized boolean eliminarUsuario(String login){
      if (tabla.remove(login)!= null){
        avisaTabla((short)1,login);
        return true;
      }
      else
        return false;                    //la entrada no existe
    }

  //devuelve el contenido de la entrada solicitara para un cierto login
    protected synchronized entradaUsuario getEntrada(String login){
      return (entradaUsuario)tabla.get(login);
    }

    protected synchronized boolean setLogin(String oldlogin, String newlogin){
       entradaUsuario temp;
       StringBuffer login = new StringBuffer(oldlogin);

       if ((tabla.containsKey(oldlogin) == true) &&
           (tabla.containsKey(newlogin) == false)){
         temp = (entradaUsuario) tabla.get(oldlogin);

         tabla.remove(oldlogin);
         tabla.put(newlogin, temp);

         login.append("#");
         login.append(newlogin);
```

```
      avisaTabla((short)2,login.toString());
      return true;
    }
    else
      return false;
  }

  protected synchronized boolean setPassword(String login, String password){
    entradaUsuario temp;

    if (tabla.containsKey(login) == true){
      temp = (entradaUsuario) tabla.get(login);
      if (!temp.getPassword().equals(password)){
        temp.setPassword(password);
        avisaTabla( (short) 2, login);
        return true;
      }
      else
        return true;
    }
    else
      return false;
  }

  protected synchronized boolean setNombre(String login, String nombre){
    entradaUsuario temp;

    if (tabla.containsKey(login) == true){
      temp = (entradaUsuario) tabla.get(login);
      if (!temp.getNombre().equals(nombre)){
        temp.setNombre(nombre);
        avisaTabla( (short) 2, login);
        return true;
      }
```

```
      else
        return true;
    }
    else
      return false;
  }

  protected synchronized boolean setPermisos(String login, short tipo){
    entradaUsuario temp;

    if (tabla.containsKey(login) == true){
      temp = (entradaUsuario) tabla.get(login);
      if (temp.getPermisos() != tipo){
        temp.setTipo(tipo);
        avisaTabla( (short) 2, login);
        return true;
      }
      else
        return true;
    }
    else
      return false;
  }

  protected synchronized boolean actualizarAcceso(String login){
    entradaUsuario temp;
    if (tabla.containsKey(login) == true){
      temp = (entradaUsuario) tabla.get(login);
      temp.setUltimoAcceso();
      avisaTabla((short)2,login);
      return true;
    }
    else
      return false;
```

```
  }

  protected synchronized Iterator getIterator(){
    return tabla.keySet().iterator();
  }

//*****************************Control del evento*******************************
    synchronized public void addUsuarioListener(UsuarioListener l) {
      if (listeners == null)
        listeners = new Vector();

      listeners.addElement(l);
    }

    /** Elimina un "listener" de la lista*/
    synchronized public void removeUsuarioListener(UsuarioListener l) {
      if (listeners == null)
        listeners = new Vector();

      listeners.removeElement(l);
    }

    /** Indica el evento a todos los "listeners" registrados */
    protected synchronized void avisaTabla(short suceso, String login) {
      // si no hay listeners, no hacemos nada
      if (listeners != null && !listeners.isEmpty()) {
        // creamos el objeto (evento) a enviar
        EventoUsuario evento = new EventoUsuario(this,suceso, login);

        Vector targets;                    // hacemos una copia de la lista,
        synchronized (this) {                //por si acaso alquien
          targets = (Vector) listeners.clone(); //añade ó elimina otro evento
        }
```

```
      Enumeration enum = targets.elements();  //recorremos la lista, llamando
      while (enum.hasMoreElements()) {        //al método recibido de cada uno
        UsuarioListener l = (UsuarioListener) enum.nextElement();
        l.recibido(evento);                   //de los "listener" registrados
      }
     }
   }
}
```

**********CONFIGURACION**********

OpcionesMotores.java

```
import java.io.*;
import java.text.DateFormat;
import java.util.*;
import javax.swing.*;

public class OpcionesMotores { //gestor de la tabla de valores de configuracion de
motores
  private valoresMotores tablaValores;
  private RandomAccessFile[] archivos;
  private int selector; //indica en qué archivo estamos
  private int tamanoLog;

  public OpcionesMotores() {
     selector = 0;

     try {
       for (int i = 0;i < 3;i++){
         File elimina = new File(archivoTemporal(i));
         elimina.delete();
       }

       archivos = new RandomAccessFile[3];
```

```
        archivos[0] = new RandomAccessFile(archivoTemporal(0), "rw");
        archivos[1] = null;
        archivos[2] = null;
      }
      catch (IOException e) {}

    //cargamos el archivo de CONFIGURACION
    if (cargarTabla("motores.inf") == false){
      tablaValores = new valoresMotores(0, 1, 1, (short) 0, 0, 360.0,
                                        (short)127,0, 1, 1, (short) 0, 0,
                                        360.0, (short)127, 1);
      /**si no se puede cargar la tabla, se crea una nueva con los valores por
defecto:
       * paso : 0
       * velocidad : 1
       * angulo del paso = 1
       * flag : 0 (multivueltas)
       * vueltas : 1 (al ser multivueltas, es ignorado)
       * angulo maximo : 360.0 (al ser multivueltas, es ignorado)
       * velocidad máxima : 127
       * tamaño del log: 1 kB
       * */
      JOptionPane.showMessageDialog(null,"No ha sido posible localizar el "
      + "archivo de configuración de los motores. Activadas las opciones " +
      "por defecto");
    }
    else{
      setVueltaActual(1,0); //inicializamos el nº de vueltas
      setVueltaActual(2,0);

      if (obtenerUltimaLinea() != null){ //si el archivo existe, pero está vacío
        tablaValores.paso1 = (int)(getUltimaPosicion((short)1)/
                                   tablaValores.angulo_paso1);
        tablaValores.paso2 = (int)(getUltimaPosicion((short)2)/
```

```
                        tablaValores.angulo_paso2);

    tablaValores.velocidad1 = obtenerVelocidad(getUltimaVelocidad((short)1),
          (int)(360.0/tablaValores.angulo_paso1));

    tablaValores.velocidad2 = obtenerVelocidad(getUltimaVelocidad((short)2),
          (int)(360.0/tablaValores.angulo_paso2));


  }
//inicializamos los archivos temporales del log con el Log anterior
try{
  RandomAccessFile antiguoLog = new RandomAccessFile("Log.txt", "r");
  String cadena;

  cadena = antiguoLog.readLine();

  if ((cadena != null) &&(antiguoLog.length() >=
                    (tablaValores.tamano_log * 1024 / 2))){
    antiguoLog.seek(antiguoLog.length() - (tablaValores.tamano_log * 1024));

    cadena = antiguoLog.readLine();
    if (cadena.indexOf("/") < 0) {//no hay barra de fecha
      cadena = antiguoLog.readLine();
    }

    while ((cadena != null) && (antiguoLog.getFilePointer() <
                        (tablaValores.tamano_log * 1024 / 2))){
      archivos[selector].writeBytes(cadena + "\n");
      cadena = antiguoLog.readLine();
    }

      archivos[selector].close();
      selector = siguienteArchivo(selector);
```

```
      }

      archivos[selector] = new RandomAccessFile(archivoTemporal(selector),"rw");
      //creamos el segundo fichero, si es que el nuevo log es >= que el anterior,
//      o el primero, si es menor

      while (cadena != null){
       archivos[selector].writeBytes(cadena + "\n");
       cadena = antiguoLog.readLine();
      }
      antiguoLog.close(); //cerramos el log
    }
    catch (IOException e){}

    tamanoLog = tablaValores.tamano_log;
    }
  }

  public synchronized boolean guardarTabla() {
    try {
      String nombreArchivo = new String("motores.inf");
      FileOutputStream archivo = new FileOutputStream(nombreArchivo);
      ObjectOutputStream out = new ObjectOutputStream(archivo);
      out.writeObject(tablaValores);
      out.close();
      return true; //se guardó correctamente
    }
    catch (Exception e) {
      return false; //NO se guardó correctamente
    }
  }

//------------------GET
```

```
synchronized int getPaso(int ID) {
  if (ID == 1)
    return tablaValores.paso1;
  else
    return tablaValores.paso2;
}

synchronized double getAnguloActual(int ID) {
  if (ID == 1)
    return (double) (tablaValores.paso1 * 1.0 * tablaValores.angulo_paso1);
  else
    return (double) (tablaValores.paso2 * 1.0 * tablaValores.angulo_paso2);
}

synchronized int getVelocidad(int ID) {
  if (ID == 1)
    return tablaValores.velocidad1;
  else
    return tablaValores.velocidad2;
}

synchronized double getAnguloPaso(int ID) {
  if (ID == 1)
    return tablaValores.angulo_paso1;
  else
    return tablaValores.angulo_paso2;
}

synchronized int getVueltas(int ID) {
  if (ID == 1)
    return tablaValores.vueltas1;
  else
    return tablaValores.vueltas2;
}
```

```
synchronized int getVueltaActual(int ID) {
  if (ID == 1)
    return tablaValores.vueltaActual1;
  else
    return tablaValores.vueltaActual2;
}


synchronized double getAnguloMaximo(int ID) {
  if (ID == 1)
    return tablaValores.angulo_maximo1;
  else
    return tablaValores.angulo_maximo2;
}

synchronized short getVelocidadMaxima(int ID){
  if (ID == 1)
    return tablaValores.velocidadMaxima1;
  else
    return tablaValores.velocidadMaxima2;
}

synchronized short getFlag(int ID) {
  if (ID == 1)
    return tablaValores.flag1;
  else
    return tablaValores.flag2;
}

synchronized int getTamanoLog() {
  return tablaValores.tamano_log;
}
```

```
//----------------------SET

  synchronized void setPaso(int ID, int paso) {
    if (ID == 1) {
      if (paso < 0) {
        tablaValores.paso1 = (int)((360/tablaValores.angulo_paso1) + paso);
        if (getFlag(1) == 1) //depende del número de vueltas
          this.decrementaVuelta(1);
      }
      else {
        if (paso >= (360 / tablaValores.angulo_paso1)) { //supera el máximo de pasos
          tablaValores.paso1 = (int) (paso - (360 / tablaValores.angulo_paso1));
          if (getFlag(1) == 1) //depende del número de vueltas
            this.incrementaVuelta(1);
        }
        else
          tablaValores.paso1 = paso;
      }
    }
    else {
      if (paso < 0) {
        tablaValores.paso2 = (int)((360/tablaValores.angulo_paso2) + paso);
        if (getFlag(2) == 1) //depende del número de vueltas
          this.decrementaVuelta(2);
      }
      else {
        if (paso >= (360 / tablaValores.angulo_paso2)) { //supera el máximo de pasos
          tablaValores.paso2 = (int) (paso - (360 / tablaValores.angulo_paso2));
          if (getFlag(2) == 1) //depende del número de vueltas
            this.incrementaVuelta(2);
        }
        else
          tablaValores.paso2 = paso;
      }
```

```
  }
 }

 synchronized void setVelocidad(int ID, int velocidad) {
  if (ID == 1)
    tablaValores.velocidad1 = velocidad;
  else
    tablaValores.velocidad2 = velocidad;
 }

 synchronized void setAnguloPaso(int ID, double angulo_paso) {
  if (ID == 1)
    tablaValores.angulo_paso1 = angulo_paso;
  else
    tablaValores.angulo_paso2 = angulo_paso;
 }

 synchronized void setVueltas(int ID, int vueltas) {
  if (ID == 1)
    tablaValores.vueltas1 = vueltas;
    if (tablaValores.vueltaActual1 > vueltas)
      tablaValores.vueltaActual1 = vueltas;
  else
    tablaValores.vueltas2 = vueltas;
    if (tablaValores.vueltaActual2 > vueltas)
      tablaValores.vueltaActual2 = vueltas;
 }

 synchronized void setVueltaActual(int ID, int vueltas) {
  if (ID == 1)
    tablaValores.vueltaActual1 = vueltas;
  else
    tablaValores.vueltaActual2 = vueltas;
 }
```

```
synchronized void setVelocidadMaxima(int ID, short velocidad){
  if (ID == 1)
    tablaValores.velocidadMaxima1 = velocidad;
  else
    tablaValores.velocidadMaxima2 = velocidad;
}

synchronized void incrementaVuelta(int ID){
  if (ID == 1)
    tablaValores.vueltaActual1++;
  else
    tablaValores.vueltaActual2++;
}

synchronized void decrementaVuelta(int ID){
  if (ID == 1)
    tablaValores.vueltaActual1--;
  else
    tablaValores.vueltaActual2--;
}



synchronized void setAnguloMaximo(int ID, double angulo_maximo) {
  if (ID == 1)
    tablaValores.angulo_maximo1 = angulo_maximo;
  else
    tablaValores.angulo_maximo2 = angulo_maximo;
}

synchronized void setFlag(int ID, short flag) {
  if (ID == 1)
    tablaValores.flag1 = flag;
  else
    tablaValores.flag2 = flag;
```

```
  }

  synchronized void setTamanoLog(int tamano) {
    tablaValores.tamano_log = tamano;
  }

  private synchronized boolean cargarTabla(String nombreArchivo) {
    try {
      FileInputStream archivo = new FileInputStream(nombreArchivo);
      ObjectInputStream in = new ObjectInputStream(archivo);
      tablaValores = (valoresMotores) in.readObject();
      in.close();
      return true; //se puedo cargar
    }
    catch (Exception e) {
      //en caso de error, se mantiene la tabla anterior y se avisa del error
      return false;
    }
  }

//----------------------------FUNCIONES DE LOG-----------------------
  protected synchronized void movido(double angulo_final1, int velocidad1,
                                     double angulo_final2, int velocidad2,
                                     String usuario) {
    String cadena;
    DateFormat fecha = DateFormat.getInstance();

    try {
      cadena = new String(fecha.format(new Date()) +
                          "\t\t" +
                          "Posicion Motor 1 : " +
                          cadenaNumero(angulo_final1,"º") +
                           "Velocidad Motor 1 : " +
                          cadenaNumero(cuentaRevoluciones(velocidad1,
```

```
                    (int)(360/tablaValores.angulo_paso1)),"rpm") +
                    "Posicion Motor 2 : " +
                    cadenaNumero(angulo_final2,"º") +
                    "Velocidad Motor 2 : " +
                    cadenaNumero(cuentaRevoluciones(velocidad2,
                      (int)(360/tablaValores.angulo_paso2)),"rpm") +
                    " Usuario : " + usuario + "\n");

    if ( (archivos[selector].getFilePointer() + cadena.length()) >
        (tamanoLog * 1024 / 2)) { //es mayor que la mitad del maximo
      if (archivos[siguienteArchivo(selector)] != null) {
        archivos[siguienteArchivo(selector)].close();

        File elimina = new File(archivoTemporal(siguienteArchivo(selector)));
        elimina.delete();
      }
      selector = siguienteArchivo(selector);
      archivos[selector] = new RandomAccessFile(archivoTemporal(selector),
                                        "rw");
    }

    archivos[selector].writeBytes(cadena);
  }
  catch (IOException e) {}
}

protected synchronized double getUltimaVelocidad(short ID) {
  String cadena;
  int pos;

  try {
    if ((cadena = obtenerUltimaLinea()) != null){
     if (ID == 1)
       pos = cadena.lastIndexOf("Velocidad Motor 1 : ") + 20;
```

```
      else
        pos = cadena.lastIndexOf("Velocidad Motor 2 : ") + 20;

      return (new Double(cadena.substring(pos, cadena.indexOf("rpm", pos)))).
          doubleValue();
    }
    else
      return 0; //es la velocidad mínima
  }

  catch (Exception e) {
    return 0;
  }
 }

 protected synchronized double getUltimaPosicion(short ID) {
   String cadena;
   int pos;

   try {
     if ((cadena = obtenerUltimaLinea()) != null){
       if (ID == 1)
         pos = cadena.lastIndexOf("Posicion Motor 1 : ") + 19;
       else
         pos = cadena.lastIndexOf("Posicion Motor 2 : ") + 19;

      return (new Double(cadena.substring(pos, cadena.indexOf("º", pos)))).
         doubleValue();
      }
      else
        return 0;

   }
   catch (Exception e) {
```

```
    return 0;
  }
}

protected synchronized boolean crearLogFinal() {
  try {
    File fregistro = new File("Log.txt");
    fregistro.delete();

    RandomAccessFile registro = new RandomAccessFile(fregistro, "rw");
    String cadena;
    int selectorTemp;

    if (archivos[selectorTemp = siguienteArchivo(selector)] != null) {
      archivos[selectorTemp] = new RandomAccessFile(
          archivoTemporal(selectorTemp),"r");

      archivos[selectorTemp].seek(archivos[selector].getFilePointer());
      cadena = archivos[selectorTemp].readLine();
      if (cadena.indexOf("/") < 0) {
        cadena = archivos[selectorTemp].readLine();
      }

      cadena = archivos[selectorTemp].readLine();
      while (cadena != null) {
        registro.writeBytes(cadena + "\r\n");
        cadena = archivos[selectorTemp].readLine();
      }
      archivos[selectorTemp].close();
    }

    if (archivos[selectorTemp = previoArchivo(selector)] != null) {
      archivos[selectorTemp] = new RandomAccessFile(
          archivoTemporal(selectorTemp),"r");
```

```
        archivos[selectorTemp].seek(0);
        cadena = archivos[selectorTemp].readLine();
        while (cadena != null) {
          registro.writeBytes(cadena + "\r\n");
          cadena = archivos[selectorTemp].readLine();
        }
        archivos[selectorTemp].close();
      }

      if (archivos[selector] != null) {
        long temp = archivos[selector].getFilePointer();

        archivos[selector].seek(0);
        cadena = archivos[selector].readLine();
        while (cadena != null) {
          registro.writeBytes(cadena + "\r\n");
          cadena = archivos[selector].readLine();
        }
        archivos[selector].seek(temp);
      }
      registro.close();
      return true;
    }
    catch (IOException e) {
      return false;
    }
  }

  public synchronized void close() {
    try {
      File eliminar;
      crearLogFinal();

      archivos[selector].close();
```

```
    for (int i = 0; i < archivos.length; i++) {
      eliminar = new File(archivoTemporal(i));
      eliminar.delete();
    }
  }
  catch (IOException e) {}
  catch (NullPointerException e) {}
}

private String archivoTemporal(int ID) {
  switch (ID) {
    case 0:
      return "tmp0.tmp";
    case 1:
      return "tmp1.tmp";
    default:
      return "tmp2.tmp";
  }
}

private int siguienteArchivo(int ID) {
  switch (ID) {
    case 0:
      return 1;
    case 1:
      return 2;
    default:
      return 0;
  }
}

private int previoArchivo(int ID) {
  switch (ID) {
    case 0:
```

```
      return 2;
    case 1:
      return 0;
    default:
      return 1;
  }
}

 private String cadenaNumero(double numero, String unidad) {
  String cadena;

  cadena = (new Double(numero)).toString();
  if (cadena.length() >= cadena.indexOf('.') + 2) //1 o mas decimales
    cadena = cadena.substring(0, cadena.indexOf('.') + 2);  //cogemos 1 decimal

  return (cadena + unidad + anadeEspacios(8 - cadena.length()));
}

 private String cadenaNumero(String numero, String unidad) {
  return (numero + unidad +  anadeEspacios(8 - numero.length()));
}


 private String anadeEspacios(int espacios) {
  String cadena = new String("        ");

  if (espacios > 0)
    for (int i = 0; i < espacios; i++)
      cadena = cadena + " ";
  return cadena;
}

 private String cuentaRevoluciones(int velocidad,int n_pasos){
  if (velocidad != 0){
```

```
    Double rev = new Double(60.0 / (((0xFF - velocidad+1) * 256 * 0x40 *
                                4.0/6000000.0) * (2.0 * n_pasos)));
    String cadena = new String(rev.toString());
    if (cadena.length() >= cadena.indexOf('.') + 2) //1 o mas decimales
      return cadena.substring(0, cadena.indexOf('.') + 2); //sin los "rpm"
    else
      return cadena;
  }
else
  return new String("0");
}

private int obtenerVelocidad(double revoluciones, int n_pasos){
  int temp;
 temp = (int)(1+0xFF-(60.0 /
              ((revoluciones *0x40*4.0/6000000.0) * (2.0 * n_pasos))));
if (temp > 255)//error
  return 255; //valor mínimo
if (temp <= 0) //error
  return 1; //valor mínimo

return temp;
}

private String obtenerUltimaLinea(){
  try{
    RandomAccessFile archivo = new RandomAccessFile("Log.txt", "r");
    String cadena;

    if (archivo.length() > 300) {//valor mayor que un registro, pero menor que dos
      archivo.seek(archivo.length() - 300);
      cadena = archivo.readLine();
      if (cadena.indexOf("/") < 0) {
        cadena = archivo.readLine();
```

```
        }
      }
      else
        if (archivo.length() != 0)
          cadena = archivo.readLine();
        else
          return null;

      archivo.close();
      return cadena;
    }
    catch(IOException e){
      return null;
    }
    catch(NullPointerException e){
      return null;
    }
  }
}

//tabla con los valores
class valoresMotores implements Serializable{
    public int paso1;       //paso actual del motor 1
    public int paso2;       //paso actual del motor 2
    public int velocidad1;   //vel. actual del motor 1
    public int velocidad2;   //vel. actual del motor 1

    public double angulo_paso1;   //angulo de cada paso del motor 1
    public short flag1;  //0: Multivueltas, 1:nºde vueltas, 2: angulo maximo
    public int vueltas1;          //numero de vueltas
    public double angulo_maximo1; //angulo maximo de giro
    transient public int vueltaActual1;
    public short velocidadMaxima1;
```

```
public double angulo_paso2;   //angulo de cada paso del motor 2
public short flag2;  //0: Multivueltas, 1:nºde vueltas, 2: angulo maximo
public int vueltas2;          //numero de vueltas
public double angulo_maximo2; //angulo maximo de giro
transient public int vueltaActual2;
public short velocidadMaxima2;

public int tamano_log;

public valoresMotores(
  int paso1,int velocidad1,double angulo_paso1, short flag1, int vueltas1,
  double angulo_maximo1,  short velocidadMaxima1,  int paso2,
  int velocidad2, double angulo_paso2, short flag2, int vueltas2,
  double angulo_maximo2, short velocidadMaxima2, int tamano_log){

  this.paso1 = paso1;
  this.paso2 = paso2;
  this.velocidad1 = velocidad1;
  this.velocidad2 = velocidad2;

  this.angulo_paso1 = angulo_paso1;
  this.angulo_paso2 = angulo_paso2;
  this.flag1 = flag1;
  this.flag2 = flag2;
  this.vueltas1 = vueltas1;
  this.vueltas2 = vueltas2;
  this.angulo_maximo1 = angulo_maximo1;
  this.angulo_maximo2 = angulo_maximo2;

  this.vueltaActual1 = 0;   //siempre se toma como vuelta 0
  this.vueltaActual2 = 0;

  this.velocidadMaxima1 = velocidadMaxima1;
  this.velocidadMaxima2 = velocidadMaxima2;
```

```
      this.tamano_log = tamano_log;
    }
}
```

OpcionesRed.java

```
import java.io.*;

public class OpcionesRed {
  valoresRed tablaValores;

  public OpcionesRed(){
    if (cargarTabla("red.inf") == false) //no se pudo cargar
      tablaValores = new valoresRed(5,8000,5000,5000,60000,30,200);
       /** Valores por defecto:
       * maximo nº de conexiones = 5
       * puertoGestion = 8000
       * timeout gestion = 5sg
       * timeout actuador = 5sg
       * timeout espectador = 60sg
       * tiempo de conexion de gestion = 30sg  (viene dado en segundos)
       * tiempo de conexion actuador = 200 sg  (viene dado en segundos)
       */
      guardarTabla();
    }

  public synchronized boolean guardarTabla(){
     try{
       String nombreArchivo = new String("red.inf");
       FileOutputStream archivo = new FileOutputStream(nombreArchivo);
       ObjectOutputStream out = new ObjectOutputStream(archivo);
       out.writeObject(tablaValores);
       out.close();
       return true;                          //se guardó correctamente
```

```
     }
    catch (Exception e){
      return false;
    }
   }

public synchronized boolean copiaParaApplet(){
  try{
    String temp = getRuta();
    tablaValores.ruta = null;

    FileOutputStream archivo = new FileOutputStream(temp+"\\"+"red.inf");
    ObjectOutputStream out = new ObjectOutputStream(archivo);
    out.writeObject(tablaValores);
    out.close();

    tablaValores.ruta = temp;
    return true;
    }
  catch (Exception e){
    return false;
  }

}

private synchronized boolean cargarTabla(String nombreArchivo){
    try {
      FileInputStream archivo = new FileInputStream(nombreArchivo);
      ObjectInputStream in = new ObjectInputStream(archivo);
      tablaValores = (valoresRed)in.readObject();
      in.close();
      return true;
    }
    catch (Exception e) {
```

```
        return false;
      }
    }

//--------SET

    synchronized void setMaximoConexiones(int n_conexiones){
      tablaValores.maximoConexiones = n_conexiones;
    }

    synchronized void setPuertoGestion(int puerto){
      tablaValores.puertoGestion = puerto;
    }

    synchronized void setTimeoutGestion(int timeout){
      tablaValores.timeoutGestion = timeout;
    }

    synchronized void setTimeoutActuador(int timeout){
      tablaValores.timeoutActuador = timeout;
    }

    synchronized void setTimeoutEspectador(int timeout){
      tablaValores.timeoutEspectador = timeout;
    }

    synchronized void setTiempoConexionGestion(int tiempo){
      tablaValores.tiempoConexionGestion = tiempo * 1000;
    }

    synchronized void setTiempoConexionActuador(int tiempo){
      tablaValores.tiempoConexionActuador = tiempo * 1000;
    }
```

```
    synchronized void setRuta(String ruta){
      tablaValores.ruta = ruta;
    }
//---------GET
    synchronized int getMaximoConexiones(){
      return tablaValores.maximoConexiones;
    }

    synchronized int getPuertoGestion(){
      return tablaValores.puertoGestion;
    }

    synchronized int getTimeoutGestion(){
      return tablaValores.timeoutGestion;
    }

    synchronized int getTimeoutActuador(){
      return tablaValores.timeoutActuador;
    }

    synchronized int getTimeoutEspectador(){
       return tablaValores.timeoutActuador;
    }

    synchronized int getTiempoConexionGestionLong(){
      return (int)(tablaValores.tiempoConexionGestion);
    }

    synchronized int getTiempoConexionActuadorLong(){
      return (int)(tablaValores.tiempoConexionActuador);
    }

    synchronized int getTiempoConexionGestion(){
      return (int)(tablaValores.tiempoConexionGestion / 1000);
```

```
    }

    synchronized int getTiempoConexionActuador(){
      return (int)(tablaValores.tiempoConexionActuador / 1000);
    }

    synchronized String getRuta(){
      return tablaValores.ruta;
    }
}

class valoresRed implements Serializable{
  int maximoConexiones;

  int puertoGestion;

  int timeoutGestion;
  int timeoutActuador;
  int timeoutEspectador;

  long tiempoConexionGestion;
  long tiempoConexionActuador;

  String ruta; //define la ruta del applet

  public valoresRed(int maximoConexiones, int puertoGestion, int timeoutGestion,
                    int timeoutActuador, int timeoutEspectador,
                    long tiempoConexionGestion, long tiempoConexionActuador){

    this.maximoConexiones =maximoConexiones;
    this.puertoGestion = puertoGestion;
    this.timeoutGestion = timeoutGestion;
    this.timeoutActuador = timeoutActuador;
    this.timeoutEspectador = timeoutEspectador;
```

```
    this.tiempoConexionGestion = tiempoConexionGestion;
    this.tiempoConexionActuador = tiempoConexionActuador;

    this.ruta = new String("");
  }
}
```

HTMLFilter.java

```
import java.io.File;
import javax.swing.filechooser.*;


//clase utilizada para el cuadro que busca el archivo principal del applet
public class HTMLFilter extends FileFilter {
  public boolean accept(File f) {
    if (f.isDirectory())
    { return true; }
    String extension = getExtension(f);
    if (extension != null) {
      if (extension.equals("html"))
        return true;
      else
        return false;
    }
    return false;
  }

  public String getDescription() { return "Archivos HTML";}

  public static String getExtension(File f) {
    String ext = null;
    String s = f.getName();
```

```
    int i = s.lastIndexOf('.');
    if (i > 0 && i < s.length() - 1)
      { ext = s.substring(i+1).toLowerCase();
    }
    return ext;
  }
 }
```

**********GESTION USB**********

pipeC.java

```
class PipeC {

  char datoRecibido[] = new char[10];

  public native boolean escribir(char[] bufferOut);
  public native boolean abrir();
  public native void cerrar();

  static {
    System.loadLibrary("pipeC");
  }
}
```

pipeC.h ***(CODIGO NATIVO)***

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class Pipe */

#ifndef _Included_Pipe
#define _Included_Pipe
#ifdef __cplusplus
extern "C" {
#endif
```

```
/*
 * Class:     Pipe
 * Method:    escribir
 * Signature: ([C)Z
 */
JNIEXPORT jboolean JNICALL Java_Pipe_escribir
  (JNIEnv *, jobject, jcharArray);

#ifdef __cplusplus
}
#endif
#endif
```

pipeC.c ***(CODIGO NATIVO)***

```
#include "stdafx.h"

#include <jni.h>
#include "PipeC.h"
#include <stdio.h>

#define LongitudTrama		10	//longitud de la trama enviada
HANDLE proceso;

bool iniciaPipe(HANDLE *, HANDLE *, OVERLAPPED *);//declaracion de las
funciones
bool leePipe(HANDLE , char *, DWORD, HANDLE, LPOVERLAPPED, long) ;
long calculaTiempo(char, char); //calcula el tiempo a esperar

JNIEXPORT jboolean JNICALL Java_PipeC_escribir(JNIEnv *env, jobject obj,
jcharArray arr)
{
	jfieldID fid;
	jclass cls = (env)->GetObjectClass(obj); //identificador de la clase
	jcharArray devuelto;
```

```
jchar datoJava[LongitudTrama];  //vector temporal para almacenar el valor
                                //que devolveremos
jboolean conseguido;

char bufferRead[LongitudTrama];
HANDLE hPipe,hEvento;
DWORD bytesEscritos;
OVERLAPPED hOverlapped;
int Result=0;

if(iniciaPipe(&hPipe,&hEvento,&hOverlapped) == false)
{
        datoJava[0]=3;//error al iniciar el pipe
        conseguido = false;
}

else
{
        jsize len = (env)->GetArrayLength(arr);
        jchar *vector = (env)->GetCharArrayElements(arr, 0); //obtenemos el
                //vector

        char envia[LongitudTrama];
        for(int i=0;i<LongitudTrama;i=i++)
                envia[i] = vector[i];

        int tiempo = calculaTiempo(envia[LongitudTrama-2],
                envia[LongitudTrama-1]);

        WriteFile(hPipe, //enviamos los datos al pipe
                        envia,
                        len,
                        &bytesEscritos,
                        (LPOVERLAPPED) &hOverlapped);
```

```
            if (GetLastError() == ERROR_IO_PENDING)
            {
              Result = WaitForSingleObject (hEvento, tiempo);
            }

            switch (Result)
            {
            case WAIT_OBJECT_0:
                    {
                    ResetEvent(hEvento);
                    if (leePipe(hPipe, bufferRead, LongitudTrama, hEvento,
&hOverlapped, tiempo)==true)
                            {
                            if (bufferRead[0] != 5){
                                    for(int i=0;i<LongitudTrama;i++) //copiamos el
                                            //valor recibido a una variable jchar*
                                            datoJava[i]=bufferRead[i];
                                            conseguido = true;


                                            if (bufferRead[0] == 1){
            //comprobamos que son los mismos datos que los enviados
                                            for(int j=0; j<LongitudTrama, bufferRead[j]
                                                    == envia[j];j++){}

                                            if (j != LongitudTrama)
                                                    datoJava[0] = 2;
                            }
                    }
                    else
                            {
                            conseguido = false;
                            datoJava[0] = bufferRead[0];
```

```
                        }
                }
                else
                {
                        datoJava[0] = 4;      //Error en el pipe
                        conseguido = false;
                }
                break;
                }
        case WAIT_TIMEOUT:
                {
                //el tiempo ha expirado
                CloseHandle(hPipe);

                //Indicamos error en la aplicacion USB (no ha devuelto el dato
//en el tiempo acordado)
                datoJava[0] = 4;
                conseguido = false;
                break;
        default:
                datoJava[0] = 4;
                conseguido = false;
                break;
                }
        }

        (env)->ReleaseCharArrayElements(arr, vector, 0);
    }
    //Preparamos el valor a devolver:
    fid = (env)->GetFieldID(cls, "datoRecibido", "[C");
    devuelto = (env)->NewCharArray(LongitudTrama);
    (env)->SetCharArrayRegion(devuelto,0,LongitudTrama,datoJava);
    (env)->SetObjectField(obj, fid, devuelto);
```

```
        CloseHandle(hPipe);

        return conseguido;
}

JNIEXPORT jboolean JNICALL Java_PipeC_abrir(JNIEnv *env, jobject obj)
{
   STARTUPINFO si;
   PROCESS_INFORMATION pi;

   ZeroMemory( &si, sizeof(si) );
   si.cb = sizeof(si);
   ZeroMemory( &pi, sizeof(pi) );

   if( !CreateProcess( NULL,
      TEXT("pipe.exe"),
      NULL,
      NULL,
      FALSE,
      0,
      NULL,
      NULL,
      &si,
      &pi))
        return false;

   proceso = pi.hProcess;
   CloseHandle( pi.hThread ); //liberamos el recurso
   return true;
}

JNIEXPORT void JNICALL Java_PipeC_cerrar(JNIEnv *env, jobject obj){

        TerminateProcess(proceso,0);
```

```
        CloseHandle(proceso);
        return;
}

bool leePipe(HANDLE hPipe, char *buffer, DWORD bufferSize, HANDLE hEvento,
LPOVERLAPPED hOverlapped, long tiempo)
{
        DWORD bytesEscritos;
        int Result;

        ReadFile(
    hPipe,
    buffer,
    bufferSize,
    &bytesEscritos,
    (LPOVERLAPPED) hOverlapped);

        Result = WaitForSingleObject
                (hEvento, tiempo);

        switch (Result)
        {
        case WAIT_OBJECT_0:
                {

                        break;
                }
        case WAIT_TIMEOUT:
                {
                Result = CancelIo(hPipe);

                CloseHandle(hPipe);
                return false;
                break;
```

```
        default:
                return false;
                break;
                }
        }

        ResetEvent(hEvento);
        return true;
}

bool iniciaPipe(HANDLE *hPipe, HANDLE *hEvento, OVERLAPPED
*hOverlapped){
        LPTSTR lpszPipename = "\\\\.\\pipe\\pfc_usb_pipe";

        *hPipe=CreateFile
                (lpszPipename,
                GENERIC_READ|GENERIC_WRITE, //lectura y escritura
                0,
                (LPSECURITY_ATTRIBUTES)NULL,
                OPEN_EXISTING,
                FILE_FLAG_OVERLAPPED,
                NULL);


        if (*hPipe == INVALID_HANDLE_VALUE)
        {
                if (GetLastError() == ERROR_PIPE_BUSY)
                {
        // todas las instancias están ocupadas, esperamos
                        if (WaitNamedPipe(lpszPipename, 20000) == 0)
                        {
                                return false;  //no se pudo iniciar
                        }
                }
```

```
                else
                {
                        return false;
                }
        }

        *hEvento = CreateEvent
                (NULL,
                TRUE,
                TRUE,
                "");

        //establecemos los miembros de la estructura Overlapped
        hOverlapped->hEvent = *hEvento;
        hOverlapped->Offset = 0;
        hOverlapped->OffsetHigh = 0;
return true;
}

long calculaTiempo(char alto, char bajo){
  return (((alto * 0x100) + bajo + 5)*1000); //damos 5 segundos de margen
}
```

ToUSB.java

```
import java.util.*;

public class ToUSB extends Thread
   implements PanelMovimientoListener,MoverMotorRedListener{
 private PipeC pipe;
 private OpcionesMotores tablaMotores;
 private char mensaje[]; //lo que enviamos al USB

 Vector peticiones;
```

```
boolean procesando; //indica si está procesando una petición
Vector listeners;


public ToUSB(OpcionesMotores opcionesMotores){
  this.pipe = new PipeC();
  this.tablaMotores = opcionesMotores;
  this.mensaje = new char[10];

  this.peticiones = new Vector();
  this.listeners = new Vector();
  procesando = false;

  pipe.abrir();

  start();
}

private synchronized void anadePeticion(char[] mensaje){
   if (peticiones.isEmpty() == false)
     peticiones.add(mensaje);
   else{
     peticiones.add(mensaje);
     if (getProcesando() == false)
       notify();
   }
}

private synchronized void vaciaListaPeticiones(){
  peticiones.clear();
}

private synchronized void esperar(){
  try{
```

```
    if (peticiones.isEmpty() == true){
      setProcesando(false);
      wait();
    }
  }
  catch (InterruptedException e) {}
}

private synchronized Object getPeticion(){
  try{
    if (peticiones.isEmpty())
      return null;
    else{
      setProcesando(true);
      Object temp = peticiones.elementAt(0);
      peticiones.remove(0);
      return temp;
    }
  }
  catch(ArrayIndexOutOfBoundsException e){
    return null;
  }
}

private synchronized void setProcesando(boolean estado){
  procesando = estado;
}

private synchronized boolean getProcesando(){
  return procesando;
}

public void run(){
  char[] info;
```

```
char tipo = 1;

try{
  while (tipo == 1) { //datos
    esperar();

    info = (char[])getPeticion();
    tipo = info[0];

    if (pipe.escribir(info) == true){
      recibido(info);
    }

    else{ //problemas para acceder al pipe
      if (pipe.datoRecibido[0] != 5){
        pipe.cerrar();
        if (pipe.abrir() == false)
          avisaMovido(3);

        if (pipe.escribir(info) == true) {
          recibido(info);
        }
        else
          avisaMovido(4);
      }
      else{ //dispositivo desconectado
        avisaMovido(pipe.datoRecibido[0]);
        pipe.datoRecibido[0] = 1;
      }
    }
  }
}
catch (NullPointerException e){
}
```

```
 }

 public void close(){
  vaciaListaPeticiones();
  mensaje[0] = 0;
  anadePeticion(mensaje);
  pipe.cerrar();  //seguro
 }

 private void recibido(char[] info){
  int paso1 = conformaPaso(1, info);
  int paso2 = conformaPaso(2, info);

  tablaMotores.setPaso(1, tablaMotores.getPaso(1) + paso1);
  tablaMotores.setPaso(2, tablaMotores.getPaso(2) + paso2);

  tablaMotores.setVelocidad(1, info[2]);
  tablaMotores.setVelocidad(2, info[5]);

  avisaMovido(1);
 }

 public void moverMotor(PanelMovimientoEvent ev) {
  mensaje[0] = 1;

  switch (ev.ID){
    case 0:
      if (ev.paso1 !=  tablaMotores.getAnguloPaso(1))
        setInfoMotor((short)1,ev.paso1,(short)ev.velocidad1);
      else
        setNuloMotor((short)1);

      if (ev.paso2 != tablaMotores.getAnguloPaso(2))
        setInfoMotor((short)2,ev.paso2,(short)ev.velocidad2);
```

```
      else
        setNuloMotor((short)2);
      break;
    case 1: //SOLO motor 1
      setInfoMotor((short)1,ev.paso1,(short)ev.velocidad1);
      setNuloMotor((short)2);
      break;
    case 2: //SOLO motor 2
      setNuloMotor((short)1);
      setInfoMotor((short)2,ev.paso2,(short)ev.velocidad2);
      break;
  }
  calculoTiempo(mensaje);//añadimos cuánto se debe esperar
  anadePeticion(mensaje);
}

public void moverMotorRed(MoverMotorRedEvent ev){
  mensaje[0] = 1;

  switch (ev.ID){
    case 0:
      if (ev.paso1 != 0)
        setInfoMotor((short)1,ev.paso1,(short)ev.velocidad1);
      else
        setNuloMotor((short)1);

      if (ev.paso2 != 0)
        setInfoMotor((short)2,ev.paso2,(short)ev.velocidad2);
      else
        setNuloMotor((short)2);
      break;
    case 1: //SOLO motor 1
      setInfoMotor((short)1,ev.paso1,(short)ev.velocidad1);
      setNuloMotor((short)2);
```

```
        break;
      case 2: //SOLO motor 2
        setNuloMotor((short)1);
        setInfoMotor((short)2,ev.paso2,(short)ev.velocidad2);
        break;
    }

    calculoTiempo(mensaje);//añadimos cuánto se debe esperar
    anadePeticion(mensaje);
  }

//--------------------------TEMPORIZACION----------------------
 private void calculoTiempo(char mensaje[]){
   double tiempo1;
   double tiempo2;

   tiempo1 = duracion((mensaje[3] * 0x100) + mensaje[4],mensaje[2]);
   tiempo2 = duracion((mensaje[6] * 0x100) + mensaje[7],mensaje[5]);

   if (tiempo1 >= tiempo2){  //cogemos el que vaya a tardar más

     mensaje[8] = (char) ((Math.round(tiempo1) & 0xFF00)/ 0x100);
     mensaje[9] = (char) (Math.round(tiempo1) & 0x00FF);
   }
   else
     mensaje[8] = (char) ((Math.round(tiempo2) & 0xFF00)/ 0x100);
     mensaje[9] = (char) (Math.round(tiempo2) & 0x00FF);
 }

 private double duracion(int pasos, int velocidad){
   return (((0xFF - velocidad+1)*256*0x40*4.0/6000000.0)*(2.0*pasos));
 }
//----------------CONFIGURACION DE LOS MOTORES (Y SUS MENSAJES)-----
```

```
 private synchronized void setInfoMotor(short IDMotor,double anguloFinal,
                                        short velocidad){
  int pasos = calculaPasos(IDMotor,anguloFinal);

  if (IDMotor == 1){ //motor 1
    if (pasos > 0)
      mensaje[1] = (char)(mensaje[1] | 0x10);//derecha  (1)
    else{
      mensaje[1] = (char) (mensaje[1] & 0x01); //izquierda (0)
      pasos = pasos * (-1);
    }

    mensaje[2] = (char)velocidad;
    mensaje[3] = (char) ((pasos & 0xFF00) / 0x100);
    mensaje[4] = (char) (pasos & 0x00FF);
  }
  else{//motor 2
    if (pasos > 0)
      mensaje[1] = (char)(mensaje[1] | 0x01);//derecha (1)
    else{
      mensaje[1] = (char)(mensaje[1] & 0x10);//izquierda (0)
      pasos = pasos * (-1);
    }

    mensaje[5] = (char)velocidad;
    mensaje[6] = (char) ((pasos & 0xFF00) / 0x100);
    mensaje[7] = (char) (pasos & 0x00FF);
  }
 }

  private synchronized void setNuloMotor(short IDMotor){
    if (IDMotor == 1){ //motor 1

      mensaje[2] = 0;
```

```
        mensaje[3] = 0;
        mensaje[4] = 0;
      }
      else{//motor 2
       mensaje[5] = 0;
       mensaje[6] = 0;
       mensaje[7] = 0;
      }
  }

  //dado los dos caracteres devueltos, conforma el número
  private int conformaPaso(int ID, char []mensaje){
    int paso;
    if (ID ==1){
      paso = ( (mensaje[3] * 0x100) + mensaje[4]);
      if ( (mensaje[1] & 0xF0) == 0)
        paso = paso * ( -1);
    }
    else{
      paso = ( (mensaje[6] * 0x100) + mensaje[7]);
      if ( (mensaje[1] & 0x0F) == 0)
        paso = paso * ( -1);
    }
    return paso;
  }

  private int calculaPasos(short IDMotor,double anguloFinal){
    double temp;

//multivueltas ó podemos dar una vuelta más
    if ((tablaMotores.getFlag(IDMotor) == 0) || ((
      tablaMotores.getVueltas(IDMotor) - tablaMotores.getVueltaActual(IDMotor)
      > 0) && (tablaMotores.getFlag(IDMotor) == 1))){//camino más corto
       if (((tablaMotores.getAnguloActual(IDMotor) >= 180)
```

```
        && (anguloFinal >=180)) || ((tablaMotores.getAnguloActual(IDMotor)
        < 180) && (anguloFinal <180))){

         return (int)Math.round((anguloFinal -
tablaMotores.getAnguloActual(IDMotor)) /
           tablaMotores.getAnguloPaso(IDMotor));
      }
      else{//NO estan en la misma mitad
        if ((tablaMotores.getAnguloActual(IDMotor) > 180)
           && (anguloFinal < 180)){
          if ((temp = (360.0 + anguloFinal -
                  tablaMotores.getAnguloActual(IDMotor))) <
                  (tablaMotores.getAnguloActual(IDMotor) - anguloFinal)){ //paso
                                                                  //por cero
           //aumentamos una vuelta
           return (int)Math.round(temp / tablaMotores.getAnguloPaso(IDMotor));
          }
          else
           return (int)Math.round(-((tablaMotores.getAnguloActual(IDMotor)-
anguloFinal)
                    / tablaMotores.getAnguloPaso(IDMotor)));
        }
        else{
          if ( (temp = (360.0 - anguloFinal +
                   tablaMotores.getAnguloActual(IDMotor))) <
             (anguloFinal - tablaMotores.getAnguloActual(IDMotor)) &&
             ((tablaMotores.getVueltaActual(IDMotor) > 0) ||
             (tablaMotores.getFlag(IDMotor) != 1))) { //paso por cero
           //disminuimos una vuelta
           return (int)Math.round( - (temp / tablaMotores.getAnguloPaso(IDMotor)));
          }
          else
           return (int)Math.round(((anguloFinal-
tablaMotores.getAnguloActual(IDMotor))
```

```
                    / tablaMotores.getAnguloPaso(IDMotor)));
        }
       }
      }
      else{//NO es posible aumentar una vuelta
       if (tablaMotores.getFlag(IDMotor) == 1){
        if (((tablaMotores.getAnguloActual(IDMotor) >= 180)
              && (anguloFinal >=180)) || ((tablaMotores.getAnguloActual(IDMotor)
              < 180) && (anguloFinal <180))){

          return (int)Math.round((anguloFinal -
tablaMotores.getAnguloActual(IDMotor))/
                    tablaMotores.getAnguloPaso(IDMotor));
        }

        if ((tablaMotores.getAnguloActual(IDMotor) > 180)
           && (anguloFinal < 180)){
           return (int)Math.round(-((tablaMotores.getAnguloActual(IDMotor)-
anguloFinal)
                    / tablaMotores.getAnguloPaso(IDMotor)));
        }
        else{
         if ( (temp = (360.0 - anguloFinal +
                    tablaMotores.getAnguloActual(IDMotor))) <
            (anguloFinal - tablaMotores.getAnguloActual(IDMotor))) {
           return (int)Math.round( - (temp / tablaMotores.getAnguloPaso(IDMotor)));
         }
         else
           return (int)Math.round(((anguloFinal -
tablaMotores.getAnguloActual(IDMotor))
                    / tablaMotores.getAnguloPaso(IDMotor)));
        }
       }
       else{  //angulo máximo
```

```
        if (anguloFinal > tablaMotores.getAnguloMaximo(IDMotor)){
         if (360.0 - anguloFinal <  //está mas cerca del 0
            anguloFinal - tablaMotores.getAnguloMaximo(IDMotor))

           return (int)Math.round( -(tablaMotores.getAnguloActual(IDMotor)) /
                    tablaMotores.getAnguloPaso(IDMotor));
         else  //está más cerca del ángulo máximo
           return (int)Math.round( (tablaMotores.getAnguloMaximo(IDMotor) -
                     tablaMotores.getAnguloActual(IDMotor)) /
                    tablaMotores.getAnguloPaso(IDMotor));
        }

        else //está dentro de los límites
          return (int)Math.round(anguloFinal -
tablaMotores.getAnguloActual(IDMotor));
       }
    }
  }
//-------------------------EVENTO GENERADO-----------------------------
   synchronized public void addMotorMovidoListener(MotorMovidoListener l) {
    if (listeners == null)
      listeners = new Vector();
      listeners.addElement(l);
    }

    synchronized public void removeMotorMovidoListener(MotorMovidoListener l) {
      if (listeners == null)
        listeners = new Vector();
        listeners.removeElement(l);
      }

   protected synchronized void avisaMovido(int tipo) {
    if (listeners != null && !listeners.isEmpty()) {
      MotorMovidoEvent evento = new MotorMovidoEvent(this,tipo);
```

```
        Vector targets;
        synchronized (this) {
          targets = (Vector) listeners.clone();
        }

        Enumeration enum = targets.elements();
        while (enum.hasMoreElements()) {
          MotorMovidoListener l =(MotorMovidoListener) enum.nextElement();
          l.motorMovido(evento);
        }
     }
   }
}
```

MotorMovidoEvent.java

```
//YA se ha movido el motor
public class MotorMovidoEvent extends java.util.EventObject{
//el sentido se indica con el signo del paso

  public int tipo; //indicará si es correcto ó ha habido algún error

  public MotorMovidoEvent(ToUSB source, int tipo){
    super(source);
    this.tipo = tipo;
 }
}

interface MotorMovidoListener extends java.util.EventListener {
 void motorMovido(MotorMovidoEvent e);
}
```

```
import java.util.*;

//LANZADO POR LA RED CUANDO QUIERE MOVER UN MOTOR

public class MoverMotorRedEvent extends java.util.EventObject{
   public double paso1;       //se cogen directamente
   public double paso2;
   public int velocidad1;
   public int velocidad2;
   public short ID; /**0 : Ambos Motores
                      1 : Motor 1
                      2 : Motor 2
                     */


   public MoverMotorRedEvent(MoverMotor source, double paso1,
                             double paso2, int velocidad1, int velocidad2){
     super(source);
     this.paso1 = paso1;
     this.paso2 = paso2;
     this.velocidad1 = velocidad1;
     this.velocidad2 = velocidad2;
     this.ID = 0;
   }

   public MoverMotorRedEvent(MoverMotor source, double pasoX,
                             int velocidadX, short ID){
     super(source);
     this.ID = ID;

     if (ID == 1){
       this.paso1 = pasoX;
       this.velocidad1 = velocidadX;
     }
```

```
      else{
        this.paso2 = pasoX;
        this.velocidad2 = velocidadX;
      }
    }
}

 interface MoverMotorRedListener extends java.util.EventListener {
   void moverMotorRed(MoverMotorRedEvent e);
 }

//clase que maneja los eventos de este tipo en el servidor de red
 class MoverMotor {
 Vector listeners;

 public MoverMotor(){
  listeners = new Vector();
 }

 synchronized public void addMoverMotorListener(MoverMotorRedListener l)
{//recepcion
   if (listeners == null)
     listeners = new Vector();
     listeners.addElement(l);
   }

   synchronized public void removeMoverMotorListener(MoverMotorRedListener l)
{
     if (listeners == null)
       listeners = new Vector();
       listeners.removeElement(l);
     }

   protected synchronized void avisaMotor(double angulo1, double angulo2,
```

```
                                    int velocidad1, int velocidad2) {
  if (listeners != null && !listeners.isEmpty()) {
    MoverMotorRedEvent evento = new MoverMotorRedEvent(this,
                                        angulo1,
                                        angulo2,
                                        velocidad1,
                                        velocidad2);

      Vector targets;
      synchronized (this) {
        targets = (Vector) listeners.clone();
      }

      Enumeration enum = targets.elements();
      while (enum.hasMoreElements()) {
        MoverMotorRedListener l =(MoverMotorRedListener) enum.nextElement();
        l.moverMotorRed(evento);
      }
   }
}

 protected synchronized void avisaMotor(double anguloX, int velocidadX,
                                    short ID) {
   if (listeners != null && !listeners.isEmpty()) {

     MoverMotorRedEvent evento = new MoverMotorRedEvent(this,
                                        anguloX,
                                        velocidadX,
                                        ID);

     Vector targets;
     synchronized (this) {
       targets = (Vector) listeners.clone();
     }
```

```
      Enumeration enum = targets.elements();
      while (enum.hasMoreElements()) {
        MoverMotorRedListener l =
        (MoverMotorRedListener) enum.nextElement();
        l.moverMotorRed(evento);
      }
     }
    }
  }
```

**********INTERFAZ GRÁFICA**********

PanelLoginServidor.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;

public class PanelLoginServidor extends JPanel{
  private TablaUsuarios usuarios;
  private TablaConexiones conexiones;
  private GridBagConstraints constraints;

  private JTextField alias;
  private JPasswordField password;

  public PanelLoginServidor(TablaUsuarios tusuarios, TablaConexiones
tconexiones){
    super();

    setName("panel central");

    JButton boton;
```

```
    this.usuarios = tusuarios;
    this.conexiones = tconexiones;

    constraints = new GridBagConstraints();
    setLayout(new GridBagLayout());

    alias = new JTextField(1);
    password = new JPasswordField(1);

    boton = new JButton("login");
    boton.setActionCommand("login");
    boton.setDefaultCapable(true);

    constraints.anchor = GridBagConstraints.EAST;
    addGB(this,new JLabel("login  "),0,0);
    constraints.ipadx = 99;
    constraints.ipady = 3;
    addGB(this,alias,1,0);
    constraints.ipady = 0;
    constraints.ipadx = 0;
    addGB(this,new JLabel(" "),0,1);
    addGB(this,new JLabel("password  "),0,2);
    constraints.ipadx = 100;
    addGB(this,password,1,2);
    constraints.ipady = 0;
    constraints.ipadx = 0;
    addGB(this,new JLabel(" "),0,3);
    constraints.anchor = GridBagConstraints.CENTER;
    addGB(this,boton,1,4);
/**listeners*/
    boton.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ev){
        try{
          if (((JButton) ev.getSource()).getActionCommand().equals("login")) {
```

```
//LOGIN
          switch (usuarios.getPermiso(alias.getText(),
                                password.getPassword())) {
            case TablaUsuarios.admin:
              conexiones.anadirConexion(alias.getText(),
                                InetAddress.getLocalHost(),
                                conexiones.admin);

              JOptionPane.showMessageDialog(foo(),
                 "Ha sido identificado correctamente");

              avisaLogin(alias.getText(),TablaUsuarios.admin); //avisamos al Frame

              preparaLogout( (JButton) ev.getSource());
              usuarios.actualizarAcceso(alias.getText());
              break;

            case TablaUsuarios.actuador:
            case TablaUsuarios.udp:
              JOptionPane.showMessageDialog(foo(), "No tiene permisos de " +
                                "Administrador");
              break;

            default:
              JOptionPane.showMessageDialog(foo(), "La información de login" +
                                " no es correcta");
              break;
          }
        }
//LOGOUT
       else{
         int result = JOptionPane.showConfirmDialog(foo(),
            "¿Desea abandonar la sesión?", "USBMotor",
            JOptionPane.YES_NO_OPTION);
```

```
        if (result == JOptionPane.YES_OPTION) {

          if (conexiones.eliminarConexionAdmin()) {
            JOptionPane.showMessageDialog(foo(), "Ha abandonado la sesión");
            preparaLogin( (JButton) ev.getSource());

            avisaLogin("login", (short) (TablaUsuarios.admin +
                                TablaUsuarios.actuador +
                                TablaUsuarios.udp));
            //avisamos al Frame de que es un logout
          }
          else
            JOptionPane.showMessageDialog(foo(),
                                "Se ha producido un error, " +
                                "no es posible realizar su desconexion");
        }
       }
      }
      catch(UnknownHostException e){
        conexiones.anadirConexion(alias.getText(),
                                null,
                                conexiones.admin);
        preparaLogout((JButton) ev.getSource());
        JOptionPane.showMessageDialog(foo(),"Ha sido identificado
correctamente");
      }
      catch(NullPointerException e){
        JOptionPane.showMessageDialog(foo(), "Debe completar todos los
campos" +  " antes de proceder al login");
      }
      finally{
        alias.setText("");
        password.setText("");
      }
```

```
    }
  });
}

private void preparaLogout(JButton boton){
  boton.setActionCommand("logout");
  boton.setText("logout");

  alias.setEnabled(false);
  password.setEnabled(false);
}

private void preparaLogin(JButton boton){
  boton.setActionCommand("login");
  boton.setText("login");

  alias.setEnabled(true);
  password.setEnabled(true);
}

private void addGB(JPanel panel,Component component,int x, int y){
  constraints.gridx = x;
  constraints.gridy = y;
  panel.add(component,constraints);
}

public void addPanelLoginListener(PanelLoginListener listener){
  listenerList.add(PanelLoginListener.class, listener);
}

public void removePanelLoginListener(PanelLoginListener listener) {
  listenerList.remove(PanelLoginListener.class, listener);
}
```

```
  void avisaLogin(String login,short permiso){
    Object[] listeners = listenerList.getListenerList();
    for(int i = 0;i < listeners.length; i += 2 )
      if (listeners[i] == PanelLoginListener.class)
        ((PanelLoginListener)listeners[i+1]).loged(
          new PanelLoginEvent(this,login,permiso));
  }

  private JPanel foo(){
    return this;
  }
}
```

PanelOpciones.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.beans.*;

  public class PanelOpciones extends JSplitPane{
  private OpcionesMotores tablaMotores;
  private OpcionesRed tablaRed;
  private boolean conectado_red;

  public PanelOpciones(OpcionesMotores valoresMotores, OpcionesRed
valoresRed,
                       boolean conectadoAred) {
    super();

    setName("panel central");

    this.tablaMotores = valoresMotores;
    this.tablaRed = valoresRed;
```

```
    this.conectado_red = conectadoAred;

    final JButton motoresButton;
    final JButton redButton;

    JPanel panelBotones = new JPanel();
    panelBotones.setLayout(new GridBagLayout());
    GridBagConstraints constraints = new GridBagConstraints();

    motoresButton = new JButton("Configuración de los motores");
    motoresButton.setActionCommand("Configuración de los motores");

    redButton = new JButton("Configuración de la red");
    redButton.setActionCommand("Configuración de la red");

    addGB(panelBotones, motoresButton, constraints, 0, 0);
    addGB(panelBotones, new JLabel(" "), constraints, 0,
        GridBagConstraints.RELATIVE);
    constraints.ipadx = 28;
    addGB(panelBotones, redButton, constraints, 0,
GridBagConstraints.RELATIVE);

    setRightComponent(panelBotones);

    JPanel panel = new JPanel();
    panel.setLayout(new GridBagLayout());
    constraints.ipadx = 0;
    constraints.fill = GridBagConstraints.BOTH;
    addGB(panel,new JLabel(crearIcono("iconos/USBMotor.png")),
        constraints,0,0);
    setLeftComponent(panel);

    addPropertyChangeListener(new PropertyChangeListener(){
      public void propertyChange(PropertyChangeEvent ev){
```

```
        ((JSplitPane)ev.getSource()).setDividerLocation(420);
      }
    });

    setDividerLocation(420);

    /***listeners*/
    motoresButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
        if (! (getLeftComponent() instanceof PanelOpcionesMotor)){
          setLeftComponent(new PanelOpcionesMotor(tablaMotores,
conectado_red));
          setDividerLocation(420);
          ((JButton)ev.getSource()).setEnabled(false);
          redButton.setEnabled(true);
        }

      }
    });

    //boton red
    redButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
        if (! (getLeftComponent() instanceof PanelOpcionesRed)){
          setLeftComponent(new PanelOpcionesRed(tablaRed, conectado_red));
          setDividerLocation(420);
          ((JButton)ev.getSource()).setEnabled(false);
          motoresButton.setEnabled(true);
        }
      }
    });
  }
```

```
  private void addGB(JPanel panel, Component component,
                     GridBagConstraints constraints, int x, int y) {
    constraints.gridx = x;
    constraints.gridy = y;
    panel.add(component, constraints);
  }

  public void conectado(boolean conectado){
    Component componente = getLeftComponent();

    if (componente instanceof PanelOpcionesRed)
      ((PanelOpcionesRed)componente).conectado(conectado);
    else
      if (componente instanceof PanelOpcionesMotor)
        ((PanelOpcionesMotor)componente).conectado(conectado);
  }

  private ImageIcon crearIcono(String path) {
    return new ImageIcon(path);
  }
}
```

PanelOpcionesMotor.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;

public class PanelOpcionesMotor extends JPanel
             implements ActionListener, ChangeListener{
  private GridBagConstraints constraints;
  private OpcionesMotores tablaMotores;
```

```
  private JButton aceptar;
  private JButton cancelar;
  private boolean conectado_red;

  public PanelOpcionesMotor(OpcionesMotores tablaMotor,boolean
conectado_red){
    super();
    this.conectado_red = conectado_red;
    this.tablaMotores = tablaMotor;

    setName("PanelOpcionesMotores");

    Border borde = BorderFactory.createEtchedBorder();
    setBorder(BorderFactory.createTitledBorder(borde,"Configuración de los
Motores"));

    constraints = new GridBagConstraints();
    setLayout(new GridBagLayout());

    JPanel tempPanel;

    tempPanel = new JPanel();
    tempPanel.setLayout(new GridBagLayout());
    tempPanel.setName("Menú Aceptar");

    aceptar = new JButton("Aceptar");
    aceptar.setActionCommand("aceptar");

    cancelar = new JButton("Cancelar");
    cancelar.setActionCommand("cancelar");

    addGB(tempPanel,aceptar,0,0);
    addGB(tempPanel,cancelar,1,0);
```

```
    constraints.gridheight = 2;
    constraints.anchor = GridBagConstraints.EAST;
    constraints.weighty = 0.15;

    addGB(this,tempPanel,0,3);

    constraints.ipady = 0;
    constraints.gridheight = 1;
    constraints.weighty = 0;

//Inicializamos los paneles de configuracion de los motores
    tempPanel = new JPanel();
    tempPanel.setLayout(new GridBagLayout());
    tempPanel.setBorder(BorderFactory.createTitledBorder(borde,"Motor 1"));
    tempPanel.setName("Motor1");
    crearPanelConfig(1,tempPanel);

    constraints.anchor = GridBagConstraints.WEST;
    addGB(this,tempPanel,0,0);

    tempPanel = new JPanel();
    tempPanel.setLayout(new GridBagLayout());
    tempPanel.setBorder(BorderFactory.createTitledBorder(borde,"Motor 2"));
    tempPanel.setName("Motor2");
    crearPanelConfig(2,tempPanel);

    constraints.anchor = GridBagConstraints.WEST;
    addGB(this,tempPanel,0,GridBagConstraints.RELATIVE);

    aceptar.setEnabled(false);
    cancelar.setEnabled(false);

//Panel de configuracion del fichero Log
    tempPanel = new JPanel();
```

```
    tempPanel.setLayout(new GridBagLayout());
    tempPanel.setBorder(BorderFactory.createTitledBorder(borde,"Tamaño del
archivo de registro"));
    tempPanel.setName("log");

    SpinnerNumberModel model = new SpinnerNumberModel(
      tablaMotores.getTamanoLog(),1,Integer.MAX_VALUE,1);
    JSpinner spinner = new JSpinner(model);
    spinner.setName("spinnerlog");

    //boton que prepara el archivo de log
    JButton logButton = new JButton("Ver Registro");
    logButton.setActionCommand("verRegistro");

    constraints.anchor = GridBagConstraints.CENTER;
    constraints.ipadx= -60;
    constraints.anchor = constraints.WEST;
    addGB(tempPanel,spinner,0,0);
    constraints.ipadx= 0;
    addGB(tempPanel,new JLabel(" kB"),1,0);
    constraints.anchor = constraints.EAST;
    constraints.ipadx = 120;
    addGB(tempPanel,new JLabel(" "),2,0);
    constraints.ipadx = 0;
    addGB(tempPanel,logButton,4,0);

    constraints.fill = GridBagConstraints.BOTH;
    addGB(this,tempPanel,0,GridBagConstraints.RELATIVE);


/***Listeners de los botones de accion*/
    aceptar.addActionListener(this);
    cancelar.addActionListener(this);
```

```
    spinner.addChangeListener(new ChangeListener(){
      public void stateChanged(ChangeEvent ev){
        aceptar.setEnabled(true); //ha habido cambios
        cancelar.setEnabled(true);
      }
    });

    logButton.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ev){
        tablaMotores.crearLogFinal();
        MuestraDocumento log = new MuestraDocumento("Archivo de Registro",
                                                    "Log.txt");
        log.pack();
        log.setVisible(true);
      }
    });
  }

  private void crearPanelConfig(final int ID,JPanel panel) {
    SpinnerNumberModel model;
    JSpinner angulo_paso;
    final JSpinner n_vueltas;
    final JSpinnerDial anguloMaximo;
    final JSpinnerDial anguloInicial;

    JRadioButton multivuelta;
    JRadioButton vueltas;
    JRadioButton giroMaximo;
    JSlider velocidad;
    final JLabel marcador;

    ButtonGroup grupoBotones = new ButtonGroup();

/***dialogo del angulo del paso*/
```

```
    model = new SpinnerNumberModel(
      tablaMotores.getAnguloPaso(ID),0.1, 360.0, 0.1);

    angulo_paso = new JSpinnerDial(0.1,tablaMotores.getAnguloPaso(ID));
    angulo_paso.setEnabled(conectado_red ? false : true);
    angulo_paso.setName("angulo_paso");

/***dialogo de tipo*/
    //radiobutton de Multivuelta
    multivuelta = new JRadioButton("Sin límite de vueltas");
    multivuelta.setActionCommand("0"); //asociado a su valor del flag
    multivuelta.setName("multivuelta");
    grupoBotones.add(multivuelta);  //añadimos al grupo excluyente

    //radiobutton de numero de vueltas
    vueltas = new JRadioButton("Máximo nº de vueltas");
    vueltas.setActionCommand("1"); //asociado a su valor del flag
    vueltas.setName("vueltas");
    grupoBotones.add(vueltas);
    //spinner del numero de vueltas:
    model = new SpinnerNumberModel(
      (int)tablaMotores.getVueltas(ID),0, 360, 1);
    n_vueltas = new JSpinner(model);
    n_vueltas.setName("n_vueltas");

    //radiobutton de angulo máximo de giro
    giroMaximo = new JRadioButton("Angulo máximo de giro (º)");
    giroMaximo.setActionCommand("2"); //asociado a su valor del flag
    giroMaximo.setName("giroMaximo");
    grupoBotones.add(giroMaximo);

    anguloMaximo = new JSpinnerDial(tablaMotores.getAnguloPaso(ID),
                                    tablaMotores.getAnguloMaximo(ID));
    anguloMaximo.setName("angulo_maximo");
```

```
/***dialogo de la posicion inicial*/
   anguloInicial = new JSpinnerDial(getSpinnerDouble(angulo_paso),
                              tablaMotores.getAnguloActual(ID));
   anguloInicial.setEnabled(conectado_red ? false : true);//solo activo si no esta
conectado a la red
   anguloInicial.setName("anguloInicial");

/***dialogo de velocidad máxima*/
   velocidad = new JSlider(JSlider.HORIZONTAL,1,255,
                        tablaMotores.getVelocidadMaxima(ID));
   velocidad.setName("velocidad");
   velocidad.setMajorTickSpacing(50);
   velocidad.setMinorTickSpacing(25);
   velocidad.setPaintTicks(true);
   velocidad.setEnabled(conectado_red ? false : true);

   marcador = new JLabel(cuentaRevoluciones(
     tablaMotores.getVelocidadMaxima(ID),
     (int)(360.0 / tablaMotores.getAnguloPaso(ID))));

//insertamos todos los elementos en el panel
   constraints.anchor = GridBagConstraints.WEST;
   constraints.gridwidth = 2;
   addGB(panel,new JLabel("       Angulo del Paso (º)"),0,0);

   constraints.ipadx = 180;
   addGB(panel,new JLabel(" "),GridBagConstraints.RELATIVE,0);
   constraints.ipadx = 0;

   constraints.anchor = GridBagConstraints.EAST;
   constraints.gridwidth = 2;
   addGB(panel,angulo_paso,2,0);
   constraints.gridwidth = 2;
```

```
    constraints.anchor = GridBagConstraints.WEST;
    addGB(panel,multivuelta,0,2);
    addGB(panel,vueltas,0,3);
    constraints.gridwidth = 1;
    constraints.anchor = GridBagConstraints.EAST;
    addGB(panel,n_vueltas,3,3);
    constraints.gridwidth = 2;
    constraints.anchor = GridBagConstraints.WEST;
    addGB(panel,giroMaximo,0,4);
    constraints.gridwidth = 1;
    constraints.anchor = GridBagConstraints.EAST;
    addGB(panel,anguloMaximo,3,4);
    constraints.anchor = GridBagConstraints.WEST;
    constraints.gridwidth = 2;
    addGB(panel,new JLabel("  Fijar Angulo de la posición actual        "),
        0,6);
    constraints.gridwidth = 1;
    constraints.anchor = GridBagConstraints.EAST;
    addGB(panel,anguloInicial,3,6);

    constraints.anchor = GridBagConstraints.WEST;
    addGB(panel,new JLabel("Velocidad Máxima: "),0,7);
    constraints.anchor = GridBagConstraints.EAST;
    addGB(panel, marcador,3,7);
    constraints.anchor = GridBagConstraints.WEST;
    constraints.fill = constraints.HORIZONTAL;
    constraints.gridwidth = constraints.REMAINDER;
    addGB(panel, velocidad,0,8);
    constraints.fill = constraints.NONE;
    constraints.gridwidth = 1;

/**Listeners:*/
    //RadioButton multivuelta
    multivuelta.addActionListener(new ActionListener(){
```

```
    public void actionPerformed(ActionEvent ev){
      n_vueltas.setEnabled(false);
      anguloMaximo.setEnabled(false);

      aceptar.setEnabled(true);
      cancelar.setEnabled(true);
    }
  });

  //RadioButton vueltas
  vueltas.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ev){
      n_vueltas.setEnabled(true);
      anguloMaximo.setEnabled(false);

      aceptar.setEnabled(true);
      cancelar.setEnabled(true);
    }
  });
  //RadioButton GiroMaximo
  giroMaximo.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ev){
      n_vueltas.setEnabled(false);
      anguloMaximo.setEnabled(true);

      aceptar.setEnabled(true);
      cancelar.setEnabled(true);
    }
  });

  //spinners:
  n_vueltas.addChangeListener(this);
  anguloMaximo.addChangeListener(this);
  anguloInicial.addChangeListener(this);
```

```
    angulo_paso.addChangeListener(new ChangeListener(){
      public void stateChanged(ChangeEvent ev){
        SpinnerNumberModel model = new SpinnerNumberModel(
          anguloCorrecto(getSpinnerDouble(
            anguloInicial),getSpinnerDouble( (JSpinner) ev.getSource())),
            0.0,360.0,(new Double(((JSpinner) ev.getSource()).getValue().
            toString())).doubleValue());

        anguloInicial.setModel(model);
        aceptar.setEnabled(true);
        cancelar.setEnabled(true);
      }
    });
    //slider de velocidad
    velocidad.addChangeListener(new ChangeListener(){
      public void  stateChanged(ChangeEvent ev){
        marcador.setText(cuentaRevoluciones(
       ((JSlider)ev.getSource()).getValue(),
       (int)(360.0 / tablaMotores.getAnguloPaso(ID))));

        aceptar.setEnabled(true);
        cancelar.setEnabled(true);
      }
    });

//Estado inicial de los RadioButtons
    switch(tablaMotores.getFlag(ID)){
      case 0:
        multivuelta.doClick();
        break;
      case 1:
        vueltas.doClick();
        break;
```

```
      case 2:
        giroMaximo.doClick();
        break;
    }
  }

//listener de los botones "aceptar" y "cancelar"
  public void actionPerformed(ActionEvent ev){
    if (((JButton)ev.getSource()).getActionCommand().equals("aceptar"))
      actionAceptar();
    else
      actionCancelar();

    aceptar.setEnabled(false);
    cancelar.setEnabled(false);
  }

private void actionAceptar(){
    JPanel temp;
    JSpinnerDial dial;
    double giroInicial1 = 0;
    double giroInicial2 = 0;

    int pasoAnterior1 = 0;
    int pasoAnterior2 = 0;

  //actualizamos los datos del panel 1
    temp = (JPanel)buscaComponente(this,"Motor1");

    for(int ID = 1; ID <= 2; ID++){
      if (conectado_red == false){
        tablaMotores.setAnguloPaso(ID, getSpinnerDouble(
            buscaComponente(temp, "angulo_paso")));
        }
```

```
  tablaMotores.setVueltas(ID, getSpinnerInt(
      buscaComponente(temp, "n_vueltas")));
  tablaMotores.setAnguloMaximo(ID, getSpinnerDouble(
      buscaComponente(temp, "angulo_maximo")));

  //si el angulo actual es mayor que el máximo
  if (getSpinnerDouble(buscaComponente(temp, "angulo_maximo")) <
      getSpinnerDouble(dial = (JSpinnerDial)buscaComponente(
        temp, "anguloInicial")))
     dial.setValue(new Double(getSpinnerDouble(buscaComponente(
        temp, "angulo_maximo"))));


  tablaMotores.setFlag(ID,buscaJRadioButtonPulsado(temp));

  tablaMotores.setVelocidadMaxima(ID,
       (short)(((JSlider)buscaComponente(temp, "velocidad")).getValue()));

  if (conectado_red == false){
    giroInicial2 = getSpinnerDouble(buscaComponente(temp, "anguloInicial"));
    pasoAnterior2 = tablaMotores.getPaso(ID);
    tablaMotores.setPaso(ID, (int) ( giroInicial2/
      tablaMotores.getAnguloPaso(ID)));
  }

  if (ID < 2){
    temp = (JPanel) buscaComponente(this, "Motor2");
    giroInicial1 = giroInicial2;
    pasoAnterior1 = pasoAnterior2;
  }
}

//Cambios en el tamaño del Log
```

```
    temp = (JPanel)buscaComponente(this,"log");
    int tamanoLog =
getSpinnerInt((JSpinner)buscaComponente(temp,"spinnerlog"));

    if (tablaMotores.getTamanoLog() != tamanoLog){
      tablaMotores.setTamanoLog(tamanoLog);
      JOptionPane.showMessageDialog(this,"Los cambios efectuados en el
tamaño " + "del Archivo de Registro tendrán efecto tras reiniciar la aplicación");
    }

    if (conectado_red == false){
      if (((giroInicial1/tablaMotores.getAnguloPaso(1)) != pasoAnterior1)  ||
        ((giroInicial2/tablaMotores.getAnguloPaso(2)) != pasoAnterior2))
          tablaMotores.movido(giroInicial1,0,giroInicial2,
                      0, "ADMIN"); //indicamos la nueva posicion inicial
    }

    tablaMotores.guardarTabla();    //guardamos los nuevos valores
}

private void actionCancelar(){
  JPanel temp;

  temp = (JPanel)buscaComponente(this,"Motor1");

  for(int ID = 1; ID <= 2; ID++){
    if (conectado_red == false){
      ((JSpinner)buscaComponente(temp, "angulo_paso")).setValue(
        new Double(tablaMotores.getAnguloPaso(ID)));
    }
    ((JSpinner)buscaComponente(temp, "n_vueltas")).setValue(
      new Integer(tablaMotores.getVueltas(ID)));
    ((JSpinner)buscaComponente(temp, "angulo_maximo")).setValue(
      new Double(tablaMotores.getAnguloMaximo(ID)));
```

```
    if (conectado_red == false){
      ((JSpinner)buscaComponente(temp, "anguloInicial")).setValue(
          new Double(tablaMotores.getPaso(ID) * getSpinnerDouble(
            buscaComponente(temp, "angulo_paso"))));
    }

    ((JSlider)buscaComponente(temp, "velocidad")).setValue(
      tablaMotores.getVelocidadMaxima(ID));

    if (ID < 2)
//actualizamos los datos del panel 2
      temp = (JPanel)buscaComponente(this,"Motor2");
  }
  temp = (JPanel)buscaComponente(this,"log");
  ((JSpinner)buscaComponente(temp, "spinnerlog")).setValue(
        new Integer(tablaMotores.getTamanoLog()));

    fijaEstadoRadioButtons(tablaMotores.getFlag(1),tablaMotores.getFlag(2));
}

  public void stateChanged(ChangeEvent ev){
    aceptar.setEnabled(true);
    cancelar.setEnabled(true);
  }

  private void addGB(JPanel panel,Component component,int x, int y){
      constraints.gridx = x;
      constraints.gridy = y;
      panel.add(component,constraints);
    }

  private Component buscaComponente(JPanel panel,String nombre){
    try {
      Component[] componentes = panel.getComponents();
```

```
      for (int i = 0; i < panel.getComponentCount(); i++) {
        if (componentes[i].getName() != null)
          if (componentes[i].getName().equals(nombre))
            return componentes[i];
      }
      return null;
    }
    catch (NullPointerException e) {
      return null;
    }
  }

//obtiene el valor del flag, buscando el ActionCommand del boton pulsado
  private short buscaJRadioButtonPulsado(JPanel panel){
    try {
      Component[] componente = panel.getComponents();

      for (int i = 0; i < panel.getComponentCount(); i++) {
          if ((componente[i]) instanceof JRadioButton)
            if (((JRadioButton)componente[i]).isSelected())
              return (new Short(((JRadioButton)componente[i])
                          .getActionCommand())).shortValue();
      }
      return 4;
    }
    catch (NullPointerException e) {
      return 4;
    }
  }

  private double getSpinnerDouble(Component componente){
    return (new Double(((JSpinner)componente).getValue().toString())).
        doubleValue();
```

```
}

private int getSpinnerInt(Component componente){
  return (new Double(((JSpinner)componente).getValue().toString())).
     intValue();
}

private void fijaEstadoRadioButtons(short flag1,short flag2){
  JPanel temp;

  switch(flag1){
    case 0:
      temp = (JPanel)buscaComponente(this,"Motor1");
      ((JRadioButton)buscaComponente((JPanel)temp,"multivuelta")).doClick();
      break;
    case 1:
      temp = (JPanel)buscaComponente(this,"Motor1");
      ((JRadioButton)buscaComponente((JPanel)temp,"vueltas")).doClick();
      break;
    case 2:
      temp = (JPanel)buscaComponente(this,"Motor1");
      ((JRadioButton)buscaComponente((JPanel)temp,"giroMaximo")).doClick();
      break;
  }
  switch(flag2){
      case 0:
        temp = (JPanel)buscaComponente(this,"Motor2");
        ((JRadioButton)buscaComponente((JPanel)temp,"multivuelta")).doClick();
        break;
      case 1:
        temp = (JPanel)buscaComponente(this,"Motor2");
        ((JRadioButton)buscaComponente((JPanel)temp,"vueltas")).doClick();
        break;
      case 2:
```

```
            temp = (JPanel)buscaComponente(this,"Motor2");
            ((JRadioButton)buscaComponente((JPanel)temp,"giroMaximo")).doClick();
            break;
        }
    }

    private double anguloCorrecto(double angulo, double angulo_paso){
        double temp = (double)(angulo / (angulo_paso));

        if (Math.floor(temp) == temp)
            return angulo;

        if (Math.floor(temp) + 0.5 > temp)
            return (Math.floor(temp))*(angulo_paso);
        else
            return (Math.floor(temp)+1)*(angulo_paso);
    }

    public void conectado(boolean conectado){
        this.conectado_red = conectado;

        JPanel panel;
        JSpinnerDial dial;
        JSlider slider;

        //actualizamos los componentes susceptibles del cambio (de conexion ó no
conexion)
        panel = (JPanel) buscaComponente(this,"Motor1");
        dial = (JSpinnerDial) buscaComponente(panel,"angulo_paso");
        dial.setEnabled(conectado_red ? false : true);
        dial = (JSpinnerDial) buscaComponente(panel,"anguloInicial");
        dial.setEnabled(conectado_red ? false : true);
        slider = (JSlider)buscaComponente(panel,"velocidad");
        slider.setEnabled(conectado_red ? false : true);
```

```
    panel = (JPanel) buscaComponente(this,"Motor2");
    dial = (JSpinnerDial) buscaComponente(panel,"angulo_paso");
    dial.setEnabled(conectado_red ? false : true);
    dial = (JSpinnerDial) buscaComponente(panel,"anguloInicial");
    dial.setEnabled(conectado_red ? false : true);
    slider = (JSlider)buscaComponente(panel,"velocidad");
    slider.setEnabled(conectado_red ? false : true);
  }

  private String cuentaRevoluciones(int velocidad,int n_pasos){
    Double rev = new Double(60.0 / (((0xFF - velocidad+1) * 256 * 0x40 *
                                  4.0/6000000.0) * (2.0 * n_pasos)));
    //(carga_timer{C A2} * 64_repeticiones * 4(divisor) / frec_oscilador)
    // * 2 * numero_pasos = tiempo que tarda en dar 1 vuelta
    //si lo invertimos y lo multiplicamos por 60, obtenemos las vueltas por minuto
    //es el doble del numero de pasos, ya que lo hacemos por la tecnica del medio-
paso
    String cadena = new String(rev.toString());
    if (cadena.length() >= cadena.indexOf('.') + 2) //1 o mas decimales
       return cadena.substring(0, cadena.indexOf('.') + 3) + " rpm";
    else
      return cadena + " rpm";
  }
}
```

MuestraDocumento.java

```
import java.io.*;
import javax.swing.*;
import java.awt.*;

public class MuestraDocumento extends JFrame{
```

```
  public MuestraDocumento(String tituloFrame, String nombreDocumento){
    super(tituloFrame);
    setSize(200,100);
    try{
      RandomAccessFile documento = new RandomAccessFile(nombreDocumento,
"r");
      JTextArea area = new JTextArea();
      String cadena;

      cadena = documento.readLine();
      while (cadena != null){ //cargamos el documento
        area.append(cadena + "\n");
        cadena = documento.readLine();
      }
      documento.close();
      area.setEditable(false); //no se puede editar
      JScrollPane scrollpane = new JScrollPane(area);
      scrollpane.setWheelScrollingEnabled(true);
      scrollpane.setPreferredSize(new Dimension(700,400));

      getContentPane().add(scrollpane);  //añadimos, con una barra
    }
    catch(IOException e){
      JOptionPane.showMessageDialog(this,"No ha sido posible abrir "
       + "el archivo de registro");
    }
  }
}
```

PanelOpcionesRed.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
```

```
import javax.swing.event.*;
import java.io.File;

public class PanelOpcionesRed extends JPanel
    implements ChangeListener,ActionListener{

  private GridBagConstraints constraints;
  private final OpcionesRed tablaRed;
  private JButton aceptar;
  private JButton cancelar;
  private boolean conectado;

  public PanelOpcionesRed(OpcionesRed valoresRed, boolean conectadoAred){
    super();
    this.tablaRed = valoresRed;
    this.conectado = conectadoAred;
    Border borde = BorderFactory.createEtchedBorder();
    setBorder(BorderFactory.createTitledBorder(borde,"Opciones de red"));

    constraints = new GridBagConstraints();
    setLayout(new GridBagLayout());

    JPanel tempPanel;

    JSpinner spinner;
    SpinnerNumberModel model;

    tempPanel = new JPanel();
    tempPanel.setLayout(new GridBagLayout());
    tempPanel.setBorder(BorderFactory.createTitledBorder(borde,
        "Número máximo de conexiones"));
    tempPanel.setName("panelMaxConexiones");

    model = new SpinnerNumberModel(tablaRed.getMaximoConexiones(),
```

```
                    0, Math.pow(2,16)-1, 1);
constraints.anchor = GridBagConstraints.WEST;
spinner = new JSpinner(model);
spinner.setName("maximoConexiones");
spinner.addChangeListener(this);
constraints.anchor = GridBagConstraints.EAST;
constraints.ipadx = -20;
addGB(tempPanel,new JLabel(" "),1,0);
addGB(tempPanel,spinner,1,1);
constraints.ipadx = 0;

constraints.fill = GridBagConstraints.BOTH;
addGB(this,tempPanel,0,0);

constraints.ipady = 10;
addGB(this,new JLabel(" "),0,1);
constraints.ipady = 0;
constraints.fill = GridBagConstraints.NONE;

tempPanel = new JPanel();
tempPanel.setLayout(new GridBagLayout());
tempPanel.setBorder(BorderFactory.createTitledBorder(borde,
   "Puerto de Gestión"));
tempPanel.setName("panelPuertoGestion");

model = new SpinnerNumberModel(tablaRed.getPuertoGestion(),
                    0, Math.pow(2,16)-1, 1);
constraints.anchor = GridBagConstraints.WEST;
spinner = new JSpinner(model);
spinner.setName("puertoGestion");
spinner.addChangeListener(this);
constraints.anchor = GridBagConstraints.EAST;
constraints.ipadx = -20;
addGB(tempPanel,new JLabel(" "),1,0);
```

```
addGB(tempPanel,spinner,1,1);
constraints.ipadx = 0;

constraints.fill = GridBagConstraints.BOTH;
addGB(this,tempPanel,0,2);

constraints.ipady = 5;
addGB(this,new JLabel(" "),0,3);
constraints.ipady = 0;

constraints.fill = GridBagConstraints.NONE;
constraints.ipadx = 59;

tempPanel = new JPanel();
tempPanel.setLayout(new GridBagLayout());
tempPanel.setBorder(BorderFactory.createTitledBorder(borde,"Timeout"));
tempPanel.setName("panelTimeout");

model = new SpinnerNumberModel(tablaRed.getTimeoutGestion(),
                    0, Math.pow(2,16)-1, 1);
constraints.anchor = GridBagConstraints.WEST;
addGB(tempPanel,new JLabel("Gestión (mseg)"),0,0);

constraints.ipadx = 130;
addGB(tempPanel,new JLabel(" "),GridBagConstraints.RELATIVE,0);

constraints.ipadx = 0;
spinner = new JSpinner(model);
spinner.setName("timeoutGestion");
spinner.addChangeListener(this);
constraints.anchor = GridBagConstraints.EAST;
constraints.ipadx = -20;
addGB(tempPanel,spinner,3,0);
constraints.ipadx = 59;
```

```
model = new SpinnerNumberModel(tablaRed.getTimeoutActuador(),
                        0, Math.pow(2,16)-1, 1);
constraints.anchor = GridBagConstraints.WEST;
addGB(tempPanel,new JLabel("Actuador (mseg)"),0,1);
constraints.ipadx = 0;
spinner = new JSpinner(model);
spinner.setName("timeoutActuador");
spinner.addChangeListener(this);
constraints.anchor = GridBagConstraints.EAST;
constraints.ipadx = -20;
addGB(tempPanel,spinner,3,1);
constraints.ipadx = 59;

model = new SpinnerNumberModel(tablaRed.getTimeoutEspectador(),
                        0, Math.pow(2,16)-1, 1);
constraints.anchor = GridBagConstraints.WEST;
addGB(tempPanel,new JLabel("Espectador (mseg)   "),0,2);
constraints.ipadx = 0;
spinner = new JSpinner(model);
spinner.setName("timeoutEspectador");
spinner.addChangeListener(this);
constraints.anchor = GridBagConstraints.EAST;
constraints.ipadx = -20;
addGB(tempPanel,spinner,3,2);

constraints.ipadx = 0;
constraints.fill = GridBagConstraints.BOTH;
addGB(this,tempPanel,0,4);

constraints.ipady = 5;
addGB(this,new JLabel(" "),0,5);
constraints.ipady = 0;

constraints.fill = GridBagConstraints.NONE;
```

```
constraints.ipadx = 59;

tempPanel = new JPanel();
tempPanel.setLayout(new GridBagLayout());
tempPanel.setBorder(BorderFactory.createTitledBorder(borde,
   "Duración de la conexion"));
tempPanel.setName("panelTiempo");

model = new SpinnerNumberModel(tablaRed.getTiempoConexionGestion(),
                        0, Math.pow(2,16)-1, 1);
constraints.anchor = GridBagConstraints.WEST;
addGB(tempPanel,new JLabel("Gestora (seg)"),0,0);

constraints.ipadx = 130;
addGB(tempPanel,new JLabel(" "),GridBagConstraints.RELATIVE,0);

constraints.ipadx = 0;
spinner = new JSpinner(model);
spinner.setName("tiempoConexionGestion");
spinner.addChangeListener(this);
constraints.anchor = GridBagConstraints.EAST;
constraints.ipadx = -20;
addGB(tempPanel,spinner,3,0);
constraints.ipadx = 59;

model = new SpinnerNumberModel(tablaRed.getTiempoConexionActuador(),
                        0, Math.pow(2,16)-1, 1);
constraints.anchor = GridBagConstraints.WEST;
addGB(tempPanel,new JLabel("Actuadora (seg)       "),0,1);
constraints.ipadx = 0;
spinner = new JSpinner(model);
spinner.setName("tiempoConexionActuador");
spinner.addChangeListener(this);
constraints.anchor = GridBagConstraints.EAST;
```

```
    constraints.ipadx = -20;
    addGB(tempPanel,spinner,3,1);

    constraints.ipadx = 0;
    constraints.fill = GridBagConstraints.BOTH;
    addGB(this,tempPanel,0,6);
    constraints.fill = GridBagConstraints.NONE;

    constraints.ipady = 5;
    addGB(this,new JLabel(" "),0,7);
    constraints.ipady = 0;

/****************Panel de localización de Applet*************/
    tempPanel = new JPanel();
    tempPanel.setLayout(new GridBagLayout());
    tempPanel.setBorder(BorderFactory.createTitledBorder(borde,"Ubicación de"
       +" la pagina principal del applet"));
    tempPanel.setName("panelApplet");

    JTextField cuadroRuta = new JTextField();
    cuadroRuta.setName("ruta");
    cuadroRuta.setText(this.tablaRed.getRuta());
    cuadroRuta.setEnabled(conectado ? false : true);

    JButton examinar = new JButton("Examinar...");
    examinar.setName("examinar");
    examinar.setEnabled(conectado ? false : true);

    constraints.anchor = GridBagConstraints.WEST;
    constraints.ipadx = 250;
    addGB(tempPanel,cuadroRuta,0,0);

    constraints.ipadx = 30;
    addGB(tempPanel,new JLabel(" "),1,0);
```

```
constraints.ipadx = 0;

constraints.anchor = GridBagConstraints.EAST;
addGB(tempPanel,examinar,2,0);

examinar.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent ev){
    JFileChooser chooser = new JFileChooser();

    HTMLFilter filtro = new HTMLFilter();
    chooser.setFileFilter(filtro);

    int returnVal = chooser.showOpenDialog(esto());

    if(returnVal == JFileChooser.APPROVE_OPTION) {
      JPanel panel = (JPanel)buscaComponente(esto(),"panelApplet");
      ((JTextField)buscaComponente(panel,"ruta")).setText(
                        chooser.getSelectedFile().getParent());
      aceptar.setEnabled(true);
      cancelar.setEnabled(true);
    }
  }
});

cuadroRuta.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent ev){
    aceptar.setEnabled(true);
    cancelar.setEnabled(true);
  }
});

constraints.ipadx = 0;
constraints.fill = GridBagConstraints.BOTH;
addGB(this,tempPanel,0,8);
```

```
    constraints.fill = GridBagConstraints.NONE;
/*************************************************************/

   aceptar = new JButton("Aceptar");
   aceptar.setActionCommand("aceptar");
   aceptar.addActionListener(this);
   aceptar.setEnabled(false);

   cancelar = new JButton("Cancelar");
   cancelar.setActionCommand("cancelar");
   cancelar.addActionListener(this);
   cancelar.setEnabled(false);

   tempPanel = new JPanel();
   tempPanel.setLayout(new GridBagLayout());

   constraints.anchor = GridBagConstraints.EAST;
   addGB(tempPanel,aceptar,0,0);
   constraints.anchor = GridBagConstraints.WEST;
   addGB(tempPanel,cancelar,1,0);

   constraints.weighty = 5;
   constraints.anchor = GridBagConstraints.SOUTHEAST;
   addGB(this,tempPanel,0,10);
   constraints.weighty = 0;
  }

  private void addGB(JPanel panel,Component component,int x, int y){
     constraints.gridx = x;
     constraints.gridy = y;
     panel.add(component,constraints);
  }

  //listener de los Spinners
```

```
  public void stateChanged(ChangeEvent ev){
    aceptar.setEnabled(true);
    cancelar.setEnabled(true);
  }

  public void actionPerformed(ActionEvent ev){
    if (((JButton)ev.getSource()).getActionCommand().equals("aceptar"))
      actionAceptar();
    else
      actionCancelar();
  }

  private void actionAceptar(){
    JPanel panel = (JPanel)buscaComponente(this,"panelMaxConexiones");
    tablaRed.setMaximoConexiones(getSpinnerInt(panel,"maximoConexiones"));

    panel = (JPanel)buscaComponente(this,"panelPuertoGestion");
    tablaRed.setPuertoGestion(getSpinnerInt(panel,"puertoGestion"));

    panel = (JPanel)buscaComponente(this,"panelTimeout");
    tablaRed.setTimeoutGestion(getSpinnerInt(panel,"timeoutGestion"));
    tablaRed.setTimeoutActuador(getSpinnerInt(panel,"timeoutActuador"));
    tablaRed.setTimeoutEspectador(getSpinnerInt(panel,"timeoutEspectador"));

    panel = (JPanel)buscaComponente(this,"panelTiempo");
    tablaRed.setTiempoConexionGestion(getSpinnerInt(
      panel,"tiempoConexionGestion"));
    tablaRed.setTiempoConexionActuador(getSpinnerInt(
      panel,"tiempoConexionActuador"));

//tratamiento del archivo del applet:
    panel = (JPanel)buscaComponente(this,"panelApplet");
    JTextField ruta = (JTextField)buscaComponente(panel,"ruta");
```

```
    File directorio = new File(ruta.getText());
    if (!directorio.isDirectory()){
      JOptionPane.showMessageDialog(this, "La ubicación indicada no es válida,"
                              + " se mantendrá la ubicación anterior");
    }
    else{
      tablaRed.setRuta(ruta.getText());
      aceptar.setEnabled(false);
      cancelar.setEnabled(false);
    }

    tablaRed.guardarTabla();    //guardamos los nuevos valores

    if (conectado)
      JOptionPane.showMessageDialog(this,"Los cambios tendrán efecto " +
                              "la próxima vez que inicie el servidor");
  }

   private void actionCancelar(){
    JPanel panel = (JPanel)buscaComponente(this,"panelMaxConexiones");
    ((JSpinner)buscaComponente(panel,"maximoConexiones")).setValue(new
Integer(
      tablaRed.getMaximoConexiones()));

    panel = (JPanel)buscaComponente(this,"panelPuertoGestion");
    ((JSpinner)buscaComponente(panel,"puertoGestion")).setValue(new Integer(
      tablaRed.getPuertoGestion()));

    panel = (JPanel)buscaComponente(this,"panelTimeout");
    ((JSpinner)buscaComponente(panel,"timeoutGestion")).setValue(new Integer(
      tablaRed.getTimeoutGestion()));
    ((JSpinner)buscaComponente(panel,"timeoutActuador")).setValue(new Integer(
      tablaRed.getTimeoutActuador()));
```

```
   ((JSpinner)buscaComponente(panel,"timeoutEspectador")).setValue(new
Integer(
      tablaRed.getTimeoutEspectador()));

     panel = (JPanel)buscaComponente(this,"panelTiempo");
   ((JSpinner)buscaComponente(panel,"tiempoConexionGestion")).setValue(new
Integer(
      tablaRed.getTiempoConexionGestion()));
   ((JSpinner)buscaComponente(panel,"tiempoConexionActuador")).setValue(new
Integer(
      tablaRed.getTiempoConexionActuador()));

    panel = (JPanel)buscaComponente(this,"panelApplet");
    JTextField ruta = (JTextField)buscaComponente(panel,"ruta");
    ruta.setText(tablaRed.getRuta());

    aceptar.setEnabled(false);
    cancelar.setEnabled(false);
  }


  private int getSpinnerInt(JPanel panel,String nombre){
      Component componente = buscaComponente(panel,nombre);
      return (new Double(((JSpinner)componente).getValue().toString())).
          intValue();
 }

 private Component buscaComponente(JPanel panel,String nombre){
 try {
   Component[] componentes = panel.getComponents();

   for (int i = 0; i < panel.getComponentCount(); i++) {
     if (componentes[i].getName() != null)
       if (componentes[i].getName().equals(nombre))
```

```
        return componentes[i];
    }
    return null;
  }
  catch (NullPointerException e) {
    return null;
  }
}

  public void conectado(boolean conectado){
    this.conectado = conectado;
  }

  public JPanel esto(){
    return this;
  }
}
```

entradaJTable.java

```
import java.util.*;

public class entradaJTable {
    public Vector vector;


   public entradaJTable(Vector entrada){
     this.vector = entrada;
   }

   public entradaJTable(){
     this.vector = new Vector();
   }
```

```
 public Vector getDataVector(){
   return vector;
 }
//-----------------GET-----------------
    synchronized String getLogin()
     throws ArrayIndexOutOfBoundsException, NullPointerException{
      return (String)vector.elementAt(0);
    }

    synchronized String getNombre()
     throws ArrayIndexOutOfBoundsException, NullPointerException{
      return (String)vector.elementAt(1);
    }

    synchronized String getPermiso()
     throws ArrayIndexOutOfBoundsException, NullPointerException{
      return (String)vector.elementAt(2);
    }

    synchronized Boolean getConectado()
     throws ArrayIndexOutOfBoundsException, NullPointerException{
      return (Boolean)vector.elementAt(3);
    }

    synchronized boolean getboolConectado()
     throws ArrayIndexOutOfBoundsException, NullPointerException{
      return ((Boolean)vector.elementAt(3)).booleanValue();
    }

    synchronized String getTipoConexion()
      throws ArrayIndexOutOfBoundsException, NullPointerException{
       return (String)vector.elementAt(4);
   }
```

```
    synchronized String getDireccion()
     throws ArrayIndexOutOfBoundsException, NullPointerException{
      return (String)vector.elementAt(5);
    }

   synchronized Date getUltimoAcceso()
     throws ArrayIndexOutOfBoundsException, NullPointerException{
      return (Date)vector.elementAt(6);
    }

//------------------SET-------------
   synchronized void setLogin(String login)
    throws ArrayIndexOutOfBoundsException, NullPointerException{
     vector.remove(0);
     vector.insertElementAt(login,0);
   }
   synchronized void setNombre(String nombre)
    throws ArrayIndexOutOfBoundsException, NullPointerException{
     vector.remove(1);
     vector.insertElementAt(nombre,1);
   }

   synchronized void setPermiso(String permiso)
    throws ArrayIndexOutOfBoundsException, NullPointerException{
     vector.remove(2);
     vector.insertElementAt(permiso,2);
   }

   synchronized void setConectado(boolean conectado)
    throws ArrayIndexOutOfBoundsException, NullPointerException{
     vector.remove(3);
     vector.insertElementAt(new Boolean(conectado),3);
   }
```

```
  synchronized void setTipoConexion(String tipoConexion)
   throws ArrayIndexOutOfBoundsException, NullPointerException{
    vector.remove(4);
    vector.insertElementAt(tipoConexion,4);
  }

  synchronized void setDireccion(String direccion)
   throws ArrayIndexOutOfBoundsException, NullPointerException{
    vector.remove(5);
    vector.insertElementAt(direccion,5);
  }

  synchronized void setUltimoAcceso(Date ultimoAcceso)
   throws ArrayIndexOutOfBoundsException, NullPointerException{
    vector.remove(6);
    vector.insertElementAt(ultimoAcceso,6);
  }

//------------------------NUEVO
synchronized void nuevoLogin(String login)
 throws ArrayIndexOutOfBoundsException, NullPointerException{
  vector.insertElementAt(login,0);
}

synchronized void nuevoNombre(String nombre)
 throws ArrayIndexOutOfBoundsException, NullPointerException{
  vector.insertElementAt(nombre,1);
}

synchronized void nuevoPermiso(String permiso)
 throws ArrayIndexOutOfBoundsException, NullPointerException{
  vector.insertElementAt(permiso,2);
}
```

```
  synchronized void nuevoConectado(boolean conectado)
   throws ArrayIndexOutOfBoundsException, NullPointerException{
    vector.insertElementAt(new Boolean(conectado),3);
  }

  synchronized void nuevoTipoConexion(String tipoConexion)
   throws ArrayIndexOutOfBoundsException, NullPointerException{
    vector.insertElementAt(tipoConexion,4);
  }


  synchronized void nuevoDireccion(String direccion)
   throws ArrayIndexOutOfBoundsException, NullPointerException{
    vector.insertElementAt(direccion,5);
  }

  synchronized void nuevoUltimoAcceso(Date ultimoAcceso)
   throws ArrayIndexOutOfBoundsException, NullPointerException{
    vector.insertElementAt(ultimoAcceso,6);
  }
}
```

JModeloTabla.java

```
import javax.swing.table.*;
import java.text.*;
import java.util.*;

public class JModeloTabla extends DefaultTableModel{
  private int columna_referencia;

  public JModeloTabla(){
      super();
```

```
      setColumnIdentifiers( new String[] {"Login", "Nombre", "Permiso",
       "Activo", "Conexión", "Dirección", "Ultimo Acceso"});

     columna_referencia = 3; //inicialmente ordenamos por conectados/no
conectados
  }

  public boolean isCellEditable(int fila, int columna){
    return false;
  }

  public void setComlumnaReferencia(int columnaReferencia){
    this.columna_referencia = columnaReferencia;
  }

  public Class getColumnClass(int columna){
    switch(columna){
      case 0:
      case 1:
      case 2:
      case 4:
      case 5:
        return (new String()).getClass();
      case 3:
        return (new Boolean(true)).getClass();
      case 6:
        return DateFormat.getDateInstance(DateFormat.SHORT).getClass();
      default:
       return null;
    }
  }

  public synchronized void anadirFila(String login, entradaUsuario usuario,
                    entradaConexion conexion){
```

```
  entradaJTable entrada = new entradaJTable(conformaTabla(
                                    login, usuario, conexion));

    int fila = buscaFila(entrada);
    insertRow(fila,entrada.getDataVector());
}

public synchronized void anadirFila(Vector entrada){
    int fila = buscaFila(new entradaJTable(entrada));
    insertRow(fila, entrada);
}

public synchronized boolean eliminaFila(String login){
 try{
   removeRow(buscaLogin(login));
   return true;
 }
  catch(ArrayIndexOutOfBoundsException e){
    return false;
  }
 }

 public synchronized boolean eliminaFila(int fila){
   try{
     removeRow(fila);
     return true;
   }
   catch(ArrayIndexOutOfBoundsException e){
    return false;
   }
 }

public synchronized boolean desconectaUsuario(entradaConexion entrada){
  String login = entrada.getLogin();
```

```
  entrada.setLogin("#ADIOS#");

  if(modificaFila(login,login,null,entrada)){
    entrada.setLogin(login);
    return true;
  }
  else{
    entrada.setLogin(login);
    return false;
  }
}

public synchronized boolean modificaFila(String oldLogin, String newLogin,
                    entradaUsuario usuario, entradaConexion conexion){
   int fila;
   Vector vector;
   try{
      fila = buscaLogin(oldLogin);

      vector = modificaEntrada((Vector)getDataVector().elementAt(fila),
                  newLogin,usuario,conexion);
      removeRow(fila);
      anadirFila(vector);
      return true;
    }
    catch(ArrayIndexOutOfBoundsException e){
      return false;
    }
    catch(NullPointerException e){
      return false;
    }
  }

 //crea el vector fila, a partir de la informacion de la tabla
```

```
private Vector conformaTabla(String login, entradaUsuario usuario,
                    entradaConexion conexion){
   entradaJTable entrada = new entradaJTable();

   entrada.nuevoLogin(login);
   entrada.nuevoNombre(usuario.getNombre());

   entrada.nuevoPermiso(getPermisoString(usuario.getPermisos()));


   if (conexion != null) {
     entrada.nuevoConectado(true);

     entrada.nuevoTipoConexion(getPermisoString(conexion.getTipo()));

     if (conexion.getDireccion().isAnyLocalAddress())
       entrada.nuevoDireccion(new String("local"));
     else
       entrada.nuevoDireccion(conexion.getDireccion().toString());
   }

   else{
     entrada.nuevoConectado(false);
     entrada.nuevoTipoConexion(new String("N.C."));
     entrada.nuevoDireccion(new String("N.C."));
   }

   entrada.nuevoUltimoAcceso(usuario.getUltimoAcceso());

   return entrada.getDataVector();
  }

private String getPermisoString(short permiso){
  switch(permiso){
```

```
      case TablaUsuarios.udp:
        return new String("Espectador");
      case TablaUsuarios.actuador:
        return new String("Actuador");
      case TablaUsuarios.admin:
        return new String("Administrador");
      default:
        return null;
    }
  }
//-------------------Funciones Para añadir ordenadamente---
  private int buscaFila(entradaJTable entrada){
    switch(columna_referencia){
      case 0:
        return buscaSitioString(entrada.getLogin(), 0, getRowCount(),
                          columna_referencia);
      case 1:
        return buscaSitioString(entrada.getNombre(), 0, getRowCount(),
                          columna_referencia);

      case 2:
        return buscaSitioPermiso(entrada.getPermiso(), entrada.getLogin(),
                           0, getRowCount(),columna_referencia);

      case 3:
        return buscaSitioConectado(entrada.getboolConectado(),
          entrada.getLogin(), 0, getRowCount(),entrada.getTipoConexion());

      case 4:
          return buscaSitioPermiso(entrada.getTipoConexion(), entrada.getLogin(),
                           0, getRowCount(),columna_referencia);

      case 5:
        return buscaSitioString(entrada.getDireccion(), 0, getRowCount(),
```

```
                        columna_referencia);
    case 6:
      return this.buscaSitioUltimoAcceso(entrada.getUltimoAcceso(),
       entrada.getLogin(),entrada.getboolConectado(),
        entrada.getTipoConexion());
    default:
      return 0;
  }
 }

 private int buscaSitioString(String busca, int primeraFila, int ultimaFila,
                        int columna){
  int i =primeraFila;

  while ((i < ultimaFila) && (busca.compareTo((String)getValueAt( i, columna)) >
0)){
    i++;
  }
  return i;
 }

 private int buscaSitioPermiso(String busca, String login, int primeraFila,
                        int ultimaFila, int columna){
  int primero = primeraFila;
  int ultimo;

  if (busca.equals("Actuador")){
   while ( (primero < ultimaFila) && (busca.compareTo((String)
      getValueAt(primero, columna)) != 0) &&
       (!((String)getValueAt(primero, columna)).equals("Espectador"))) {
     primero++;
   }

   if (primero < ultimaFila)
```

```
      if (busca.equals((String)getValueAt(primero, columna)))
        return (primero);
    }

    else{
      while ( (primero < ultimaFila) && (busca.compareTo((String)
          getValueAt(primero, columna)) != 0)) {
        primero++;
      }
    }

    if (primero == ultimaFila)
      return primero;

    ultimo = primero;

    while ( (ultimo < ultimaFila) && (busca.compareTo((String)
        getValueAt(ultimo, columna)) == 0)) {
      ultimo++;
    }
    return buscaSitioString(login, primero, ultimo,0);
    //dentro del tipo, ordenamos por login
  }

  private int buscaSitioConectado(boolean busca, String login, int primeraFila,
                                  int ultimaFila, String tipoConexion){
    int primero = primeraFila;
    int ultimo;
    int columna = 3;

    if ((getRowCount() == 0) ||
        ((((Boolean) getValueAt(primero, columna)).booleanValue() == false) &&
        (busca == true)))
        return primero;
```

```
  while ((primero < ultimaFila) && (((Boolean)getValueAt(primero, columna)).
     booleanValue() != busca)){
   primero++;
  }
  if (primero == getRowCount())
   return primero;

  ultimo = primero;
  while ((ultimo < ultimaFila) && (((Boolean)getValueAt(ultimo, columna)).
     booleanValue() == busca)){
   ultimo++;
  }
  return buscaSitioPermiso(tipoConexion,login, primero, ultimo,4);
}

private int buscaSitioUltimoAcceso(Date busca, String login,
                                   boolean conectado, String tipoConexion){
   int primero = 0, ultimo;
   DateFormat fecha = DateFormat.getDateInstance();
   int columna = 6;

   while ( (primero < getRowCount()) && (busca.before(
      (Date) getValueAt(primero, columna)))) {
    primero ++;
   }

   if (primero == getRowCount())
    return primero;

   ultimo = primero;
   while ( (ultimo < getRowCount()) && (busca.before(
      (Date) getValueAt(ultimo, columna)))) {
    ultimo ++;
   }
```

```
    return buscaSitioConectado(conectado, login, primero,ultimo,tipoConexion);
  }

//-----------------------------FUNCIONES PARA BUSCAR
  private int buscaLogin(String login){
    Vector vector = getDataVector();
    int i =0;
    try{
      while ( (i < vector.size()) && (!((String)((Vector) vector.elementAt(i)).
                                  elementAt(0)).equals(login))) {
        i++;
      }

      if (i < vector.size())
        return i;
      else
        return vector.size();
    }
    catch (NullPointerException e){
      return vector.size();
    }
  }

//-------------------------FUNCIONES PARA MODIFICAR UNA FILA
  private Vector modificaEntrada(Vector vector, String newLogin,
                                 entradaUsuario usuario,
                                 entradaConexion conexion)
          throws ArrayIndexOutOfBoundsException, NullPointerException{

      entradaJTable entrada = new entradaJTable(vector);

      if (!newLogin.equals(entrada.getLogin()))
        entrada.setLogin(newLogin);
```

```
    if (usuario != null) {
      if (!entrada.getNombre().equals(usuario.getNombre()))
        entrada.setNombre(usuario.getNombre());
      if (!entrada.getPermiso().equals(getPermisoString(usuario.getPermisos())))
        entrada.setPermiso(getPermisoString(usuario.getPermisos()));
      if (!entrada.getUltimoAcceso().equals(usuario.getUltimoAcceso()))
        entrada.setUltimoAcceso(usuario.getUltimoAcceso());
    }
    if (conexion != null) {
      if (conexion.getLogin().equals("#ADIOS#")){
        entrada.setConectado(false);
        entrada.setDireccion("N.C.");
        entrada.setTipoConexion("N.C.");
      }
      else{
        if (entrada.getConectado().booleanValue() == false)
          entrada.setConectado(true);
        if
(!entrada.getTipoConexion().equals(getPermisoString(conexion.getTipo())))
          entrada.setTipoConexion(getPermisoString(conexion.getTipo()));
        if (!entrada.getDireccion().equals(conexion.getDireccion().toString()))
          entrada.setDireccion(conexion.getDireccion().toString());
      }
    }

    return entrada.getDataVector();
  }
}
```

TableSorter.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
```

```
import java.util.List;

import javax.swing.*;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.*;

public class TableSorter extends AbstractTableModel {
    protected TableModel tableModel;

    public static final int DESCENDING = -1;
    public static final int NOT_SORTED = 0;
    public static final int ASCENDING = 1;

    private static Directive EMPTY_DIRECTIVE = new Directive(-1,
NOT_SORTED);

    public static final Comparator COMPARABLE_COMAPRATOR = new
Comparator() {
        public int compare(Object o1, Object o2) {
            return ((Comparable) o1).compareTo(o2);
        }
    };
    public static final Comparator LEXICAL_COMPARATOR = new Comparator() {
        public int compare(Object o1, Object o2) {
            return o1.toString().compareTo(o2.toString());
        }
    };

    private Row[] viewToModel;
    private int[] modelToView;

    private JTableHeader tableHeader;
    private MouseListener mouseListener;
```

```
    private TableModelListener tableModelListener;
    private Map columnComparators = new HashMap();
    private List sortingColumns = new ArrayList();

    public TableSorter() {
        this.mouseListener = new MouseHandler();
        this.tableModelListener = new TableModelHandler();
    }

    public TableSorter(TableModel tableModel) {
        this();
        setTableModel(tableModel);
    }

    public TableSorter(TableModel tableModel, JTableHeader tableHeader) {
        this();
        setTableHeader(tableHeader);
        setTableModel(tableModel);
    }

    private void clearSortingState() {
        viewToModel = null;
        modelToView = null;
    }

    public TableModel getTableModel() {
        return tableModel;
    }

    public void setTableModel(TableModel tableModel) {
        if (this.tableModel != null) {
            this.tableModel.removeTableModelListener(tableModelListener);
        }
```

```
        this.tableModel = tableModel;
        if (this.tableModel != null) {
            this.tableModel.addTableModelListener(tableModelListener);
        }

        clearSortingState();
        fireTableStructureChanged();
    }

    public JTableHeader getTableHeader() {
        return tableHeader;
    }

    public void setTableHeader(JTableHeader tableHeader) {
        if (this.tableHeader != null) {
            this.tableHeader.removeMouseListener(mouseListener);
            TableCellRenderer defaultRenderer =
this.tableHeader.getDefaultRenderer();
            if (defaultRenderer instanceof SortableHeaderRenderer) {
                this.tableHeader.setDefaultRenderer(((SortableHeaderRenderer)
defaultRenderer).tableCellRenderer);
            }
        }
        this.tableHeader = tableHeader;
        if (this.tableHeader != null) {
            this.tableHeader.addMouseListener(mouseListener);
            this.tableHeader.setDefaultRenderer(
                    new
SortableHeaderRenderer(this.tableHeader.getDefaultRenderer()));
        }
    }

    public boolean isSorting() {
        return sortingColumns.size() != 0;
```

```
    }

    private Directive getDirective(int column) {
        for (int i = 0; i < sortingColumns.size(); i++) {
            Directive directive = (Directive)sortingColumns.get(i);
            if (directive.column == column) {
                return directive;
            }
        }
        return EMPTY_DIRECTIVE;
    }
    public int getSortingStatus(int column) {
        return getDirective(column).direction;
    }

    private void sortingStatusChanged() {
        clearSortingState();
        fireTableDataChanged();
        if (tableHeader != null) {
            tableHeader.repaint();
        }
    }

    public void setSortingStatus(int column, int status) {
        Directive directive = getDirective(column);
        if (directive != EMPTY_DIRECTIVE) {
            sortingColumns.remove(directive);
        }
        if (status != NOT_SORTED) {
            sortingColumns.add(new Directive(column, status));
        }
        sortingStatusChanged();
    }
```

```
    protected Icon getHeaderRendererIcon(int column, int size) {
        Directive directive = getDirective(column);
        if (directive == EMPTY_DIRECTIVE) {
            return null;
        }
        return new Arrow(directive.direction == DESCENDING, size,
sortingColumns.indexOf(directive));
    }

    private void cancelSorting() {
        sortingColumns.clear();
        sortingStatusChanged();
    }

    public void setColumnComparator(Class type, Comparator comparator) {
        if (comparator == null) {
            columnComparators.remove(type);
        } else {
            columnComparators.put(type, comparator);
        }
    }

    protected Comparator getComparator(int column) {
        Class columnType = tableModel.getColumnClass(column);
        Comparator comparator = (Comparator)
columnComparators.get(columnType);
        if (comparator != null) {
            return comparator;
        }
        if (Comparable.class.isAssignableFrom(columnType)) {
            return COMPARABLE_COMAPRATOR;
        }
        return LEXICAL_COMPARATOR;
    }
```

```
    private Row[] getViewToModel() {
        if (viewToModel == null) {
            int tableModelRowCount = tableModel.getRowCount();
            viewToModel = new Row[tableModelRowCount];
            for (int row = 0; row < tableModelRowCount; row++) {
                viewToModel[row] = new Row(row);
            }

            if (isSorting()) {
                Arrays.sort(viewToModel);
            }
        }
        return viewToModel;
    }

    public int modelIndex(int viewIndex) {
        return getViewToModel()[viewIndex].modelIndex;
    }

    private int[] getModelToView() {
        if (modelToView == null) {
            int n = getViewToModel().length;
            modelToView = new int[n];
            for (int i = 0; i < n; i++) {
                modelToView[modelIndex(i)] = i;
            }
        }
        return modelToView;
    }

    // TableModel interface methods

    public int getRowCount() {
        return (tableModel == null) ? 0 : tableModel.getRowCount();
```

```
}

public int getColumnCount() {
    return (tableModel == null) ? 0 : tableModel.getColumnCount();
}

public String getColumnName(int column) {
    return tableModel.getColumnName(column);
}

public Class getColumnClass(int column) {
    return tableModel.getColumnClass(column);
}

public boolean isCellEditable(int row, int column) {
    return tableModel.isCellEditable(modelIndex(row), column);
}

public Object getValueAt(int row, int column) {
    return tableModel.getValueAt(modelIndex(row), column);
}

public void setValueAt(Object aValue, int row, int column) {
    tableModel.setValueAt(aValue, modelIndex(row), column);
}

// Helper classes

private class Row implements Comparable {
    private int modelIndex;

    public Row(int index) {
        this.modelIndex = index;
    }
```

```
        public int compareTo(Object o) {
            int row1 = modelIndex;
            int row2 = ((Row) o).modelIndex;

            for (Iterator it = sortingColumns.iterator(); it.hasNext();) {
                Directive directive = (Directive) it.next();
                int column = directive.column;
                Object o1 = tableModel.getValueAt(row1, column);
                Object o2 = tableModel.getValueAt(row2, column);

                int comparison = 0;
                // Define null less than everything, except null.
                if (o1 == null && o2 == null) {
                    comparison = 0;
                } else if (o1 == null) {
                    comparison = -1;
                } else if (o2 == null) {
                    comparison = 1;
                } else {
                    comparison = getComparator(column).compare(o1, o2);
                }
                if (comparison != 0) {
                    return directive.direction == DESCENDING ? -comparison :
comparison;
                }
            }
            return 0;
        }
    }

    private class TableModelHandler implements TableModelListener {
        public void tableChanged(TableModelEvent e) {
            if (!isSorting()) {
```

```
            clearSortingState();
            fireTableChanged(e);
            return;
        }

        if (e.getFirstRow() == TableModelEvent.HEADER_ROW) {
            cancelSorting();
            fireTableChanged(e);
            return;
        }

         int column = e.getColumn();
        if (e.getFirstRow() == e.getLastRow()
                && column != TableModelEvent.ALL_COLUMNS
                && getSortingStatus(column) == NOT_SORTED
                && modelToView != null) {
            int viewIndex = getModelToView()[e.getFirstRow()];
            fireTableChanged(new TableModelEvent(TableSorter.this,
                                        viewIndex, viewIndex,
                                        column, e.getType()));
            return;
        }

        clearSortingState();
        fireTableDataChanged();
        return;
    }
}

private class MouseHandler extends MouseAdapter {
    public void mouseClicked(MouseEvent e) {
        JTableHeader h = (JTableHeader) e.getSource();
        TableColumnModel columnModel = h.getColumnModel();
        int viewColumn = columnModel.getColumnIndexAtX(e.getX());
```

```
            int column = columnModel.getColumn(viewColumn).getModelIndex();
            if (column != -1) {
                int status = getSortingStatus(column);
                if (!e.isControlDown()) {
                    cancelSorting();
                }
                status = status + (e.isShiftDown() ? -1 : 1);
                status = (status + 4) % 3 - 1; // signed mod, returning {-1, 0, 1}
                setSortingStatus(column, status);
            }
        }
    }

    private static class Arrow implements Icon {
        private boolean descending;
        private int size;
        private int priority;

        public Arrow(boolean descending, int size, int priority) {
            this.descending = descending;
            this.size = size;
            this.priority = priority;
        }

        public void paintIcon(Component c, Graphics g, int x, int y) {
            Color color = c == null ? Color.GRAY : c.getBackground();
            int dx = (int)(size/2*Math.pow(0.8, priority));
            int dy = descending ? dx : -dx;
            y = y + 5*size/6 + (descending ? -dy : 0);
            int shift = descending ? 1 : -1;
            g.translate(x, y);

            g.setColor(color.darker());
            g.drawLine(dx / 2, dy, 0, 0);
```

```
            g.drawLine(dx / 2, dy + shift, 0, shift);

            g.setColor(color.brighter());
            g.drawLine(dx / 2, dy, dx, 0);
            g.drawLine(dx / 2, dy + shift, dx, shift);

            if (descending) {
                g.setColor(color.darker().darker());
            } else {
                g.setColor(color.brighter().brighter());
            }
            g.drawLine(dx, 0, 0, 0);

            g.setColor(color);
            g.translate(-x, -y);
        }

        public int getIconWidth() {
            return size;
        }

        public int getIconHeight() {
            return size;
        }
    }

    private class SortableHeaderRenderer implements TableCellRenderer {
        private TableCellRenderer tableCellRenderer;

        public SortableHeaderRenderer(TableCellRenderer tableCellRenderer) {
            this.tableCellRenderer = tableCellRenderer;
        }

        public Component getTableCellRendererComponent(JTable table,
```

```
                                        Object value,
                                        boolean isSelected,
                                        boolean hasFocus,
                                        int row,
                                        int column) {
        Component c = tableCellRenderer.getTableCellRendererComponent(table,
                value, isSelected, hasFocus, row, column);
        if (c instanceof JLabel) {
            JLabel l = (JLabel) c;
            l.setHorizontalTextPosition(JLabel.LEFT);
            int modelColumn = table.convertColumnIndexToModel(column);
            l.setIcon(getHeaderRendererIcon(modelColumn, l.getFont().getSize()));
        }
        return c;
    }
}

private static class Directive {
    private int column;
    private int direction;

    public Directive(int column, int direction) {
        this.column = column;
        this.direction = direction;
    }
}

//--------------------------FUNCIONES PROPIAS  (basicamente para hacer la
traducción)
public Vector getDataVector(int fila){
  return (Vector)(((JModeloTabla)tableModel).getDataVector().
            elementAt(modelIndex(fila)));
}
```

```
public boolean desconectaUsuario(entradaConexion entrada){
  if (((JModeloTabla)tableModel).desconectaUsuario(entrada))
    return true;
  else
    return false;
}

public void anadirFila(String login, entradaUsuario usuario,
                entradaConexion conexion){
  ((JModeloTabla)tableModel).anadirFila(login,usuario,conexion/*,false*/);
//NO requiere traducción
}

public synchronized boolean modificaFila(String oldLogin, String newLogin,
                    entradaUsuario usuario, entradaConexion conexion){
  if (((JModeloTabla)tableModel).modificaFila(oldLogin,newLogin,
                                usuario,conexion))
    return true;
  else
    return false;
}

public boolean eliminaFila(int fila){
  if (((JModeloTabla)tableModel).eliminaFila(modelIndex(fila)))
    return true;
  else
    return false;
}

public boolean eliminaFila(String login){
  if (((JModeloTabla)tableModel).eliminaFila(login))//NO requiere traducción
    return true;
  else
    return false;
```

```
    }

    public boolean isConectado(int fila){
      entradaJTable entrada = new entradaJTable(getDataVector(fila));
      return entrada.getConectado().booleanValue();
    }
}
```

PanelUsuarios.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;
import java.beans.*;

public class PanelUsuarios extends JSplitPane
    implements UsuarioListener,ConexionListener{
  private TablaUsuarios tablaUsuarios;
  private TablaConexiones tablaConexiones;
  private int filaSeleccionada;
  private TableSorter modelo;
  private ServidorRed servidor;

  DialogoUsuario dialogo;

  public PanelUsuarios(TablaUsuarios tablaU, TablaConexiones tablaC,
                       ServidorRed servidorRed) {
    super();

    setName("panel central");

    final JTable tabla;
```

```
    this.tablaUsuarios = tablaU;
    this.tablaConexiones = tablaC;

    this.servidor = servidorRed;

    TableSorter sorter = new TableSorter(new JModeloTabla());
    tabla = new JTable(sorter);
    sorter.setTableHeader(tabla.getTableHeader());

    modelo =(TableSorter) tabla.getModel();

    tabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    tabla.setDragEnabled(false);

    tabla.setName("tabla");

    ToolTipManager.sharedInstance().unregisterComponent(tabla);

ToolTipManager.sharedInstance().unregisterComponent(tabla.getTableHeader());

    setColumnWidth(tabla);
    tabla.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    inicializaTabla(tabla);

    JScrollPane panelTabla = new JScrollPane(tabla);
    panelTabla.setHorizontalScrollBarPolicy(JScrollPane.
                                HORIZONTAL_SCROLLBAR_ALWAYS);

panelTabla.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AL
WAYS);

    setLeftComponent(panelTabla);

    addPropertyChangeListener(new PropertyChangeListener(){
```

```
      public void propertyChange(PropertyChangeEvent ev){
        ((JSplitPane)ev.getSource()).setDividerLocation(475);
      }
    });

    this.setDividerLocation(475);

//------Panel Derecho (botones)
    final JPanel panelBotones = new JPanel();
    panelBotones.setLayout(new GridBagLayout());

    GridBagConstraints constraints = new GridBagConstraints();
    JButton anadirUsuario;
    JButton modificaUsuario;
    JButton eliminaUsuario;
    JButton desconectaUsuario;

    anadirUsuario = new JButton("Añadir Usuario");
    anadirUsuario.setName("anadirUsuario");

    modificaUsuario = new JButton("Modificar Valores");
    modificaUsuario.setName("modificausuario");
    modificaUsuario.setEnabled(false);

    eliminaUsuario = new JButton("Eliminar Usuario");
    eliminaUsuario.setName("eliminausuario");
    eliminaUsuario.setEnabled(false);

    desconectaUsuario = new JButton("Desconectar Usuario");
    desconectaUsuario.setName("desconectausuario");
    desconectaUsuario.setEnabled(false);

    constraints.ipadx = 30;
    addGB(panelBotones,anadirUsuario,constraints,0,0);
```

```
constraints.ipadx = 100;
addGB(panelBotones, new JLabel(" "),constraints,0,1);
constraints.ipadx = 18;
addGB(panelBotones,modificaUsuario,constraints,0,2);
addGB(panelBotones, new JLabel(" "),constraints,0,3);
constraints.ipadx = 25;
addGB(panelBotones,eliminaUsuario,constraints,0,4);
addGB(panelBotones, new JLabel(" "),constraints,0,5);
constraints.ipadx = 0;
addGB(panelBotones,desconectaUsuario,constraints,0,6);

setRightComponent(panelBotones);

/***Listeners*/
//Tabla (seleccionada)
ListSelectionModel rowSM = tabla.getSelectionModel();
rowSM.addListSelectionListener(new ListSelectionListener() {
  public void valueChanged(ListSelectionEvent ev) {
    if (ev.getValueIsAdjusting())return;

    ListSelectionModel lsm = (ListSelectionModel) ev.getSource();
    if (lsm.isSelectionEmpty()) {
      ((JButton)buscaComponente(panelBotones,"modificausuario")).
          setEnabled(false);
      ((JButton)buscaComponente(panelBotones,"eliminausuario")).
          setEnabled(false);
      ((JButton)buscaComponente(panelBotones,"desconectausuario")).
          setEnabled(false);
    }
    else {
      setFilaSeleccionada(lsm.getMinSelectionIndex());

      ((JButton)buscaComponente(panelBotones,"modificausuario")).
        setEnabled(true);
```

```
      ((JButton)buscaComponente(panelBotones,"eliminausuario")).
          setEnabled(true);

       if (((Boolean)tabla.getValueAt(getFilaSeleccionada(),3)).
          booleanValue())
        ((JButton)buscaComponente(panelBotones,"desconectausuario")).
            setEnabled(true);
       else
        ((JButton)buscaComponente(panelBotones,"desconectausuario")).
            setEnabled(false);
     }
   }
});

//boton "añadir usuario"
anadirUsuario.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ev) {
    if (dialogo == null) {
      dialogo = new DialogoUsuario("USBMotor : Añadir Usuario", tablaUsuarios,
                                   null,null);
      dialogo.setVisible(true);
    }
    else
    if (!dialogo.isVisible()) {
      dialogo = new DialogoUsuario("USBMotor : Añadir Usuario", tablaUsuarios,
                                   null,null);
      dialogo.setVisible(true);
    }
  }
});

//boton "modifica usuario"
modificaUsuario.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent ev){
```

```
    if (dialogo == null) {
      dialogo = new DialogoUsuario("USBMotor : Modificar Usuario",
              tablaUsuarios,tablaConexiones,
              modelo.getDataVector(getFilaSeleccionada()));
      dialogo.setVisible(true);
    }
    else
    if (!dialogo.isVisible()) {
      dialogo = new DialogoUsuario("USBMotor : Modificar entrada Usuario",
              tablaUsuarios,tablaConexiones,
              modelo.getDataVector(getFilaSeleccionada()));
      dialogo.setVisible(true);
    }
  }
});

//boton eliminaUsuario
eliminaUsuario.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent ev){
    int result = JOptionPane.showConfirmDialog(foo(),
       "¿Está seguro de que desea eliminar al usuario?", "USBMotor",
       JOptionPane.YES_NO_OPTION);

    if (result == JOptionPane.YES_OPTION) {

      int fila = getFilaSeleccionada();
      String login = (String) tabla.getValueAt(getFilaSeleccionada(), 0);
      if (modelo.isConectado(fila)) {
        int pos = tablaConexiones.buscaLogin(login);

        entradaConexion entrada = tablaConexiones.getEntrada(pos);

        if (entrada.getTipo() != TablaConexiones.admin) {
```

```
          if (entrada.getTipo() == TablaConexiones.udp) {
            if (servidor != null)
              servidor.desconectarUDP_GUI(entrada);
            else
              tablaConexiones.eliminarConexion(pos);
          }
          else {
            if (entrada.getTipo() == TablaConexiones.actuador)
              if (servidor != null)
                servidor.desconectarTCP_GUI(pos);
              else
                tablaConexiones.eliminarConexion(pos);
          }
          tablaUsuarios.eliminarUsuario(login);
        }
        else
          dialogo("No se ha podido eliminar el usuario");
      }
      else {
        tablaUsuarios.eliminarUsuario(login);
      }
    }
  }
  });

  //boton desconectaUsuario
  desconectaUsuario.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ev){

      int result = JOptionPane.showConfirmDialog(foo(),
          "¿Está seguro de que desea desconectar al usuario?", "USBMotor",
          JOptionPane.YES_NO_OPTION);

      if (result == JOptionPane.YES_OPTION) {
```

```
        int pos = tablaConexiones.buscaLogin( (String) tabla.
                                    getValueAt(getFilaSeleccionada(),
           0));
        entradaConexion entrada = tablaConexiones.getEntrada(pos);
        if (entrada.getTipo() != TablaConexiones.admin) {
          if (entrada.getTipo() == TablaConexiones.udp) {
            if (servidor != null)
              servidor.desconectarUDP_GUI(entrada);
            else
              tablaConexiones.eliminarConexion(pos);
          }

          else {
            if (entrada.getTipo() == TablaConexiones.actuador) {
              if (servidor != null)
                servidor.desconectarTCP_GUI(pos);
              else
                tablaConexiones.eliminarConexion(pos);
            }
          }
        }
        else
          dialogo("No ha sido posible desconectar el usuario");
      }
    }
    });

//tablas
   tablaUsuarios.addUsuarioListener(this);
   tablaConexiones.addConexionListener(this);
  }

//Eventos de usuario
```

```
public void recibido(EventoUsuario e) {

  switch (e.suceso) {
    case 0: //nueva entrada
      modelo.anadirFila(e.login, tablaUsuarios.getEntrada(e.login), null);
      break;
    case 1: //entrada eliminada
      if (!modelo.eliminaFila(e.login))
        dialogo("Error de sincronización en la tabla de usuarios");
      break;
    case 2: //entrada modificada
      if (e.login.indexOf('#') >= 0) {
        modelo.modificaFila(e.login.substring(0, e.login.indexOf('#')),
                    e.login.substring(e.login.indexOf('#') + 1,
                                e.login.length()),
                    tablaUsuarios.getEntrada(e.login),
                    tablaConexiones.getEntrada_login(e.login));

        tablaConexiones.setLogin(
            e.login.substring(0, e.login.indexOf('#')),
            e.login.substring(e.login.indexOf('#') + 1, e.login.length()));
      }

      else
        modelo.modificaFila(e.login, e.login,
                    tablaUsuarios.getEntrada(e.login),
                    tablaConexiones.getEntrada_login(e.login));
      break;
    case 3: //eliminar
      modelo.eliminaFila(e.login);
      int pos;

      if ( (pos = tablaConexiones.buscaLogin(e.login)) <
          tablaConexiones.MAX_CONEXIONES) {
```

```
        if (tablaConexiones.getEntrada(pos).getTipo() ==
            TablaConexiones.actuador)
          servidor.desconectarTCP_GUI(ServidorRed.actuador);
        else
        if (tablaConexiones.getEntrada(pos).getTipo() ==
            TablaConexiones.udp)
          servidor.desconectarUDP_GUI(tablaConexiones.getEntrada(pos));
      }
    }
  }

  public void recibido(EventoConexion e) {
    switch(e.suceso){
      case 0://nueva entrada
      case 2://modificación
        if (!e.entrada.getLogin().equals("ADMIN")){
          if (!modelo.modificaFila(e.entrada.getLogin(), e.entrada.getLogin(),
                                   null, e.entrada))
            dialogo("Error de sincronización en la tabla de conexiones");
        }
        break;
      case 1://eliminacion
        if (!e.entrada.getLogin().equals("ADMIN")){
          if (!modelo.desconectaUsuario(e.entrada)){
            if (e.entrada.getTipo() != TablaConexiones.admin)
              dialogo("Error de sincronización en la tabla de conexiones");
          }
        }
        break;
    }
  }

  //funcion llamada cuando se quita este panel
```

```
  public void retirar() {
    if (dialogo != null)
      if (dialogo.isVisible())
        dialogo.setVisible(false);
  }

  private void inicializaTabla(JTable tabla){
   Iterator iterador = tablaUsuarios.getIterator();

    while (iterador.hasNext()) {
          String login = (String) iterador.next();
          modelo.anadirFila(login, tablaUsuarios.getEntrada(login),
          tablaConexiones.getEntrada_login(login));
        }
  }

  private void setColumnWidth(JTable tabla){
    tabla.getColumnModel().getColumn(0).setPreferredWidth(75);
    tabla.getColumnModel().getColumn(1).setPreferredWidth(75);
    tabla.getColumnModel().getColumn(2).setPreferredWidth(80);
    tabla.getColumnModel().getColumn(3).setPreferredWidth(50);//activo
    tabla.getColumnModel().getColumn(4).setPreferredWidth(80);
    tabla.getColumnModel().getColumn(5).setPreferredWidth(110);
    tabla.getColumnModel().getColumn(6).setPreferredWidth(100);
}

  private Component buscaComponente(JPanel panel,String nombre){
  try {
    Component[] componentes = panel.getComponents();

    for (int i = 0; i < panel.getComponentCount(); i++) {
      if (componentes[i].getName() != null)
        if (componentes[i].getName().equals(nombre))
          return componentes[i];
```

```
    }
    return null;
  }
  catch (NullPointerException e) {
    return null;
    }
  }

  private void dialogo(String texto){
    JOptionPane.showMessageDialog(this,texto);
  }

  private JSplitPane foo(){
    return this;
  }

  private void addGB(JPanel panel,Component component,
                     GridBagConstraints constraints,int x, int y){
      constraints.gridx = x;
      constraints.gridy = y;
      panel.add(component,constraints);
  }

  private synchronized void setFilaSeleccionada(int fila){
    filaSeleccionada = fila;
  }

  private synchronized int getFilaSeleccionada(){
    return filaSeleccionada;
  }
}
```

DialogoUsuario.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

public class DialogoUsuario extends JFrame{
  private GridBagConstraints constraints;
  private Container contenido;
  private TablaUsuarios usuarios;
  private TablaConexiones conexiones;
  private JButton aceptar;
  private Vector vector;

  public DialogoUsuario(String titulo,TablaUsuarios tablaUsuarios,
   TablaConexiones tablaConexiones,Vector entradaModificar){
    super(titulo);
    setSize(300,250);
    this.setResizable(false);
    this.vector = entradaModificar;

    this.usuarios = tablaUsuarios;
    this.conexiones = tablaConexiones;

    contenido = getContentPane();
    contenido.setLayout(new GridBagLayout());
    constraints = new GridBagConstraints();

    JTextField login;
    JTextField nombre;
    JPasswordField password;
    JPasswordField repetirPassword;
    JComboBox comboOpciones;
```

```
JPanel panelAceptar;
JButton cancelar;

login = new JTextField(1);
login.setName("login");

nombre = new JTextField(1);
nombre.setName("nombre");
password = new JPasswordField(1);
password.setName("password");

repetirPassword = new JPasswordField(1);
repetirPassword.setName("repassword");

comboOpciones = new JComboBox();
comboOpciones.setEditable(false);
comboOpciones.setName("lista");
comboOpciones.addItem("Espectador");
comboOpciones.addItem("Actuador");
comboOpciones.addItem("Administrador");
comboOpciones.setSelectedIndex(0);

aceptar = new JButton("Aceptar");
aceptar.setActionCommand("aceptar");
aceptar.setName("aceptar");
aceptar.setEnabled(true);

cancelar = new JButton("Cancelar");
cancelar.setActionCommand("cancelar");
cancelar.setName("cancelar");
cancelar.setEnabled(true);

panelAceptar = new JPanel();
panelAceptar.setName("panelaceptar");
```

```
    panelAceptar.add(aceptar);
    panelAceptar.add(cancelar);

    addGB(new JLabel("Login:"),0,0);
    constraints.ipadx = 99;
    addGB(login,0,1);
    constraints.ipadx = 0;
    addGB(new JLabel("Permiso:"),1,0);
    addGB(comboOpciones,1,1);
    addGB(new JLabel("Nombre:"),0,2);
    constraints.ipadx = 99;
    addGB(nombre,0,3);
    constraints.ipadx = 0;
    addGB(new JLabel("Password:"),0,4);
    constraints.ipadx = 99;
    addGB(password,0,5);
    constraints.ipadx = 0;
    addGB(new JLabel("Repita el Password:"),0,6);
    constraints.ipadx = 99;
    addGB(repetirPassword,0,7);
    constraints.ipadx = 0;
    addGB(panelAceptar,1,9);

     repetirPassword.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ev){
        JPasswordField password;
        JPasswordField repassword;

        password = (JPasswordField)buscaComponenteAqui("password");
        repassword = (JPasswordField)ev.getSource();
        if (repassword.getPassword().length > 0){
         if
(!comparaPassword(password.getPassword(),repassword.getPassword())){
              repassword.setText("");
```

```
            password.setText("");
            password.requestFocus();
            dialogo("Los campos de password no " +
                                "coinciden, intentelo de nuevo");
         }
        }
       }
     });
//listener de Aceptar
    aceptar.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ev){
        JPasswordField password;
        JPasswordField repassword;
        JTextField login;

        password = (JPasswordField) buscaComponenteAqui("password");
        repassword = (JPasswordField) buscaComponenteAqui("repassword");
        login = (JTextField) buscaComponenteAqui("login");


        if (login.getText().length() == 0) {
          dialogo("Debe rellenar el campo de Login");
            login.requestFocus();
          return;
        }

        if (login.getText().length() > 13) {
          dialogo("El tamaño máximo para el campo de Login es de 13 caracteres");
            login.requestFocus();
          return;
        }

        if (password.getPassword().length == 0){
          dialogo("Introduzca un password");
```

```
  return;
}

if (!comparaPassword(password.getPassword(), //los campos NO coinciden
                repassword.getPassword())) {
 dialogo("Los campos de password no " +"coinciden, intentelo de nuevo");
 repassword.setText("");
 password.setText("");
 password.requestFocus();
 return;
}

if (login.getText().indexOf('#') >= 0) {// hay una '#'
  dialogo("El caracter '#' no está permitido en el campo " +
        "de login");
  login.setText("");
  login.requestFocus();
  return;
}

if (login.getText().equals("ADMIN")) {
  dialogo("\"ADMIN\" no es un Login permitido");
  login.setText("");
  login.requestFocus();
  return;
}

//YA se han verificado las situaciones anormales
short tipo = (short) ( ( (JComboBox) buscaComponenteAqui(
      "lista")).getSelectedIndex());


if (vector == null){
  if (anadeUsuario(login, password, tipo))
```

```
          setVisible(false);
        else {
          dialogo("No se pudo añadir el " +
                "usuario, ya que existe otro con el mismo login");
          login.setText("");
          login.requestFocus();
          }
        }
       else{
        if (modificaUsuario(login,password,tipo))
          setVisible(false);
        else{
          dialogo("No es posible cambiar el campo de login, " +
                "ya que existe otro usuario con ese login");
          login.setText("");
          login.requestFocus();
          }
        }
    }});

    if (vector != null)
    dialogoUsuarioModificar();

   cancelar.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ev) {
      setVisible(false);
    }
   });
}

 private void dialogoUsuarioModificar(){

   entradaJTable entrada = new entradaJTable(vector);
   entradaUsuario eUsuario = usuarios.getEntrada(entrada.getLogin());
```

```
   //inicializamos el dialogo con los valores de la tabla
   ((JTextField)buscaComponenteAqui("login")).setText(entrada.getLogin());
   ((JTextField)buscaComponenteAqui("nombre")).setText(entrada.getNombre());
   ((JPasswordField)buscaComponenteAqui("password")).setText(
      eUsuario.getPassword());
   ((JPasswordField)buscaComponenteAqui("repassword")).setText(
      eUsuario.getPassword());
   ((JComboBox)buscaComponenteAqui("lista")).setSelectedItem(entrada.
      getPermiso());
 }

 private void addGB(Component component,int x, int y){
  constraints.gridx = x;
  constraints.gridy = y;
  contenido.add(component,constraints);
 }

 private boolean anadeUsuario(JTextField login, JPasswordField password,
                              short tipo){
   return usuarios.anadirUsuario(login.getText(),
                        ((JTextField) buscaComponenteAqui("nombre")).
                        getText(), new String(password.getPassword()),tipo);
 }

 private boolean modificaUsuario(JTextField login, JPasswordField password,
                              short tipo){

   entradaJTable entrada = new entradaJTable(vector);
   entradaUsuario eUsuario;
   String alias =login.getText();
   String temp1 =entrada.getLogin();

   try{
      if (usuarios.setLogin(entrada.getLogin(),login.getText()))
```

```
        conexiones.setLogin(temp1,alias);

     usuarios.setNombre(alias,((JTextField)buscaComponenteAqui("nombre"))
                    .getText());
     usuarios.setPassword(alias, new String(password.getPassword()));
     usuarios.setPermisos(alias,tipo);
     return true;
  }
  catch(NullPointerException e){
    return false;
  }
}

private Component buscaComponenteAqui(String nombre){
  try {
    Component[] componentes = contenido.getComponents();

    for (int i = 0; i < contenido.getComponentCount(); i++) {
      if (componentes[i].getName() != null)
        if (componentes[i].getName().equals(nombre))
          return componentes[i];
    }
    return null; //no se ha encontrado
  }
  catch (NullPointerException e) {
    return null;
  }
}

private Component buscaComponente(JPanel panel,String nombre){
  try {
    Component[] componentes = panel.getComponents();

    for (int i = 0; i < panel.getComponentCount(); i++) {
```

```
        if (componentes[i].getName() != null)
          if (componentes[i].getName().equals(nombre))
            return componentes[i];
      }
      return null; //no se ha encontrado
    }
    catch (NullPointerException e) {
      return null;
    }
  }

  private boolean comparaPassword(char[] password1, char[] password2){
    if (password1.length != password2.length)
      return false;

    int i = 0;
    while ((i < password1.length) && (password1[i] == password2[i])){
      i++;
    }

    if (i == password1.length)
      return true;
    else
      return false;
  }

  private void dialogo(String texto){
    JOptionPane.showMessageDialog(this,texto);
  }
}
```

FrameServidor.java

```
import java.io.*;
import java.text.*;
import java.util.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class FrameServidor
    extends JFrame
    implements PanelLoginListener,
    ServidorListener, MotorMovidoListener,PanelMovimientoListener {
  private ServidorRed servidor;

  private OpcionesMotores tablaMotores;
  private OpcionesRed tablaRed;
  private TablaUsuarios tablaUsuarios;
  private TablaConexiones tablaConexiones;
  private Container contenido;
  private ToUSB usb;
  private short contador;

  private JTextArea avisos;

  private JSplitPane panelCentral;

  private boolean conectado;

  public FrameServidor(String titulo, ServidorRed servidor,
                       OpcionesMotores tablaMotores, OpcionesRed tablaRed,
```

```
                TablaUsuarios tablaUsuarios,
                TablaConexiones tablaConexiones,
                ToUSB usb) {

    super(titulo);
    this.servidor = servidor;

    this.tablaMotores = tablaMotores;
    this.tablaRed = tablaRed;
    this.tablaUsuarios = tablaUsuarios;
    this.tablaConexiones = tablaConexiones;
    this.usb = usb;
    this.usb.addMotorMovidoListener(this);
    this.conectado = false;

    panelCentral = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
    panelCentral.setDividerLocation(panelCentral.getMaximumDividerLocation());
    panelCentral.setDividerSize( -1);
    panelCentral.setRightComponent(null);

    Border borde = BorderFactory.createEtchedBorder();
    panelCentral.setBorder(BorderFactory.createTitledBorder(borde));

    PanelLoginServidor panel = new PanelLoginServidor(tablaUsuarios,
       tablaConexiones);
    panel.addPanelLoginListener(this);
    panelCentral.setLeftComponent(panel);

    contenido = this.getContentPane();
    contenido.add(barraSuperior(), BorderLayout.NORTH);
    contenido.add(panelCentral, BorderLayout.CENTER);
    contenido.add(barraInferior(), BorderLayout.SOUTH);

    setResizable(false); //no se puede modificar su tamaño
```

```
  setSize(690,270);
 }

 private JPanel barraSuperior() {
  JPanel panel = new JPanel();
  panel.setLayout(new FlowLayout());
  panel.setName("barrasuperior");

  JButton login;
  JButton movimiento;
  JButton configuracion;
  JButton usuarios;
  JButton ayuda;
  JButton salir;

  login = new JButton("Login", crearIcono("iconos/login.gif"));
  login.setActionCommand("login");
  login.setName("login");
  login.setEnabled(false);
  panel.add(login);

  movimiento = new JButton("Movimiento", crearIcono("iconos/movimiento.gif"));
  movimiento.setActionCommand("movimiento");
  movimiento.setName("movimiento");
  movimiento.setEnabled(false);
  panel.add(movimiento);

  configuracion = new JButton("Configuración",
                  crearIcono("iconos/configuracion.gif"));
  configuracion.setActionCommand("configuracion");
  configuracion.setName("configuracion");
  configuracion.setEnabled(false);
  panel.add(configuracion);
```

```
usuarios = new JButton("Usuarios", crearIcono("iconos/usuarios.gif"));
usuarios.setActionCommand("usuarios");
usuarios.setName("usuarios");
usuarios.setEnabled(false);
panel.add(usuarios);

ayuda = new JButton("Ayuda", crearIcono("iconos/ayuda.gif"));
ayuda.setActionCommand("ayuda");
ayuda.setName("ayuda");
panel.add(ayuda);

salir = new JButton("", crearIcono("iconos/apagar.gif"));
salir.setActionCommand("salir");
salir.setName("salir");
salir.setToolTipText("Salir");
panel.add(salir);

/***Listeners*/
login.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ev) {
    Component componente = buscaComponente(panelCentral, "panel central");

    if (! (componente instanceof PanelLoginServidor)) {
      if (componente instanceof PanelUsuarios) {
        ( (PanelUsuarios) componente).retirar();
      }

      if ( ( (JButton) ev.getSource()).getText().equals("Login")) {
        PanelLoginServidor panel = new PanelLoginServidor(
            tablaUsuarios, tablaConexiones);
        panel.addPanelLoginListener(getPanelLoginListener());
        panelCentral.setLeftComponent(panel);
      }
      else {
```

```
        int result = JOptionPane.showConfirmDialog(getComponent(),
            "¿Desea abandonar la sesión?", "USBMotor",
              JOptionPane.YES_NO_OPTION);

        if (result == JOptionPane.YES_OPTION) {
          if (tablaConexiones.eliminarConexionAdmin()) {
            JOptionPane.showMessageDialog(null, "Ha sido desconectado " +
                                  "satisfactoriamente");
            login(false);
            anadeAviso("Logout");
            PanelLoginServidor panel = new PanelLoginServidor(
               tablaUsuarios, tablaConexiones);
            panel.addPanelLoginListener(getPanelLoginListener());
            panelCentral.setLeftComponent(panel);
          }
          else {
            JOptionPane.showMessageDialog(null,
                                  "Se ha producido un error, " +
                                  "no es posible realizar su desconexion");
          }
        }
      }
    }
  }
});

movimiento.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ev) {
    JPanel panel = (JPanel) buscaComponente(contenido, "barrasuperior");
    JButton botonLogin = (JButton) buscaComponente(panel, "login");
    botonLogin.setEnabled(true);


    Component componente = buscaComponente(panelCentral, "panel central");
```

```
if (! (componente instanceof PanelMovimiento)) {
  if (componente instanceof PanelUsuarios) {
    ((PanelUsuarios) componente).retirar();
  }
  else{
    if(componente instanceof PanelLoginServidor){
      setVisible(false);
      setSize(690,650);
      setVisible(true);
    }
  }

  double anguloMaximo1 = 0;
  double anguloMaximo2 = 0;

  if (tablaMotores.getFlag(1) == 2) {
    anguloMaximo1 = tablaMotores.getAnguloMaximo(1);
  }
  if (tablaMotores.getFlag(2) == 2) {
    anguloMaximo2 = tablaMotores.getAnguloMaximo(2);
  }

  PanelMovimiento movimiento = new PanelMovimiento(
                        (int) (360 / tablaMotores.getAnguloPaso(1)),
                       (int) (360 / tablaMotores.getAnguloPaso(2)),
                     tablaMotores.getPaso(1),
                    tablaMotores.getPaso(2),
                   tablaMotores.getVelocidad(1),
                  tablaMotores.getVelocidad(2),
                 anguloMaximo1, anguloMaximo2,
                tablaMotores.getVelocidadMaxima(1),
               tablaMotores.getVelocidadMaxima(2),
              (tablaConexiones.buscaLogin("ADMIN") <
```

```
            tablaConexiones.MAX_CONEXIONES) ? true : false);
        movimiento.addPanelMovimientoListener(foo());


        panelCentral.setLeftComponent(movimiento);

        if (tablaConexiones.buscaLogin("ADMIN") <
            tablaConexiones.MAX_CONEXIONES) {
          ( (PanelMovimiento) panelCentral.getLeftComponent()).
            addPanelMovimientoListener(usb);
        }
      }
    }
});

configuracion.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ev) {
    JPanel panel = (JPanel) buscaComponente(contenido, "barrasuperior");
    JButton botonLogin = (JButton) buscaComponente(panel, "login");
    botonLogin.setEnabled(true);

    Component componente = buscaComponente(panelCentral, "panel central");

    if (! (componente instanceof PanelOpciones)) {
      if (componente instanceof PanelUsuarios) {
        ( (PanelUsuarios) componente).retirar();
      }
      else{
        if(componente instanceof PanelLoginServidor){
          setVisible(false);
          setSize(690,650);
          setVisible(true);
        }
      }
```

```
        panelCentral.setLeftComponent(new PanelOpciones(
            tablaMotores,
            tablaRed,
            conectado));
      }
    }
  });

  usuarios.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
      JPanel panel = (JPanel) buscaComponente(contenido, "barrasuperior");
      JButton botonLogin = (JButton) buscaComponente(panel, "login");
      botonLogin.setEnabled(true);

      Component componente = buscaComponente(panelCentral, "panel central");

      if (! (componente instanceof PanelUsuarios)) {
        if(componente instanceof PanelLoginServidor){
          setVisible(false);
          setSize(690,650);
          setVisible(true);
        }

        panelCentral.setLeftComponent(new PanelUsuarios(tablaUsuarios,
            tablaConexiones,
            servidor));
      }
    }
  });

  ayuda.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
      PanelAyuda ayuda = new PanelAyuda(false,null);
```

```
    }
  });

  salir.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
      tablaMotores.close();
      tablaUsuarios.guardarTabla("usuarios.inf");
      usb.close();
      if (servidor != null)
        servidor.close();
      System.exit(0);
    }
  });
  return panel;
}

private JPanel barraInferior() {
  JPanel panel = new JPanel();
  panel.setName("barrainferior");

  panel.setLayout(new FlowLayout());

  JButton activaRed;
  JButton controlaMovimiento;
  JScrollPane panelAvisos;

  activaRed = new JButton("  Activar Servidor",
                    crearIcono("iconos/conectar.gif"));
  activaRed.setActionCommand("activarservidor");
  activaRed.setEnabled(false);
  activaRed.setName("botonservidor");

  controlaMovimiento = new JButton("  Control Local",
                          crearIcono("iconos/controlLocal.gif"));
```

```
controlaMovimiento.setActionCommand("controllocal");
controlaMovimiento.setEnabled(false);
controlaMovimiento.setName("botoncontrolamovimiento");

avisos = new JTextArea();
avisos.setEditable(false);

panelAvisos = new JScrollPane(avisos);
panelAvisos.setWheelScrollingEnabled(true);
panelAvisos.setHorizontalScrollBarPolicy(JScrollPane.
                                HORIZONTAL_SCROLLBAR_ALWAYS);
panelAvisos.setVerticalScrollBarPolicy(JScrollPane.
                               VERTICAL_SCROLLBAR_ALWAYS);
panelAvisos.setPreferredSize(new Dimension(340, 60));

panel.add(activaRed);
panel.add(panelAvisos);
panel.add(controlaMovimiento);

/***Listeners*/
activaRed.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ev) {
    JButton boton = (JButton) ev.getSource();

    if (boton.getActionCommand().equals("activarservidor")) {

      File directorio = new File(tablaRed.getRuta());
      if (!directorio.isDirectory()) {
        JOptionPane.showMessageDialog(esto(), "No es posible iniciar el"
            + " servidor : la ubicación del applet indicada no es correcta");
      }

      else {
        try {
```

```
      servidor = new ServidorRed(tablaUsuarios,
                                 tablaConexiones,
                                 tablaRed.getTimeoutActuador(),
                                 tablaRed.getTiempoConexionActuadorLong(),
                                 tablaRed.getTiempoConexionGestionLong(),
                                 tablaRed.getPuertoGestion(),
                                 tablaRed.getTimeoutEspectador(),
                                 (int) tablaRed.getTimeoutEspectador(),
                                 tablaMotores,
                                 getServidorListener());

      estadoBotonServidor(boton, true);
      if (tablaConexiones.buscaLogin("ADMIN") ==
          tablaConexiones.MAX_CONEXIONES) {
        servidor.addMotorListener(usb);
      }
      usb.addMotorMovidoListener(servidor);

      if (!tablaRed.copiaParaApplet())
        JOptionPane.showMessageDialog(esto(),"Error al crear la "
                                      + "configuración de red para applets");
    }
    catch (IOException e) {
      dialogo("Error en la inicializacion del servidor");
    }
  }
}

else {
  usb.removeMotorMovidoListener(servidor);
  servidor.close();
  servidor = null;
  estadoBotonServidor(boton, false);
}
```

```
    }
  });

  //movimiento local ó remoto
  controlaMovimiento.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
      JButton boton = (JButton) ev.getSource();
      Component panel = panelCentral.getLeftComponent();

      if (boton.getActionCommand().equals("controllocal")) {
        if (servidor != null) {
          servidor.permitirActuador(false);
          servidor.removeMotorListener(usb);
        }
        else {
          tablaConexiones.crearActuadorFicticio();
        }
        estadoBotonControl(boton, true);

        if (panel instanceof PanelMovimiento) {
          ( (PanelMovimiento) panel).setPermisoMovimiento(true);
          ( (PanelMovimiento) panel).addPanelMovimientoListener(usb);
        }
      }
      else {
        if (servidor != null) {
          servidor.permitirActuador(true);
          servidor.addMotorListener(usb);
        }
        else {
          tablaConexiones.eliminarActuadorFicticio();
        }
        if (panel instanceof PanelMovimiento) {
          ( (PanelMovimiento) panel).setPermisoMovimiento(false);
```

```
            ( (PanelMovimiento) panel).removePanelMovimientoListener(usb);
          }

          estadoBotonControl(boton, false);
        }
      }
    });
    return panel;
  }
  private PanelLoginListener getPanelLoginListener() {
    return this;
  }

  private ServidorListener getServidorListener() {
    return this;
  }

  private Component getComponent() {
    return contenido;
  }

//----------------------------LISTENERS LOGIN
  public void loged(PanelLoginEvent ev) {
    if (ev.permiso == (TablaUsuarios.admin + TablaUsuarios.actuador +
                       TablaUsuarios.udp)) {
      login(false);
      anadeAviso("Logout");
    }
    else {
      login(true);
      anadeAviso("Login : " + ev.login);
    }
  }
```

```
private void login(boolean loged) {
  JPanel panel;
  JButton boton;

  //empezamos por la barra superior
  panel = (JPanel) buscaComponente(contenido, "barrasuperior");

  boton = (JButton) buscaComponente(panel, "login");
  if (loged) {
    boton.setText(" Logout");
  }
  else {
    boton.setText("  Login");
    boton.setEnabled(false);

    setVisible(false);
    setSize(690,270);
    setVisible(true);
  }
  boton = (JButton) buscaComponente(panel, "movimiento");
  boton.setEnabled(loged);

  boton = (JButton) buscaComponente(panel, "configuracion");
  boton.setEnabled(loged);

  boton = (JButton) buscaComponente(panel, "usuarios");
  boton.setEnabled(loged);

  //lo mismo, pero para la barra inferior
  panel = (JPanel) buscaComponente(contenido, "barrainferior");

  boton = (JButton) buscaComponente(panel, "botonservidor");
  boton.setEnabled(loged);
```

```
    boton = (JButton) buscaComponente(panel, "botoncontrolamovimiento");
    boton.setEnabled(loged);
  }

//-------------------------REPRESENTAR EN EL AREA DE EVENTOS

  public synchronized void anadeAviso(String mensaje) {
    DateFormat fecha = DateFormat.getInstance();

    avisos.append(fecha.format(new Date()) + " : " + mensaje + "\n");
  }

//-------------------------LISTENERS EVENTOS SERVIDOR
  public void recibido(EventoServidor ev) {
    String mensaje;
    PanelOpciones panelOpciones;

    if (ev.ID == 4) {
      mensaje = new String("Servidor Activado");
      anadeAviso(mensaje);
      this.conectado = true;

      if (panelCentral.getLeftComponent() instanceof PanelOpciones) {
        panelOpciones = (PanelOpciones) panelCentral.getLeftComponent();
        panelOpciones.conectado(true);
      }
      return;
    }

    if (ev.ID == 5) { //desconexion del servidor
      anadeAviso("Servidor Cerrado");
      this.conectado = false;
      if (panelCentral.getLeftComponent() instanceof PanelOpciones) {
        panelOpciones = (PanelOpciones) panelCentral.getLeftComponent();
```

```
        panelOpciones.conectado(false);
      }
      return;
    }

    switch (ev.ID) {
      case 0: //gestora
        mensaje = new String("Error en la conexion Gestora : ");
        break;
      case 1: //actuadora
        mensaje = new String("Error en la conexion Actuadora : ");
        break;
      default: //espectador
        mensaje = new String("Error en la conexion Espectadora : ");
        break;
    }

    switch (ev.suceso) {
      case 0:
        mensaje = mensaje + "Desconexion por Timeout";
        break;
      case 1:
        mensaje = mensaje + "Desconexion por error E/S";
        break;
      case 2:
        mensaje = mensaje + "Desconexion por error en el protocolo";
        break;
    }
    anadeAviso(mensaje);
  }

//-------------LISTENER DEL PANEL DE MOVIMIENTO (solicitudes de
movimiento)----
  public void moverMotor(PanelMovimientoEvent ev){
```

```
    if (ev.ID == 0)
      contador = 2;
    else
      contador = 1;

    anadeAviso("Acción solicitada ...");
  }

//----------------------LISTENER MOVIMIENTOS DEL MOTOR
  public void motorMovido(MotorMovidoEvent ev) { //indica que ya se ha movido
    Component panel = panelCentral.getLeftComponent();

    if ((ev.tipo == 1) || (ev.tipo == 2)){
      if (panel instanceof PanelMovimiento) {
        ((PanelMovimiento) panel).setDatos(tablaMotores.getPaso(1),
                                    tablaMotores.getPaso(2),
                                    tablaMotores.getVelocidad(1),
                                    tablaMotores.getVelocidad(2),
              ((tablaConexiones.buscaLogin("ADMIN") ==
              tablaConexiones.MAX_CONEXIONES) ? false : true));

        if (ev.tipo == 2){
          anadeAviso("No ha sido posible mover correctamente los motores");
          if (tablaConexiones.buscaLogin("ADMIN") ==
                tablaConexiones.MAX_CONEXIONES) {
             dialogo("No ha sido posible mover correctamente los motores");
           }
        }
        else
          anadeAviso("Movimiento ejecutado correctamente");
      }

      //escribimos en el log
      tablaMotores.movido(tablaMotores.getAnguloActual(1),
```

```
                    tablaMotores.getVelocidad(1),
                    tablaMotores.getAnguloActual(2),
                    tablaMotores.getVelocidad(2),
                    tablaConexiones.getEntrada_tipo(
        TablaConexiones.actuador, 0).getLogin());
  }
  else
  {
   if (panel instanceof PanelMovimiento)
    ((PanelMovimiento) panel).falloMover();

    switch(ev.tipo){
      case 3: //error al INICIAR el pipe
       if (tablaConexiones.buscaLogin("ADMIN") !=
            tablaConexiones.MAX_CONEXIONES) {
          dialogo("Error en la comunicación con la aplicación " +
                "controladora de USB");
        }
       anadeAviso("Error en la comunicación con la aplicación " +
              "controladora de USB");
       break;
      case 4: //error en la aplicacion USB
       if (tablaConexiones.buscaLogin("ADMIN") !=
          tablaConexiones.MAX_CONEXIONES) {
        dialogo("Error en la aplicación controladora de USB");
       }
       anadeAviso("Error en la aplicación controladora de USB");
       break;
      case 5: //error al acceder al dispositivo
       if (tablaConexiones.buscaLogin("ADMIN") !=
          tablaConexiones.MAX_CONEXIONES) {
        dialogo("Error al acceder al dispositivo : Ha sido desconectado");
       }
       anadeAviso("Error al acceder al dispositivo : Ha sido desconectado");
```

```
            break;
        }
      }

      contador--;
    }

//--------------------------------VARIOS
    private void estadoBotonServidor(JButton boton, boolean conectado) {
      if (conectado == true) {
        boton.setText("  Desactivar Servidor");
        boton.setActionCommand("desactivarservidor");
        boton.setIcon(crearIcono("iconos/desconectar.gif"));
      }
      else {
        boton.setText("  Activar Servidor");
        boton.setActionCommand("activarservidor");
        boton.setIcon(crearIcono("iconos/conectar.gif"));
      }
    }

    private void estadoBotonControl(JButton boton, boolean local) {
      if (local == true) {
        boton.setText("  Control Remoto");
        boton.setActionCommand("controlremoto");
        boton.setIcon(crearIcono("iconos/controlRemoto.gif"));
      }
      else {
        boton.setText("  Control Local");
        boton.setActionCommand("controllocal");
        boton.setIcon(crearIcono("iconos/controlLocal.gif"));
      }
    }
```

```
public void actualizaComponenteCentral(JComponent componenteCentral) {
  panelCentral.removeAll();
  panelCentral.add(componenteCentral);
}

private ImageIcon crearIcono(String path) {
  return new ImageIcon(path);
}

private void dialogo(String texto) {
  JOptionPane.showMessageDialog(this, texto);
}

private Component buscaComponente(Container panel, String nombre) {
  try {
    Component[] componentes = panel.getComponents();

    for (int i = 0; i < panel.getComponentCount(); i++) {
      if (componentes[i].getName() != null) {
        if (componentes[i].getName().equals(nombre)) {
          return componentes[i];
        }
      }
    }
    return null;
  }
  catch (NullPointerException e) {
    return null;
  }
}

private PanelMovimientoListener foo(){
  return this;
}
```

```
  private JFrame esto(){
    return this;
  }
}
```

principal.java

```
import javax.swing.*;
import java.awt.event.*;

public class principal{
  public principal() {}
    private static TablaUsuarios tablaUsuarios;
    private static ServidorRed servidor = null;
    private static FrameServidor frame;

  public static void main(String[] args) {
      tablaUsuarios = new TablaUsuarios("usuarios.inf");
      TablaConexiones tablaConexiones = new TablaConexiones(3);

      OpcionesRed tablaRed = new OpcionesRed();
      final OpcionesMotores tablaMotores= new OpcionesMotores();

      final ToUSB usb = new ToUSB(tablaMotores);

      frame = new FrameServidor("USBMotor",
                                servidor,
                                tablaMotores,
                                tablaRed,
                                tablaUsuarios,
                                tablaConexiones,
                                usb);
```

```
      frame.addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent ev) {
         try {
           tablaMotores.close();
           tablaUsuarios.guardarTabla("usuarios.inf");
           usb.close();
           servidor.close();
         }
         catch (NullPointerException e) {}
         finally{
           System.exit(0);
         }
       }

       });

     try{

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAnd
Feel");
       SwingUtilities.updateComponentTreeUI(frame);
     }
     catch(Exception e){}

     frame.setVisible(true);
  }
}
```

**********GESTION DE RED**********

ClienteRed.java

```
import java.net.*;
import java.io.*;
import java.util.*;

class ClienteRed implements PaqueteTCPListener, PaqueteUDPListener,
   PanelMovimientoListener{
  final short gestor = 0;
  final short actuador = 1;
  final short admin = 2;
  private int TCPtimeout;
  private int UDPtimeout;
  private int puerto;
  private Vector listeners;
  private InetAddress direccion_servidor;

  private PDU_TCP[] arrayTCP;     /**array de dos posiciones:
                          /* 0: conexion gestora
                          /* 1: conexion actuadora/administrador*/
  private PDU_UDP ultimaUDP;
  private PaqueteTCPCliente[] tcp;  /**array de dos posiciones:
                          /* 0: conexion gestora
                          /* 1: conexion actuadora*/

  private PaqueteUDPCliente udp;
  String login;
  String password;

  DatosMotoresCliente tablaMotores;

  short tipo_conexion;
```

```
  public ClienteRed(InetAddress direccion_servidor,int TCPtimeout, int
UDPtimeout,
                int puerto_gestion, String login,
                String password, short tipo_conexion,
                DatosMotoresCliente tablaMotores, ClienteListener l)
     throws IOException {

    this.direccion_servidor = direccion_servidor;
    this.TCPtimeout = TCPtimeout;
    this.UDPtimeout = UDPtimeout;
    this.puerto = puerto_gestion;
    this.login = login;
    this.password = password;
    this.tipo_conexion = tipo_conexion;

    this.tablaMotores = tablaMotores;

    this.addClienteListener(l);

    tcp = new PaqueteTCPCliente[2];
    arrayTCP = new PDU_TCP[2];
    ultimaUDP = null;

    tcp[gestor] = new PaqueteTCPCliente(puerto_gestion,
                        direccion_servidor,
                        gestor,
                        TCPtimeout,
                        this);

    tcp[actuador] = null;
    }

    public void close() {
      if (tcp[gestor] != null){
```

```
    PDU_TCP pdu = new PDU_TCP();
    pdu.setTipo(5,0);
    desconectarTCP(gestor,pdu);
   }

   if (tcp[actuador] != null){
    PDU_TCP pdu = new PDU_TCP();
    pdu.setTipo(5,0);
    desconectarTCP(actuador,pdu);
   }

   if (udp != null){
    PDU_UDP pdu = new PDU_UDP();
    pdu.setTipo(5,0);
    udp.enviar(pdu,direccion_servidor);
    udp.close();
   }
  }

  public boolean enviarTCP(short conexionID,PDU_TCP pdu){
   try{
    return tcp[conexionID].enviar(pdu);
   }
   catch(NullPointerException e){return false;}
   }

  public boolean enviarUDP(PDU_UDP pdu){
   return udp.enviar(pdu,direccion_servidor);
  }

  public boolean mover_motor(int angulo_motor1,int angulo_motor2,
                             int velocidad1, int velocidad2){
    PDU_TCP pdu = new PDU_TCP();
```

```
      pdu.setTipo(2, 2);
      pdu.addDato(angulo_motor1);
      pdu.addDato(velocidad1);
      pdu.addDato(angulo_motor2);
      pdu.addDato(velocidad2);
      return enviarTCP(actuador,pdu);
    }

 public void recibido(EventoPaqueteTCP paquete){
   PDU_TCP pdu = new PDU_TCP();

   try{
   switch (paquete.suceso){
     case 0://recibido correctamente
       switch(paquete.pdu.getTipo()){
         case 3:
           switch(paquete.pdu.getSubtipo()){
             case 0:
               pdu.setTipo(3,1);
               pdu.addDato("Hola, soy el applet del PFC");
               tcp[paquete.id].enviar(pdu);
               break;
             case 1:
               if ((arrayTCP[paquete.id].getTipo() == 3) &&
                 (arrayTCP[paquete.id].getSubtipo() == 0))
                   arrayTCP[paquete.id]=null;
               break;
             case 3:
               avisaCliente(paquete.id,(short)2);
               break;
           }
           break;
         case 4:
           if (paquete.pdu.getSubtipo() == 2){
```

```
    tcp[actuador].close();
    tcp[actuador] = null;
    udp = new PaqueteUDPCliente(paquete.pdu.getIntcampo(0),
                                this.UDPtimeout,
                                this);

    desconectarTCP(actuador,null);
    avisaCliente( (short) 1, (short)3);
   }
   break;
  case 5:
   switch(paquete.pdu.getSubtipo()){
    case 0:
       desconectarTCP(paquete.id,null);
       avisaCliente( (short) 3, (short)8);
       break;
    case 2:
       desconectarTCP(paquete.id,null);
       avisaCliente( (short) 3, (short)1);
    default:
       mensajesActuador(paquete.pdu);
    break;
   }
   break;
  default:
   switch (paquete.id) {
    case gestor:
      mensajesGestor(paquete.pdu);
      break;
    case actuador:
      mensajesActuador(paquete.pdu);
      break;
   }
   break;
```

```
    }
    break;

  case 1://timeout
    if (arrayTCP[paquete.id] != null){
      if ((arrayTCP[paquete.id].getTipo() == 3) &&
        (arrayTCP[paquete.id].getSubtipo() == 0)){
          pdu.setTipo(5,2);
          desconectarTCP(paquete.id, pdu);
          avisaCliente(paquete.id,(short)0);
      }
      else{
        pdu.setTipo(3,0);
        pdu.addDato("Hola, soy el applet del PFC");
        tcp[paquete.id].enviar(pdu);
        tcp[paquete.id].enviar(arrayTCP[paquete.id]);
        arrayTCP[paquete.id] = pdu;
      }
    }
    else{
      pdu.setTipo(3,0);
      pdu.addDato("Hola, soy el applet del PFC");
      tcp[paquete.id].enviar(pdu);
      arrayTCP[paquete.id] = pdu;
    }
    break;

  case 2://Error
      desconectarTCP(paquete.id,null);
      avisaCliente(paquete.id,(short)1);
    break;

  case 3:
    desconectarTCP(paquete.id, null);
```

```
      avisaCliente(paquete.id, (short) 1);
      break;

    case 7://Se ha establecido una conexion con el servidor
      switch(paquete.id){
       case gestor:
         pdu.setTipo(1,0);
         pdu.addDato(login);
         pdu.addDato(password);
         pdu.addDato(tipo_conexion);
         tcp[gestor].enviar(pdu);
         arrayTCP[gestor] = pdu;
         break;
       case actuador:
         pdu.setTipo(0,0);
         pdu.addDato(login);
         tcp[actuador].enviar(pdu);
         break;
      }
   }
  }
  catch(NullPointerException e){
    PDU_TCP pduTEMP = new PDU_TCP();
    pduTEMP.setTipo(3,3);
    tcp[paquete.id].enviar(pduTEMP);
  }
  catch(NumberFormatException e){
    pdu.setTipo(3,3);
    tcp[paquete.id].enviar(pdu);
  }
  catch(IOException e){
    avisaCliente(paquete.id,(short)1);
  }
}
```

```
private void desconectarTCP(int id,PDU_TCP motivo){
  try{
    if (motivo != null) {
      tcp[id].enviar(motivo);
      tcp[id].close();
    }

    tcp[id].close();
    tcp[id] = null;
    arrayTCP[id] = null;
  }
  catch(NullPointerException e){}
}

private void mensajesGestor(PDU_TCP pdu_recibida)
    throws NullPointerException,NumberFormatException,IOException{

  switch(pdu_recibida.getTipo()){
    case 1: //gestion del login
      switch (pdu_recibida.getSubtipo()) {
        case 1: //conexion aceptada
          arrayTCP[gestor] = null;
          switch (pdu_recibida.getIntcampo(0)) {
            case 0: //UDP
              udp = new PaqueteUDPCliente(pdu_recibida.getIntcampo(1),
                                  UDPtimeout, this);
              avisaCliente(gestor, (short) 3); //aceptado modo visor(UDP)
              break;
            case 1: //Actuador
              tcp[actuador] = new PaqueteTCPCliente(pdu_recibida.getIntcampo(
                  1),
                  direccion_servidor,
                  actuador,
                  TCPtimeout,
```

```
                this);
            avisaCliente(gestor, (short) 4); //aceptado modo actuador
            break;
          }
          tcp[gestor].close();
          break;
        case 2:
          avisaCliente(gestor, (short) 6, (short) pdu_recibida.getIntcampo(0));
          tcp[gestor].close();
          break;
      }
      break;
    case 2:
      if (pdu_recibida.getSubtipo() == 0) {
        tablaMotores.setNPasos(1, pdu_recibida.getIntcampo(0));
        tablaMotores.setNPasos(2, pdu_recibida.getIntcampo(1));

        tablaMotores.setAnguloActual(1, pdu_recibida.getDoubleCampo(2));
        tablaMotores.setAnguloActual(2, pdu_recibida.getDoubleCampo(3));

        tablaMotores.setVelocidadMaxima(1,(short)pdu_recibida.getIntcampo(4));
        tablaMotores.setVelocidadMaxima(2,(short)pdu_recibida.getIntcampo(5));

        tablaMotores.setVelocidadActual(1, pdu_recibida.getIntcampo(6));
        tablaMotores.setVelocidadActual(2, pdu_recibida.getIntcampo(7));

        tablaMotores.setFlag(1, (short) pdu_recibida.getIntcampo(8));

        int indice = 9;
        if (tablaMotores.getFlag(1) == 2) { //hay ángulo máximo
          tablaMotores.setAnguloMaximo(1, pdu_recibida.getDoubleCampo(indice));
          indice++;
        }
        else
```

```
        tablaMotores.setAnguloMaximo(1, 0);

      tablaMotores.setFlag(2, (short) pdu_recibida.getIntcampo(indice));
      indice++;

      if (tablaMotores.getFlag(2) == 2) { //hay ángulo máximo
        tablaMotores.setAnguloMaximo(2, pdu_recibida.getDoubleCampo(indice));
      }
      else
        tablaMotores.setAnguloMaximo(2, 0);

      this.avisaCliente(actuador, (short) 11);
    }
    break;
  }
}

private void mensajesActuador(PDU_TCP pdu_recibida)
  throws NullPointerException,NumberFormatException,IOException{
 PDU_TCP pdu = new PDU_TCP();

 switch(pdu_recibida.getTipo()){
  case 2:
    switch (pdu_recibida.getSubtipo()) {
      case 1:
        tablaMotores.setAnguloActual(1,pdu_recibida.getDoubleCampo(0));
        tablaMotores.setAnguloActual(2,pdu_recibida.getDoubleCampo(1));

        tablaMotores.setVelocidadActual(1,pdu_recibida.getIntcampo(2));
        tablaMotores.setVelocidadActual(2,pdu_recibida.getIntcampo(3));

        this.avisaCliente(actuador,(short)12);
        break;
```

```
    case 3://ha habido un error al mover
      this.avisaCliente(actuador,(short)13);
      break;
   }
break;

case 4:
  switch (pdu_recibida.getSubtipo()){
    case 2:
      udp = new PaqueteUDPCliente(pdu_recibida.getIntcampo(0),
                                  UDPtimeout,
                                  this);
      avisaCliente(actuador,(short)3);
      break;
    case 3:
      avisaCliente(actuador,(short)6,(short) pdu_recibida.getIntcampo(0));
      break;
  }
  break;

case 5:
  switch(pdu_recibida.getSubtipo()){
    case 1:
      desconectarTCP(actuador, null);
      udp = new PaqueteUDPCliente(pdu_recibida.getIntcampo(0),
                                  UDPtimeout,
                                  this);
      avisaCliente(actuador,(short)9);
    break;
    case 3:
      desconectarTCP(actuador, null);
      udp = new PaqueteUDPCliente(pdu_recibida.getIntcampo(0),
                        UDPtimeout,
                        this);
```

```
        avisaCliente(actuador,(short)10);
     }
    break;
   }
 }

public void recibido(EventoPaqueteUDP paquete){
  PDU_UDP pdu = new PDU_UDP();

  switch(paquete.suceso){
    case 0:
      switch (paquete.pdu.getTipo()) {
        case 2:
          if (paquete.pdu.getSubtipo() == 1)
            tablaMotores.setAnguloActual(1, paquete.pdu.getDoubleCampo(0));
            tablaMotores.setAnguloActual(2, paquete.pdu.getDoubleCampo(1));

            tablaMotores.setVelocidadActual(1, paquete.pdu.getIntcampo(2));
            tablaMotores.setVelocidadActual(2, paquete.pdu.getIntcampo(3));
            avisaCliente( (short) 3, (short) 14);
          break;

        case 3:
          switch (paquete.pdu.getSubtipo()) {
            case 0:
              pdu.setTipo(3,1);
              pdu.addDato("Hola, soy el applet del PFC");
              udp.enviar(pdu,direccion_servidor);
              ultimaUDP=null;
              break;
            case 1:
              ultimaUDP=null;
              break;
            case 2:
```

```
      avisaCliente( (short) 3, (short) 7,
                  (short) paquete.pdu.getIntcampo(0));
      break;
    case 3://error en el último paquete
      avisaCliente( (short) 3, (short)2);
  }
  break;
case 4:
  switch(paquete.pdu.getSubtipo()){
    case 2:
      try{
        tcp[actuador] = new PaqueteTCPCliente(paquete.pdu.getIntcampo(0),
                                direccion_servidor,
                                (short) (actuador),
                                TCPtimeout,
                                this);
        udp.close();
        udp = null;
        avisaCliente( (short) 3, (short)4);
      }
      catch (IOException e){
        avisaCliente( (short) 3, (short)1);
      }
      break;
    case 3:
      avisaCliente( (short) 3, (short)6,
                  (short)paquete.pdu.getIntcampo(0));
      break;
  }
  break;
case 5:
  switch(paquete.pdu.getSubtipo()){
    case 0: //cierre del programa servidor (nos hecha)
      udp.close();
```

```
            avisaCliente( (short) 3, (short)8);
            udp = null;
            break;
          case 2: //desconexion por timeout remoto (no hemos respondido)
            udp.close();
            avisaCliente( (short) 3, (short)1);
            udp = null;
            break;
        }
        break;
      }
    break;
  case 1: //timeout
    if (ultimaUDP == null)
    {
      pdu.setTipo(3, 0);
      pdu.addDato("Hola, soy el applet del PFC");
      udp.enviar(pdu, direccion_servidor);
      ultimaUDP = pdu;
    }
    else
    {
       if (ultimaUDP.getTipo() != 3)
       {
         pdu.setTipo(3, 1);
         pdu.addDato("Hola, soy el applet del PFC");
         udp.enviar(pdu, direccion_servidor);
         ultimaUDP = pdu;
       }
       else
       {
          avisaCliente( (short) 3, (short)0);
          udp.close();
       }
```

```
    }
    break;
  }
}

  public void moverMotor(PanelMovimientoEvent ev){
    PDU_TCP pdu = new PDU_TCP();

    switch(ev.ID){
      case 0: //ambos motores
        pdu.setTipo(2,1);
        pdu.addDato(ev.paso1);
        pdu.addDato(ev.paso2);
        pdu.addDato(ev.velocidad1);
        pdu.addDato(ev.velocidad2);

        this.enviarTCP(actuador,pdu);

        tablaMotores.setAnguloEnviar(1,ev.paso1);
        tablaMotores.setAnguloEnviar(2,ev.paso2);
        tablaMotores.setVelocidadEnviar(1,ev.velocidad1);
        tablaMotores.setVelocidadEnviar(2,ev.velocidad2);
        break;
      case 1: //solo motor 1
        pdu.setTipo(2,2);
        pdu.addDato(ev.paso1);
        pdu.addDato(ev.velocidad1);
        pdu.addDato(ev.ID);

        this.enviarTCP(actuador,pdu);

        tablaMotores.setAnguloEnviar(1,ev.paso1);
        tablaMotores.setVelocidadEnviar(1,ev.velocidad1);
        break;
```

```
      case 2: //solo motor 2
        pdu.setTipo(2,2);
        pdu.addDato(ev.paso2);
        pdu.addDato(ev.velocidad2);
        pdu.addDato(ev.ID);

        this.enviarTCP(actuador,pdu);

        tablaMotores.setAnguloEnviar(2,ev.paso2);
        tablaMotores.setVelocidadEnviar(2,ev.velocidad2);
        break;
    }
  }

//*****************************Control del evento*******************************
    synchronized public void addClienteListener(ClienteListener l) {
      if (listeners == null)
        listeners = new Vector();

      listeners.addElement(l);
    }

    synchronized public void removeClienteListener(ClienteListener l) {
      if (listeners == null)
        listeners = new Vector();

      listeners.removeElement(l);
    }

    protected synchronized void avisaCliente(short ID,short suceso,short motivo) {
      if (listeners != null && !listeners.isEmpty()) {
        EventoCliente evento = new EventoCliente(this,ID,suceso,motivo);
        Vector targets;
        synchronized (this) {
```

```
        targets = (Vector) listeners.clone();
      }

      Enumeration enum = targets.elements();
      while (enum.hasMoreElements()) {
        ClienteListener l = (ClienteListener) enum.nextElement();
        l.recibido(evento);
      }
    }
  }

  protected synchronized void avisaCliente(short ID,short suceso) {
    if (listeners != null && !listeners.isEmpty()) {
      EventoCliente evento = new EventoCliente(this,ID,suceso);

      Vector targets;
      synchronized (this) {
        targets = (Vector) listeners.clone();
      }

      Enumeration enum = targets.elements();
      while (enum.hasMoreElements()) {
        ClienteListener l = (ClienteListener) enum.nextElement();
        l.recibido(evento);
      }
    }
  }

}
//-----------------------PAQUETE TCP-----------------------------------
class PaqueteTCPCliente extends Thread{
  private PaqueteTCPtransmision tx;
  private PaqueteTCPrecepcion rx;
  private PaqueteTCPListener listener;
```

```
private Socket socket;
private InetAddress direccion_servidor;
private int puerto;
private int socketTimeout;
private short id;

 public PaqueteTCPCliente (int puerto, InetAddress direccion_servidor,
                    short id, int socketTimeout,
                    PaqueteTCPListener l) throws IOException{
  this.socketTimeout = socketTimeout;
  this.id = id;
  this.direccion_servidor=direccion_servidor;
  this.puerto=puerto;

  this.listener=l;
  rx = new PaqueteTCPrecepcion();

  rx.addTCPListener(listener);
  start();
}

 public synchronized void run (){
 try{
  if (socket != null)
    close();

  socket = new Socket(direccion_servidor,puerto);
  socket.setSoTimeout(socketTimeout);

  tx = new PaqueteTCPtransmision(socket);

  rx.setParam(id,socket);
  rx.start();
 }
```

```
  catch(SocketTimeoutException e){
    rx.avisaRecepcion((short) 1,id,null);
  }
  catch(SocketException e){
    rx.avisaRecepcion((short) 3,id,null);
  }
  catch(IOException e){
    rx.avisaRecepcion((short) 2,id,null);
}

  catch(NullPointerException e){
    rx = new PaqueteTCPrecepcion();
    rx.addTCPListener(listener);
    rx.avisaRecepcion((short) 2,id,null);
  }
}

public InetAddress getDireccion(){
  return socket.getInetAddress();
}

public boolean enviar(PDU_TCP pdu){
  if ((socket != null) && (socket.isClosed() == false))
    return tx.enviar(pdu);
  else
    return false;
}

public void close(){
  try {
    rx.removeTCPListener(listener);
    socket.close();
    socket = null;
    tx.close();
```

```
    }
    catch (Exception e) {}
  }
}

//------------------------PAQUETE UDP-----------------------------------
class PaqueteUDPCliente{
  private PaqueteUDPtransmision tx;
  private PaqueteUDPrecepcion rx;
  private DatagramSocket socket;
  private Date ultimaRevision;
  private PaqueteUDPListener l;

  public PaqueteUDPCliente(int puerto, int UDPtimeout, PaqueteUDPListener l)
                          throws IOException{
    try{
      socket = new DatagramSocket(puerto);
      socket.setSoTimeout(UDPtimeout);

      tx = new PaqueteUDPtransmision(puerto,socket);
      rx = new PaqueteUDPrecepcion(socket);

      rx.addUDPListener(l);
      rx.start();
    }
    catch(IOException e){throw e;}
  }

  public void close(){
    socket.close();
  }

  public synchronized boolean enviar(PDU_UDP pdu, InetAddress addr){
    if(tx.setAddress(addr) == 0){
```

```
      tx.enviar(pdu);
      return true;
    }
    else
      return false;
  }
}
```

eventosClienteRed.java

```
import java.util.EventListener;
import java.util.EventObject;

/**Usado por el cliente para notificar al proceso principal alguna ocurrencia*/
class EventoCliente extends EventObject { //clase del evento
   short ID;

   short suceso;
   short motivo;

    public EventoCliente(Object source,short ID,short suceso, short motivo) {
       super(source);
       this.ID = ID;
       this.suceso = suceso;
       this.motivo=motivo;
       }

    public EventoCliente(Object source,short ID,short suceso){
      super(source);
      this.ID = ID;
      this.suceso = suceso;
      }
    }
```

```
interface ClienteListener extends EventListener {
      public void recibido(EventoCliente evento);
      }

**********INTERFAZ GRÁFICA**********
```

PanelLoginCliente.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class PanelLoginCliente extends JPanel{
  private GridBagConstraints constraints;

  private JTextField alias;
  private JPasswordField password;

  private ButtonGroup grupoBotones;

  public PanelLoginCliente(){
    super();

    setName("panel central");

    JRadioButton actuador;
    JRadioButton espectador;
    grupoBotones = new ButtonGroup();

    JButton boton;
    constraints = new GridBagConstraints();
    setLayout(new GridBagLayout());

    alias = new JTextField(1);
```

```
password = new JPasswordField(1);

boton = new JButton("login");
boton.setActionCommand("login");
boton.setDefaultCapable(true);

//RadioButtons:
actuador = new JRadioButton("Actuador");
actuador.setActionCommand("1");
actuador.setName("actuador");
actuador.setSelected(true);
grupoBotones.add(actuador);

espectador = new JRadioButton("Espectador");
espectador.setActionCommand("0");
espectador.setName("espectador");
grupoBotones.add(espectador);

constraints.anchor = GridBagConstraints.EAST;
addGB(this,new JLabel("login  "),0,0);
constraints.ipadx = 99;
constraints.ipady = 3;
addGB(this,alias,1,0);
constraints.ipady = 0;
constraints.ipadx = 0;
addGB(this,new JLabel("password  "),0,2);
constraints.ipadx = 100;
constraints.ipady = 1;
addGB(this,password,1,2);
constraints.ipady = 0;
constraints.ipadx = 10;
addGB(this,new JLabel(" "),2,2);
constraints.ipady = 0;
```

```
    constraints.ipadx = 0;
    constraints.anchor = GridBagConstraints.CENTER;
    addGB(this,boton,1,5);
//radio buttons
    JPanel panelBotones = new JPanel();
    Border borde = BorderFactory.createEtchedBorder();
    panelBotones.setLayout(new GridBagLayout());

    panelBotones.setBorder(BorderFactory.createTitledBorder(
      borde,"Tipo de Conexion"));

    constraints.anchor = GridBagConstraints.WEST;
    addGB(panelBotones,actuador,0,0);
    addGB(panelBotones,espectador,0,1);

    constraints.anchor = GridBagConstraints.EAST;
    constraints.ipadx = 15;
    addGB(this,panelBotones,3,2);



/***listeners*/
    boton.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ev) {
        short tipo;

        if (alias.getText().length() == 0){
          JOptionPane.showMessageDialog(foo(),"Debe introducir un login");
          return;
        }

        if (password.getPassword().length == 0){
          JOptionPane.showMessageDialog(foo(),"Debe introducir un password");
          return;
        }
```

```
      if (grupoBotones.getSelection().getActionCommand().equals("1"))
        tipo = 1; //actuador
      else
        tipo = 0; //espectador

     avisaLogin(alias.getText(),password.getPassword(),tipo);
    }
  });
}

private JPanel foo(){
  return this;
}

private void addGB(JPanel panel, Component component, int x, int y) {
  constraints.gridx = x;
  constraints.gridy = y;
  panel.add(component, constraints);
}

public void addPanelLoginListener(PanelLoginListener listener) {
  listenerList.add(PanelLoginListener.class, listener);
}

public void removePanelLoginListener(PanelLoginListener listener) {
  listenerList.remove(PanelLoginListener.class, listener);
}

void avisaLogin(String login, char[] password, short permiso) {
  Object[] listeners = listenerList.getListenerList();
  for (int i = 0; i < listeners.length; i += 2)
    if (listeners[i] == PanelLoginListener.class)
      ( (PanelLoginListener) listeners[i + 1]).loged(
```

```
            new PanelLoginEvent(this, login, password, permiso));
  }
}
```

AppletCliente.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.net.*;
import java.io.*;
import java.text.DateFormat;
import java.util.Date;

public class AppletCliente extends JApplet implements ClienteListener,
    PanelLoginListener{

  private ClienteRed cliente;
  private DatosMotoresCliente tablaMotores;

  private JSplitPane panelCentral;
  private JTextArea avisos;

  private valoresRed tablaRed;
  private boolean actuador;

  public AppletCliente() {
  }

  public void init() {
    try {
      jbInit();
      PanelLoginCliente panel;
```

```
tablaMotores = new DatosMotoresCliente();
Container contenido = getContentPane();

try{//obtenemos el archivo de configuracion de la red
  URL url = new URL(getCodeBase(),"red.inf");
  ObjectInputStream in = new ObjectInputStream(url.openStream());
  tablaRed = (valoresRed) in.readObject();
  in.close();
}
catch (Exception e){
  JOptionPane.showMessageDialog(this,"No ha sido posible cargar los " +
                        "parámetros de la conexión desde el " +
                        "servidor");
  throw(e);
}

panelCentral = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
panelCentral.setDividerLocation(panelCentral.getMaximumDividerLocation());
panelCentral.setDividerSize(-1);
panelCentral.setRightComponent(null);

Border borde = BorderFactory.createEtchedBorder();
panelCentral.setBorder(BorderFactory.createTitledBorder(borde));

contenido.add(barraSuperior(),BorderLayout.NORTH);
contenido.add(panelCentral,BorderLayout.CENTER);
contenido.add(barraInferior(),BorderLayout.SOUTH);

//inicialmente mostramos el panel de login
panel = new PanelLoginCliente();
panel.addPanelLoginListener(this);
panelCentral.setLeftComponent(panel);

actuador = true;
```

```
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  public void destroy(){
    if (cliente != null)
      cliente.close();  //desconectamos
  }

  private void jbInit() throws Exception {
    try{

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAnd
Feel");
      SwingUtilities.updateComponentTreeUI(this);
      }
      catch(Exception e){}
   }

//---------------------------BARRAS
   private JPanel barraSuperior(){
     JPanel panel = new JPanel();
     panel.setLayout(new FlowLayout());
     panel.setName("barrasuperior");

     JButton login;
     JButton ayuda;

     login = new JButton("  Login",crearIcono("iconos/login.gif"));
     login.setActionCommand("login");
     login.setName("login");
     login.setEnabled(false);
```

```
    panel.add(login);

    ayuda = new JButton("  Ayuda",crearIcono("iconos/ayuda.gif"));
    ayuda.setActionCommand("ayuda");
    ayuda.setName("ayuda");
    panel.add(ayuda);

/***Listeners*/
    //login
    login.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ev){
        Component componente = buscaComponente(panelCentral, "panel central");

        if (! (componente instanceof PanelLoginCliente)) {
//LOGOUT
          int result = JOptionPane.showConfirmDialog(foo(),
             "¿Desea abandonar la sesión?", "Applet USBMotor",
             JOptionPane.YES_NO_OPTION);

          if (result == JOptionPane.YES_OPTION) {
            cliente.close();
            login(false,(short)0);
          }
        }
      }
    });

    ayuda.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
        PanelAyuda ayuda = new PanelAyuda(true,getCodeBase());
      }
    });

    return panel;
```

```
  }

  private JPanel barraInferior() {
    JPanel panel = new JPanel();
    panel.setName("barrainferior");

    panel.setLayout(new FlowLayout());

    JButton modo;
    JScrollPane panelAvisos;



    modo = new JButton("Cambio de Modo");
    modo.setActionCommand("espectador");
    modo.setEnabled(false);
    modo.setName("botontipoconexion");

    avisos = new JTextArea();
    avisos.setEditable(false);

    panelAvisos = new JScrollPane(avisos);
    panelAvisos.setWheelScrollingEnabled(true);
    panelAvisos.setHorizontalScrollBarPolicy(JScrollPane.
                                  HORIZONTAL_SCROLLBAR_ALWAYS);
    panelAvisos.setVerticalScrollBarPolicy(JScrollPane.
                                VERTICAL_SCROLLBAR_ALWAYS);
    panelAvisos.setPreferredSize(new Dimension(300, 60));

    panel.add(panelAvisos);
    panel.add(modo);

     //movimiento local ó remoto
    modo.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
```

```
        JButton boton = (JButton) ev.getSource();
        Component panel = panelCentral.getLeftComponent();

        if (boton.getActionCommand().equals("espectador")) {
          PDU_TCP pdu = new PDU_TCP();
          pdu.setTipo(4,0);
          cliente.enviarTCP(cliente.actuador,pdu);
        }
        else{
          PDU_UDP pdu = new PDU_UDP();
          pdu.setTipo(4,1);
          cliente.enviarUDP(pdu);
          }
      }
    });

    return panel;
  }

 //-------------------------REPRESENTAR EN EL AREA DE EVENTOS

   public synchronized void anadeAviso(String mensaje) {
    DateFormat fecha = DateFormat.getInstance();

    avisos.append(fecha.format(new Date()) + " : " + mensaje + "\n");
  }



//--------------------------------INFO DEL APPLET--------------
  //Get Applet information
  public String getAppletInfo() {
    return "Proyecto Fin de Carrera. Ruperto J. Hdez. Saiz";
  }
```

```
  //Get parameter info
  public String[][] getParameterInfo() {
    return null;
  }

//-----------------------------GESTORES DE EVENTOS
  public void recibido (EventoCliente ev){ //eventos de la red
    /** 0:Desconexion por Timeout (el servidor no responde)
                * 1:Desconexion por error I/O
                * 2:Desconexion por error en el protocolo (pdu)
                * 3:Aceptado modo Visor (UDP)
                * 4:Aceptado modo Actuador
                * //5:Aceptado modo Administrador
                * 6:Conexion rechazada (motivo tiene un valor valido)
                * 7:Disponible modo Actuador  (motivo tiene valor valido (indica si
esta disponible o no))
                * 8:Desconexion por cierre del servidor
                * 9:Desconexion por tiempo de la conexion agorado
                * 10:Desconexion por Administrador
                *
                * 11:Configuracion de los motores
                * 12:Motor Movido
                * 13:Error al mover el motor
                * 14:Recibimos un movimiento (estamos en espectador)
                * */

    switch(ev.suceso){
      case 0:
        JOptionPane.showMessageDialog(this, "El servidor no responde");
        anadeAviso("El servidor no responde");
        login(false,(short)0);
        cliente.close();
        break;
```

```
case 1:
  JOptionPane.showMessageDialog(this, "Error de E/S");
  anadeAviso("No es posible acceder al servidor");
  login(false,(short)0);
  cliente.close();
  break;
case 2:
  JOptionPane.showMessageDialog(this, "Error en el protocolo");
  anadeAviso("Error en el protocolo");
  login(false,(short)0);
  cliente.close();
  break;

case 3:
  anadeAviso("Aceptado Modo Espectador");
  login(true,(short) 0);
  actuador = false;

  if (panelCentral.getLeftComponent() instanceof PanelMovimiento){
    ((PanelMovimiento) panelCentral.getLeftComponent()).
        setPermisoMovimiento(false);
    cambioModo((short)0,false);
  }
   break;
case 4:
  anadeAviso("Aceptado Modo Actuador");
  login(true,(short) 1);
  actuador = true;

  if (panelCentral.getLeftComponent() instanceof PanelMovimiento){
    ( (PanelMovimiento) panelCentral.getLeftComponent()).
        setPermisoMovimiento(true);
    cambioModo((short)1,true);
  }
```

```
        break;

      case 6:
        if (panelCentral.getLeftComponent() instanceof PanelLoginCliente){
          switch (ev.motivo) {
            case 0:
              JOptionPane.showMessageDialog(this, "Usuario No Válido");
              break;
            case 1:
              JOptionPane.showMessageDialog(this, "Password Incorrecto");
              break;
            case 2:
              JOptionPane.showMessageDialog(this,
                 "No tiene permiso para acceder al servicio");
              break;
            case 3:
              JOptionPane.showMessageDialog(this,
                                  "El servicio requerido está actualmente en uso");
              break;
            case 4:
              JOptionPane.showMessageDialog(this,
                                  "Se ha alcanzado el número máximo de conexiones");
              break;
          }
          login(true,(short) 0);
        }
        else{
          switch (ev.motivo) {
            case 0:
              anadeAviso("No ha sido posible cambiar el tipo de conexion : "
                      + "Usuario No Válido");
              break;
            case 1:
              anadeAviso("No ha sido posible cambiar el tipo de conexion : "
```

```
                + "Password Incorrecto");
        break;
      case 2:
        anadeAviso("No ha sido posible cambiar el tipo de conexion : "
                + "No tiene permiso para acceder al servicio");
        break;
      case 3:
        anadeAviso("No ha sido posible cambiar el tipo de conexion : "
                + "El servicio requerido está actualmente en uso");
        break;
      case 4:
        anadeAviso("No ha sido posible cambiar el tipo de conexion : "
                + "Se ha alcanzado el número máximo de conexiones");
        break;
    }
  }
  break;

case 7:
  JPanel panelTemp = (JPanel) buscaComponente(this.getContentPane(),
                                    "barrainferior");
  JButton boton = (JButton) buscaComponente(panelTemp,
                                    "botontipoconexion");

  if (ev.motivo == 0){
    if ( (!boton.getActionCommand().equals("actuador")) ||
        (!boton.isEnabled())) {
      cambioModo( (short) 0, true);
      anadeAviso("Disponibilidad de conexion Actuadora");
    }
  }
  else{
      boton.setEnabled(false);
      anadeAviso("La conexión Actuadora ha dejado de estar disponible");
```

```
    }
    break;

  case 8:
    if (ev.ID != cliente.gestor){
      JOptionPane.showMessageDialog(this, "El servidor ha sido cerrado");
      anadeAviso("Servidor Cerrado");
      login(false, (short) 0);
    }
    break;

  case 9:
    anadeAviso("Tiempo de conexion agotado");
    actuador = false;

    if (panelCentral.getLeftComponent() instanceof PanelMovimiento){
      ( (PanelMovimiento) panelCentral.getLeftComponent()).
          setPermisoMovimiento(false);
      cambioModo( (short) 0, false);
    }
    break;

  case 10:
    JOptionPane.showMessageDialog(this,
                          "El Administrador está orientando "
                        +"los motores.\n Activado el modo Espectador");

    anadeAviso("El Administrador está orientando los motores. "+
              "Activado el modo Espectador");
    actuador = false;
    if (panelCentral.getLeftComponent() instanceof PanelMovimiento){
      ((PanelMovimiento) panelCentral.getLeftComponent()).
          setPermisoMovimiento(false);
      cambioModo((short)0,false);
```

```
  }
  break;

case 11:
  PanelMovimiento movimiento = new PanelMovimiento(
                              tablaMotores.getNpasos(1),
                              tablaMotores.getNpasos(2),
                              tablaMotores.getPasoActual(1),
                              tablaMotores.getPasoActual(2),
                              tablaMotores.getVelocidadActual(1),
                              tablaMotores.getVelocidadActual(2),
                              tablaMotores.getAnguloMaximo(1),
                              tablaMotores.getAnguloMaximo(2),
                              tablaMotores.getVelocidadMaxima(1),
                              tablaMotores.getVelocidadMaxima(2),
                              actuador);
  movimiento.addPanelMovimientoListener(cliente);

  panelCentral.setLeftComponent(movimiento);
  break;

case 12:
  boolean correcto = true;
  Component panel;

  if (tablaMotores.getAnguloActual(1) != tablaMotores.getAnguloEnviar(1))
    correcto = false;
  else
    if (tablaMotores.getAnguloActual(2) != tablaMotores.getAnguloEnviar(2))
      correcto = false;
    else
      if (tablaMotores.getVelocidadActual(1) !=
          tablaMotores.getVelocidadEnviar(1))
        correcto = false;
```

```
        else
          if (tablaMotores.getVelocidadActual(2) !=
              tablaMotores.getVelocidadEnviar(2))
            correcto = false;

    if (correcto){
     anadeAviso("El movimiento ha sido ejecutado correctamente");
     if ((panel = panelCentral.getLeftComponent())
        instanceof PanelMovimiento){
      ((PanelMovimiento) panel).setDatos(tablaMotores.getPasoActual(1),
                                tablaMotores.getPasoActual(2),
                                tablaMotores.getVelocidadActual(1),
                                tablaMotores.getVelocidadActual(2),
                                true);
      }
    }
    else{
      anadeAviso("El movimiento NO ha sido ejecutado correctamente");
      if ((panel = panelCentral.getLeftComponent())
          instanceof PanelMovimiento)
       ((PanelMovimiento)panel).falloMover();
    }
     break;
    case 13:
     anadeAviso("Se ha producido un error en el servidor "+
              "y no ha sido posible realizar el movimiento");
     JOptionPane.showMessageDialog(foo(),"Se ha producido un error en el "
              +"servidor y no ha sido posible realizar el movimiento");
     if ((panel = panelCentral.getLeftComponent())
        instanceof PanelMovimiento)
       ((PanelMovimiento)panel).falloMover();
     break;

    case 14:
```

```
        anadeAviso("Ha habido un movimiento");
        if ((panel = panelCentral.getLeftComponent())
           instanceof PanelMovimiento){
         ((PanelMovimiento) panel).setDatos(tablaMotores.getPasoActual(1),
                                    tablaMotores.getPasoActual(2),
                                    tablaMotores.getVelocidadActual(1),
                                    tablaMotores.getVelocidadActual(2),
                                    false);
        }
        break;
    }
  }

  public void loged(PanelLoginEvent ev){
    String host = getCodeBase().getHost();

    if (cliente != null)
      cliente.close();

    try{
      cliente = new ClienteRed(InetAddress.getByName(host),
                          tablaRed.timeoutGestion,
                          tablaRed.timeoutEspectador,
                          tablaRed.puertoGestion,
                          ev.login,
                          new String(ev.password),
                          ev.permiso,
                          tablaMotores,
                          this);
      anadeAviso("Login : " + ev.login);
    }
    catch (IOException e){
      JOptionPane.showMessageDialog(this,"El servidor no responde");
    }
```

```
  }

//------------------VARIOS

  private void login(boolean loged,short tipo) {
    JPanel panel;
    JButton boton;

    Container contenido = this.getContentPane();
    panel = (JPanel) buscaComponente(contenido, "barrasuperior");

    boton = (JButton) buscaComponente(panel, "login");
    if (loged) {
      boton.setText("  Logout");

      cambioModo(tipo,false);
      boton.setEnabled(true);
    }
    else {
      boton.setText("  Login");
      boton.setEnabled(false);

      panel = (JPanel) buscaComponente(contenido, "barrainferior");
      boton = (JButton) buscaComponente(panel, "botontipoconexion");
      boton.setEnabled(false);

      PanelLoginCliente panelLogin = new PanelLoginCliente();
      panelLogin.addPanelLoginListener(this);
      panelCentral.setLeftComponent(panelLogin);

      anadeAviso("Logout");
      boton.setEnabled(false);
    }
  }
```

```
private void cambioModo(short tipo,boolean permiso){
  JPanel panel;
  JButton boton;

  panel = (JPanel) buscaComponente(this.getContentPane(), "barrainferior");
  boton = (JButton) buscaComponente(panel, "botontipoconexion");

  if (tipo == 1){ //modo actuador
    boton.setText("  Espectador");
    boton.setIcon(crearIcono("iconos/controlLocal.gif"));
    boton.setActionCommand("espectador");
    boton.setEnabled(true); //siendo actuador, puede pasarse a espectador
  }
  else{
    boton.setText("  Actuador");
    boton.setIcon(crearIcono("iconos/controlRemoto.gif"));
    boton.setActionCommand("actuador");
    boton.setEnabled(permiso); //siendo espectador, puede pasarse a actuador
  }
}

private Component buscaComponente(Container panel,String nombre){
 try {
   Component[] componentes = panel.getComponents();

   for (int i = 0; i < panel.getComponentCount(); i++) {
     if (componentes[i].getName() != null)
       if (componentes[i].getName().equals(nombre))
         return componentes[i];
   }
   return null;
 }
 catch (NullPointerException e) {
```

```
     return null;
     }
   }


  private ImageIcon crearIcono(String path){
    try{
      URL url = new URL(getCodeBase(),path);
      return new ImageIcon(getImage(url));
    }
    catch(Exception e){
      return null;
    }
  }

  private JApplet foo(){
    return this;
  }
}
```

DatosMotoresCliente.java

```
public class DatosMotoresCliente {
  private short flag1;
  private short flag2;
  private double anguloMaximo1;
  private double anguloMaximo2;
  private int npasos1;
  private int npasos2;
  private short VelocidadMaxima1;
  private short VelocidadMaxima2;

  private double anguloActual1;
  private double anguloActual2;
```

```
 private int velocidadActual1;
 private int velocidadActual2;

 private double anguloEnviar1;
 private double anguloEnviar2;
 private int velocidadEnviar1;
 private int velocidadEnviar2;

/* private short flagEnviado; /**0: Ambos motores
                              1: Motor 1
                              2: Motor 2*/


 //---------------SET
 public synchronized void setNPasos(int ID,int npasos){
   if (ID == 1)
     this.npasos1 = npasos;
   else
     this.npasos2 = npasos;
 }

 public synchronized void setFlag(int ID,short flag){
   if (ID ==1)
     this.flag1 = flag;
   else
     this.flag2= flag;
 }

 public synchronized void setAnguloMaximo(int ID, double anguloMaximo){
   if (ID == 1)
     this.anguloMaximo1 = anguloMaximo;
   else
     this.anguloMaximo2 = anguloMaximo;
 }
```

```
public synchronized void setAnguloActual(int ID, double anguloActual){
  if (ID == 1)
    this.anguloActual1 = anguloActual;
  else
    this.anguloActual2 = anguloActual;
}

public synchronized void setVelocidadActual(int ID, int velocidad){
  if (ID == 1)
    this.velocidadActual1 = velocidad;
  else
    this.velocidadActual2 = velocidad;
}

public synchronized void setAnguloEnviar(int ID, double angulo){
  if (ID == 1)
    this.anguloEnviar1 = angulo;
  else
    this.anguloEnviar2 = angulo;
}

public synchronized void setVelocidadEnviar(int ID, int velocidad){
  if (ID == 1)
    this.velocidadEnviar1 = velocidad;
  else
    this.velocidadEnviar2 = velocidad;
}

public synchronized void setVelocidadMaxima(int ID, short velocidad){
  if (ID == 1)
    this.VelocidadMaxima1 = velocidad;
  else
    this.VelocidadMaxima2 = velocidad;
```

```
}

//------------------------GET

public synchronized short getFlag(int ID){
  if (ID == 1)
    return this.flag1;
  else
    return this.flag2;
}

public synchronized int getNpasos(int ID){
  if (ID == 1)
    return this.npasos1;
  else
    return this.npasos2;
}

public synchronized int getPasoActual(int ID){
  if (ID == 1)
    return (int) (anguloActual1 / (360.0 / npasos1));
  else
    return (int) (anguloActual2 / (360.0 / npasos2));
}

public synchronized double getAnguloMaximo(int ID){
  if (ID == 1)
    return this.anguloMaximo1;
  else
    return this.anguloMaximo2;
}

public synchronized double getAnguloActual(int ID){
  if (ID == 1)
```

```
      return this.anguloActual1;
    else
      return this.anguloActual2;
  }

  public synchronized int getVelocidadActual(int ID){
    if (ID == 1)
      return this.velocidadActual1;
    else
      return this.velocidadActual2;
  }

  public synchronized double getAnguloEnviar(int ID){
    if (ID == 1)
      return this.anguloEnviar1;
    else
      return this.anguloEnviar2;
  }

  public synchronized int getVelocidadEnviar(int ID){
    if (ID == 1)
      return this.velocidadEnviar1;
    else
      return this.velocidadEnviar2;
  }

  public synchronized short getVelocidadMaxima(int ID){
    if (ID == 1)
      return this.VelocidadMaxima1;
    else
      return this.VelocidadMaxima2;
  }
}
```

ePipe.h

```
class Pipe
{
private:
	HANDLE		hPipe;
	bool		fConnected;
	HANDLE		hEvento;
	OVERLAPPED	hOverlapped;

public:
	Pipe();
	virtual ~Pipe();

	bool activo();
	bool leer(char *,DWORD);
	bool escribir(char *,DWORD);
	bool IniciaPipe();
	bool ConectaPipe(DWORD);
};
```

Pipe.cpp

```
#include "stdafx.h"
#include "usbhidioc.h"
#include "Pipe.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

Pipe::Pipe()
{
```

```
        if (IniciaPipe() == true)
                ConectaPipe(INFINITE);
}

Pipe::~Pipe()
{
        CancelIo(hPipe);
        DisconnectNamedPipe(hPipe);
        CloseHandle(hPipe);
}


bool Pipe::leer(char *bufferRead,DWORD bufferSize)
{
        bool fSuccess;
        DWORD cbRead;
        int Result;

        fSuccess = ReadFile(
       hPipe,
       bufferRead,
       bufferSize,
       &cbRead,
       (LPOVERLAPPED) &hOverlapped);


        int temp = GetLastError();
        Result = WaitForSingleObject
                (hEvento,
                INFINITE);

        switch (Result)
        {
        case WAIT_OBJECT_0:
```

```
                {
                return true;
                break;
                }
        case WAIT_TIMEOUT:
                {
                Result = CancelIo(hPipe);
                return false;
                break;
        default:
                return false;
                break;
                }
        }

        ResetEvent(hEvento);
}

bool Pipe::escribir(char *bufferWrite,DWORD bufferSize)
{
        bool fSuccess;
        DWORD cbWritten;
        int Result;

        fSuccess = WriteFile(
                 hPipe,
       bufferWrite,
       bufferSize,
       &cbWritten,
       (LPOVERLAPPED) &hOverlapped);


        Result = WaitForSingleObject
                (hEvento,
```

```
            INFINITE);

        switch (Result)
        {
        case WAIT_OBJECT_0:
            {
            break;
            }
        case WAIT_TIMEOUT:
            {

            Result = CancelIo(hPipe);

            return false;
            break;
        default:
            return false;
            break;
            }
        }

        ResetEvent(hEvento);

        if (cbWritten == bufferSize)
            return true;
        else
            return false;
}

bool Pipe::activo()
{
        if (fConnected == true)
            return true;
        else
```

```
                return false;
}

bool Pipe::IniciaPipe()
{
        DWORD tamanoBuffer = 10;
        LPTSTR lpszPipename = "\\\\.\\pipe\\pfc_usb_pipe";
        int PIPE_TIMEOUT = 20000;

        hPipe = CreateNamedPipe(
       lpszPipename,           // nombre
       PIPE_ACCESS_DUPLEX |     // acceso en ambos sentidos
       FILE_FLAG_OVERLAPPED,
       PIPE_TYPE_MESSAGE |      // pipe de tipo mensaje
       PIPE_READMODE_MESSAGE |
       PIPE_WAIT,               // bloqueante
       2,                               // número de instancias
       tamanoBuffer,           // salida
       tamanoBuffer,           // entrada
       PIPE_TIMEOUT,
       NULL);                  // atributos de seguridad

        if (hPipe == INVALID_HANDLE_VALUE)
       return false;


        hEvento = CreateEvent //preparamos el evento (espera por la conexion)
                        (NULL,
                        TRUE,
                        TRUE,
                        "");

        hOverlapped.hEvent = hEvento;
        hOverlapped.Offset = 0;
```

```
        hOverlapped.OffsetHigh = 0;
        return true;
}

bool Pipe::ConectaPipe(DWORD delay)
{
  int Result;
  DisconnectNamedPipe(hPipe);
  ConnectNamedPipe(hPipe, &hOverlapped);

  if (GetLastError() == ERROR_PIPE_CONNECTED)
    SetEvent(hOverlapped.hEvent);  // El cliente ya está conectado

        Result = WaitForSingleObject
                (hEvento,
                delay);         //esperamos a que se conecte el cliente

        switch (Result)
        {
        case WAIT_OBJECT_0:
                {
                break; //correcto
                }
        case WAIT_TIMEOUT:
                {
                CancelIo(hPipe);
                DisconnectNamedPipe(hPipe);
                CloseHandle(hPipe);
                fConnected = false;
                return false;

                break;
        default:
                fConnected = false;
```

```
                return false;
                break;
                }
        }

        ResetEvent(hEvento);
        fConnected = true;
        return true;
}
```

usbhidiocDlg.h

```
#if
!defined(AFX_USBHIDIOC_H__0B2AAA82_F5A9_11D3_9F47_0050048108EA__
INCLUDED_)
#define
AFX_USBHIDIOC_H__0B2AAA82_F5A9_11D3_9F47_0050048108EA__INCLUD
ED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"                // main symbols

/////////////////////////////////////////////////////////////////////////////
// CUsbhidiocApp:
// See usbhidioc.cpp for the implementation of this class
//
```

```
class CUsbhidiocApp : public CWinApp
{
public:
	CUsbhidiocApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CUsbhidiocApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CUsbhidiocApp)
		// NOTE - the ClassWizard will add and remove member functions
here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_USBHIDIOC_H__0B2AAA82_F5A9_11D3_9F47_0050048108EA__
INCLUDED_
```

usbhidiocDlg.cpp

```
#include "stdafx.h"

#include "usbhidioc.h"
#include "usbhidiocDlg.h"

#include "Pipe.h"

#include <wtypes.h>
#include <initguid.h>

#define MAX_LOADSTRING 256

extern "C" {
#include "hidsdi.h"
#include <setupapi.h>
}

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

bool FindTheHID();
void DisplayData(CString cstrDataToDisplay);
void DisplayLastError(CString Operation);
void DisplayReceivedData(char ReceivedByte);
void GetDeviceCapabilities();
void PrepareForOverlappedTransfer();
void ReadAndWriteToDevice();
void ReadReport();
void ScrollToBottomOfListBox(USHORT Index);
```

```
void WriteReport();
UINT ControlThread(LPVOID);
UINT ControlLecturaUSB(LPVOID);

extern void WINAPI HIDIOCompletionRoutine(DWORD, DWORD,
LPOVERLAPPED);
long calculaTiempo(char, char);

#define WM_PIPE_ACTIVO      0x8008
#define WM_CERRAR                   0x8009
#define Salir                       0
#define LongitudTrama               10          //longitud de la trama enviada al
USB
#define izq                         0
#define der                         1

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
        //}}AFX_VIRTUAL


protected:
        //{{AFX_MSG(CAboutDlg)
```

```
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

CUsbhidiocDlg::CUsbhidiocDlg(CWnd* pParent)
        : CDialog(CUsbhidiocDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CUsbhidiocDlg)
        m_AutoIncrement = FALSE;
        m_ResultsString = _T("");
        m_strBytesReceived = _T("");
        m_strByteToSend0 = _T("");
        m_strByteToSend1 = _T("");
        //}}AFX_DATA_INIT
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
```

```
}

        DWORD                   ActualBytesRead;
        DWORD                   BytesRead;
        HIDP_CAPS       Capabilities;
        DWORD                   cbBytesRead;
        PSP_DEVICE_INTERFACE_DETAIL_DATA   detailData;
        bool            DeviceDetected;
        HANDLE                  DeviceHandle;
        DWORD                   dwError;
        HANDLE                  hEventObject;
        HANDLE                  hDevInfo;
        GUID            HidGuid;
        OVERLAPPED              HIDOverlapped;
        char            InputReport[LongitudTrama];
        ULONG                   Length;
        LPOVERLAPPED    lpOverLap;
        DWORD                   NumberOfBytesRead;
        HANDLE                  ReadHandle;
        ULONG                   Required;
        CString                 ValueToDisplay;


        HWND                    Ventana;
        CHAR            vector[LongitudTrama];
        HANDLE                  hEventoUSB;
        long            tiempo;

void CUsbhidiocDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CUsbhidiocDlg)
        DDX_Control(pDX, IDC_Continuous, m_Continuous);
        DDX_Control(pDX, IDC_AutoIncrement, m_cbutAutoIncrement);
```

```
        DDX_Control(pDX, IDC_cboByteToSend1, m_cboByteToSend1);
        DDX_Control(pDX, IDC_cboByteToSend0, m_cboByteToSend0);
        DDX_Control(pDX, IDC_lstBytesReceived, m_BytesReceived);
        DDX_Control(pDX, IDC_LIST2, m_ResultsList);
        DDX_Control(pDX, IDC_Once, m_Once);
        DDX_Check(pDX, IDC_AutoIncrement, m_AutoIncrement);
        DDX_LBString(pDX, IDC_LIST2, m_ResultsString);
        DDX_LBString(pDX, IDC_lstBytesReceived, m_strBytesReceived);
        DDX_CBString(pDX, IDC_cboByteToSend0, m_strByteToSend0);
        DDX_CBString(pDX, IDC_cboByteToSend1, m_strByteToSend1);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CUsbhidiocDlg, CDialog)
        //{{AFX_MSG_MAP(CUsbhidiocDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        ON_BN_CLICKED(IDC_Once, OnOnce)
        ON_EN_CHANGE(IDC_Results, OnChangeResults)
        ON_BN_CLICKED(IDC_Continuous, OnContinuous)
        ON_WM_TIMER()
        ON_WM_CLOSE()
        ON_LBN_SELCHANGE(IDC_lstBytesReceived2,
OnSelchangelstBytesReceived2)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CUsbhidiocDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        int             ByteToSend =0;
        CString         strByteToSend = "";
```

```
	CString		strComboBoxText="";

	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	SetIcon(m_hIcon, TRUE);
	SetIcon(m_hIcon, FALSE);


	DeviceDetected=FALSE;

	Ventana = GetSafeHwnd();

	AfxBeginThread(ControlThread,
				NULL,
				THREAD_PRIORITY_NORMAL,
				0,
				0,
				NULL);

	return TRUE;
```

```
}

LRESULT CUsbhidiocDlg::WindowProc(UINT message,WPARAM
wParam,LPARAM lParam )
{
	switch (message)
	{
	case WM_PIPE_ACTIVO:
		{
			if (vector[0] != Salir)
			{
				WriteReport();
				ReadReport();
			}
			else
			{
				OnClose();
				EndDialog(0);
				PostQuitMessage(0);
				DestroyWindow();
			}
		break;
		}

	case WM_DESTROY:
		{
			AfxAbort();
			break;
		}
	case WM_CLOSE:
	case WM_CERRAR:
		{
			OnClose();
			EndDialog(0);
```

```
                PostQuitMessage(0);
                DestroyWindow();
                break;
            }
        case WM_PAINT:
            {
                OnPaint();
                break;
            }
        case WM_QUERYDRAGICON:
            {
                OnQueryDragIcon();
                break;
            }
        case WM_SYSCOMMAND:
            {
                OnSysCommand(wParam, lParam);
                break;
            }
        default:
         return DefWindowProc (message, wParam, lParam);
        }
        return 0;
}


void CUsbhidiocDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
        }
        else
```

```
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

void CUsbhidiocDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this);

		SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

HCURSOR CUsbhidiocDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}
```

```
void CUsbhidiocDlg::OnClose()
{
	CloseHandle(DeviceHandle);
	DisplayLastError("CloseHandle(DeviceHandle)");

	CloseHandle(ReadHandle);
	DisplayLastError("CloseHandle(ReadHandle)");

	CDialog::OnClose();
}

bool CUsbhidiocDlg::FindTheHID(unsigned int VendorID,unsigned int ProductID)
{
	HIDD_ATTRIBUTES		Attributes;
	SP_DEVICE_INTERFACE_DATA	devInfoData;
	bool		LastDevice = FALSE;
	int		MemberIndex = 0;
	bool		MyDeviceDetected = FALSE;
	LONG		Result;

	Length = 0;
	detailData = NULL;
	DeviceHandle=NULL;

	HidD_GetHidGuid(&HidGuid); //obtiene el GUID (Globally unique identifier) para todos los sistemas HID

	hDevInfo=SetupDiGetClassDevs //devuelve el manejador de la información de los dispositivos instalados
		(&HidGuid,
		NULL,
```

```
            NULL,
            DIGCF_PRESENT|DIGCF_INTERFACEDEVICE);

        devInfoData.cbSize = sizeof(devInfoData);

        MemberIndex = 0;
        LastDevice = FALSE;

        //encuesta a los dispositivos conectados
        do
        {
            MyDeviceDetected=FALSE;


            Result=SetupDiEnumDeviceInterfaces
                (hDevInfo,
                0,
                &HidGuid,
                MemberIndex,
                &devInfoData);

            if (Result != 0)
            {//Un dispositivo ha sido detectado, por tanto obtenemos más
información del mismo

                Result = SetupDiGetDeviceInterfaceDetail
                    (hDevInfo,
                    &devInfoData,
                    NULL,
                    0,
                    &Length,
                    NULL);
```

```
                detailData =
(PSP_DEVICE_INTERFACE_DETAIL_DATA)malloc(Length);

                detailData -> cbSize =
sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

                //llamamos de nuevo a la función, teniendo ya su longitud,
para que no dé
                //la estructura en si
                Result = SetupDiGetDeviceInterfaceDetail
                        (hDevInfo,
                        &devInfoData,
                        detailData,
                        Length,
                        &Required,
                        NULL);

                //obtenemos el Manejador del dispositivo
                DeviceHandle=CreateFile
                        (detailData->DevicePath,
                        GENERIC_READ|GENERIC_WRITE,
                        FILE_SHARE_READ|FILE_SHARE_WRITE,
                        (LPSECURITY_ATTRIBUTES)NULL,
                        OPEN_EXISTING,
                        0,
                        NULL);

                DisplayLastError("CreateFile: ");

                Attributes.Size = sizeof(Attributes);

                //Solicitamos información del dispositivo
                Result = HidD_GetAttributes
                        (DeviceHandle,
```

```
                    &Attributes);

                MyDeviceDetected = FALSE;

                if (Attributes.VendorID == VendorID) //comprobamos si
coinciden
                {
                    if (Attributes.ProductID == ProductID)
                    {
                        MyDeviceDetected = TRUE;
                        DisplayData("Device detected");
                        GetDeviceCapabilities();
                        PrepareForOverlappedTransfer();
                    }

                    else
                        CloseHandle(DeviceHandle);
                }
                else
                    CloseHandle(DeviceHandle);

            free(detailData);
            }

            else
                LastDevice=TRUE;

            //si no lo hemos encontrado, lo volvemos a intentar
            MemberIndex = MemberIndex + 1;

        }
        while ((LastDevice == FALSE) && (MyDeviceDetected == FALSE));

        if (MyDeviceDetected == FALSE) //no ha sido encontrado
```

```
            DisplayData("Dispositivo NO conectado");
        else
            DisplayData("Dispositivo Conectado");

        //liberación de los recursos reservados por hDevInfo y SetupDiClassDevs
        SetupDiDestroyDeviceInfoList(hDevInfo);

        return MyDeviceDetected;
}

void CUsbhidiocDlg::GetDeviceCapabilities()
{
        PHIDP_PREPARSED_DATA     PreparsedData;

        //Devuelve un puntero a un buffer con la información
        //de las capacidades del dispositivo
        HidD_GetPreparsedData
            (DeviceHandle,
            &PreparsedData);

        //Devuelve las capacidades del dispositivo
        HidP_GetCaps
            (PreparsedData,
            &Capabilities);

        ValueToDisplay.Format("%s%X", "Usage Page: ", Capabilities.UsagePage);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Input Report Byte Length: ",
Capabilities.InputReportByteLength);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Output Report Byte Length: ",
Capabilities.OutputReportByteLength);
        DisplayData(ValueToDisplay);
```

```
        ValueToDisplay.Format("%s%d", "Feature Report Byte Length: ",
Capabilities.FeatureReportByteLength);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of Link Collection Nodes: ",
Capabilities.NumberLinkCollectionNodes);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of Input Button Caps: ",
Capabilities.NumberInputButtonCaps);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of InputValue Caps: ",
Capabilities.NumberInputValueCaps);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of InputData Indices: ",
Capabilities.NumberInputDataIndices);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of Output Button Caps: ",
Capabilities.NumberOutputButtonCaps);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of Output Value Caps: ",
Capabilities.NumberOutputValueCaps);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of Output Data Indices: ",
Capabilities.NumberOutputDataIndices);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of Feature Button Caps: ",
Capabilities.NumberFeatureButtonCaps);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of Feature Value Caps: ",
Capabilities.NumberFeatureValueCaps);
        DisplayData(ValueToDisplay);
        ValueToDisplay.Format("%s%d", "Number of Feature Data Indices: ",
Capabilities.NumberFeatureDataIndices);
        DisplayData(ValueToDisplay);
```

```
        HidD_FreePreparsedData(PreparsedData);
}

void CUsbhidiocDlg::PrepareForOverlappedTransfer()
{
        ReadHandle=CreateFile
                (detailData->DevicePath,
                GENERIC_READ|GENERIC_WRITE,
                FILE_SHARE_READ|FILE_SHARE_WRITE,
                (LPSECURITY_ATTRIBUTES)NULL,
                OPEN_EXISTING,
                FILE_FLAG_OVERLAPPED,
                NULL);

        if (hEventObject == 0)
        {
                hEventObject = CreateEvent
                        (NULL,
                        TRUE,
                        TRUE,
                        "");

        //miembros de la estructura solapada
        HIDOverlapped.hEvent = hEventObject;
        HIDOverlapped.Offset = 0;
        HIDOverlapped.OffsetHigh = 0;
        }
}

void CUsbhidiocDlg::ReadReport()
{
        //lee la respuesta del dispositivo
        CString     ByteToDisplay = "";
        CString     MessageToDisplay = "";
```

```
        DWORD       Result;

        if (DeviceDetected==FALSE)
                DeviceDetected=FindTheHID(0,0);

        if (DeviceDetected==TRUE)
                {
                Result = ReadFile
                        (ReadHandle,
                        InputReport,
                        Capabilities.InputReportByteLength,
                        &NumberOfBytesRead,
                        (LPOVERLAPPED) &HIDOverlapped);

        //llamamos a un thread para que se quede a la espera de la respuesta del
dispositivo
                AfxBeginThread(ControlLecturaUSB,
                                        NULL,
                                        THREAD_PRIORITY_NORMAL,
                                        0,
                                        0,
                                        NULL);
        }
        else{
                InputReport[0]=5;
                SetEvent(hEventoUSB);
        }

}

void CUsbhidiocDlg::WriteReport()
{
        //envía datos al dispositivo
        const unsigned short int    MAXREPORTSIZE = 256;
```

```
DWORD       BytesWritten = 0;
INT         Index =0;
CHAR OutputReport[MAXREPORTSIZE];
ULONG       Result;
CString     strBytesWritten = "";

OutputReport[0]=0;

for(int i=1;i<(LongitudTrama-2);i++)
        OutputReport[i]=(CHAR)vector[i];

if (DeviceDetected==FALSE)
        DeviceDetected=FindTheHID(0,0);

if (DeviceDetected==TRUE)
        {

        Result = WriteFile
                (DeviceHandle,
                OutputReport,
                Capabilities.OutputReportByteLength,
                &BytesWritten,
                NULL);

        if (Result == 0) //falló
        {
                CloseHandle(DeviceHandle);
                CloseHandle(ReadHandle);
                DisplayData("No se pudo enviar información al dispositivo
USB");
                DeviceDetected = FALSE;
        }
        else
```

```
                DisplayReceivedData();
        }
}

void CUsbhidiocDlg::DisplayData(CString cstrDataToDisplay)
{
        //Muestra los datos en el cuadro inferior
        USHORT      Index;
        Index=m_ResultsList.InsertString(-1, (LPCTSTR)cstrDataToDisplay);
        ScrollToBottomOfListBox(Index);
}

void CUsbhidiocDlg::DisplayLastError(CString Operation)
{
        //Muestra el "ultimo error"
        LPVOID lpMsgBuf;
        USHORT Index = 0;
        CString        strLastError = "";
        FormatMessage(
                FORMAT_MESSAGE_ALLOCATE_BUFFER |
                FORMAT_MESSAGE_FROM_SYSTEM |
                FORMAT_MESSAGE_IGNORE_INSERTS,
                NULL,
                GetLastError(),
                MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                (LPTSTR) &lpMsgBuf,
                0,
                NULL
        );

        strLastError = Operation + (LPCTSTR)lpMsgBuf;

        //quita CR/LF
        strLastError.TrimRight();
```

```
	Index = m_ResultsList.InsertString(-1, strLastError);
	ScrollToBottomOfListBox(Index);
	LocalFree(lpMsgBuf);
}


//muestra los datos enviados por el dispositivo
void CUsbhidiocDlg::DisplayReceivedData()
{
	CString cadena1 = "";
	CString cadena2 = "";
	int Index = 0;

	if ((vector[1] && 0xF0) == izq)//SENTIDO del motor 1
		cadena1.Format((" %s\t %d\t %d"),"Izq",(unsigned
char)vector[2],(vector[3]*0x100)+vector[4]);
	else
		cadena1.Format((" %s\t %d\t %d"),"Der",(unsigned
char)vector[2],(vector[3]*0x100)+vector[4]);

	cadena1.Insert(cadena1.GetLength(),"\t\t ");

	if ((vector[1] && 0xFF) == izq)	//SENTIDO del motor 2
		cadena2.Format(("%s\t%d\t %d"),"Izq",(unsigned
char)vector[5],(vector[6]*0x100)+vector[7]);
	else
		cadena2.Format(("%s\t%d\t %d"),"Der",(unsigned
char)vector[5],(vector[6]*0x100)+vector[7]);

	cadena1.Insert(cadena1.GetLength(),cadena2);

	Index = m_BytesReceived.InsertString(-1, cadena1);
	m_BytesReceived.SetCurSel( Index );
	m_BytesReceived.SetCurSel( -1 );
```

```
}


void CUsbhidiocDlg::ScrollToBottomOfListBox(USHORT Index)
{
	m_ResultsList.SetCurSel( Index );
	m_ResultsList.SetCurSel( -1 );
}

UINT ControlLecturaUSB( LPVOID pParam )
{
	int Result;

	Result = WaitForSingleObject(hEventObject, tiempo); //esperamos por la
llegada de datos

	switch (Result)
	{
	case WAIT_OBJECT_0:
		{
		SetEvent(hEventoUSB);//indicamos fin del proceso
		break;
		}
	case WAIT_TIMEOUT:
		{
		CloseHandle(ReadHandle);
		CloseHandle(DeviceHandle);
		DeviceDetected = FALSE;

		for(int i=1;i<LongitudTrama;i++)
			InputReport[i]=0;

		InputReport[0]=5; //error al acceder al dispositivo
		SetEvent(hEventoUSB);//indicamos fin del proceso
```

```
                break;
                }
        default:
                {
                Result = CancelIo(ReadHandle);

                CloseHandle(ReadHandle);
                CloseHandle(DeviceHandle);
                DeviceDetected = FALSE;
                for(int i=1;i<LongitudTrama;i++)
                        InputReport[i]=0;

                InputReport[0]=5; //error al acceder al dispositivo
                SetEvent(hEventoUSB);//indicamos fin del proceso
                break;
                }
        }
        ResetEvent(hEventObject); //lectura

        return 0;
}

/***************************************RUTINA DEL
THREAD*********************************/
UINT ControlThread( LPVOID pParam ){

        Pipe pipe;
        int Result;
        hEventoUSB = CreateEvent(NULL,
                                true,
                                true,
                                "USB");
```

```
while((pipe.activo() == true))
{
	if ((pipe.leer(vector,LongitudTrama) == true))
	{
		switch(vector[0])
		{
		case 0: //se pretende salir
			{
			SendMessage(Ventana,
					WM_CERRAR,
					NULL,
					NULL);
			AfxEndThread(0);
			}
		case 1: //mensajes desde java hacia USB
			{
			ResetEvent(hEventoUSB);		//iniciamos el evento
(avisa de la finalización)

			tiempo = calculaTiempo(vector[LongitudTrama-
2],vector[LongitudTrama-1]);
			//calculamos el tiempo a esperar

			SendMessage(Ventana, //avisamos de que ha llegado
informacion
					WM_PIPE_ACTIVO,
					NULL,
					NULL);

			Result = WaitForSingleObject (hEventoUSB, tiempo);

			switch (Result)
			{
				default:
```

```
                                case WAIT_OBJECT_0:
                                {
                                        pipe.escribir(InputReport,LongitudTrama);
//enviamos el resultado
                                        break;
                                }


                                case WAIT_TIMEOUT:
                                {
                                        Result = CancelIo(ReadHandle);

                                        CloseHandle(ReadHandle);
                                        CloseHandle(DeviceHandle);

                                        DeviceDetected = FALSE;

                                        InputReport[0]=4;//indicamos error en la
aplicacion
                                        pipe.escribir(InputReport,LongitudTrama);
                                        break;
                                }
                        }
                        }
                }
        }
        else
        {
                SendMessage(Ventana, //enviar mensaje de finalizacion
                                WM_CERRAR,
                                NULL,
                                NULL);
                AfxEndThread(0);
        }
```

```
            pipe.ConectaPipe(INFINITE);//nos quedamos a la espera
        }
        //no hay comunicacion con el otro extremo del pipe
        SendMessage(Ventana, //enviar mensaje de finalizacion
                            WM_CERRAR,
                            NULL,
                            NULL);
        AfxEndThread(0);
        return 0;
}


long calculaTiempo(char alto, char bajo){
  return (((alto * 0x100) + bajo + 5)*1000); //damos 5 segundos de margen
}
```

pfc_main.asm

```
;       Autor:              Ruperto J. Hdez. Saiz
;       Revision:           3.0
;       Fecha:              Agosto 2004
;       Configuration Bits: H4 Oscillator, WDT Off, Power up timer off
;

 ifdef __16C765             ; trabaja tanto con 16c745 como 16c765
  #include <p16c765.inc>    ; cambia el microcontrolador en el proyecto
 endif
 ifdef __16C745
  #include <p16c745.inc>
 endif

#include "usb_defs.inc"

  errorlevel -302 ; lanza el mensaje "register not in bank0, check page bits"
  __CONFIG _H4_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF

unbanked    udata_shr
W_save              res     1

bank0 udata
Status_save res     1       ; registros para guardar el contexto
PCLATH_save         res     1
FSR_save    res     1

outbuffer       res     8       ; buffer para los datos recibidos del Host


Npasos_motor1       res 2   ;almacena el contador de pasos del 1er motor
Npasos_motor2       res 2   ;almacena el contador de pasos del 2ndo motor
```

```
contador1           res 1   ;cuenta el número de interrupciones de TMR0
contador2           res 1   ;cuenta el número de interrupciones de TMR1

giro1               res 2   ;+0 : indica el paso a efectuar en el motor 1
                                    ;+1 : indica el sentido del giro
giro2               res 2   ;+0 : indica el paso a efectuar en el motor 2
                                    ;+1 : indica el sentido del giro
ultimoSentido       res 1
tempSentido         res 1

flag_motores res 1  ;flag indicador del estado del motor, tiene la forma:
;       0 | envia1 | paso1 | ultimo1 ||| 0 | envia2 | paso2 | ultimo2
;       b7
                        b0
;       envia N:
;               0 : No ha terminado de girar
;               1 : Ha terminado de girar, y ya se puede enviar confirmacion
;       paso N:
;               0 : En la última ejecución del timer se dio un semi-paso
;               1 : En la última ejecución del timer se dio un paso
;
;       ultimo N:
;               0 : Indica que NO se ha llegado a la situación final (desactivar la
;salida)
;               1 : En esta iteracion no se debe dar paso, sino desactivar el puerto
;del motor

        extern InitUSB
        extern PutEP1
        extern GetEP1
        extern ServiceUSBInt
        extern CheckSleep
        extern RemoteWakeup
```

```
STARTUP    code
        pagesel       main
        goto   main
        nop
InterruptServiceVector  ;inicio del vector de interrupción
        movwfW_save
        movf   STATUS,W
        clrf     STATUS
        movwfStatus_save  ; guarda STATUS
        movf   PCLATH,w
        movwfPCLATH_save       ; guarda PCLATH
        movf   FSR,w
        movwfFSR_save    ; guarda FSR

; *************************************************************
; Interrupt Service Routine (ISR)
; En ella se pregunta a los distintos componentes, hasta hallar
; el que generó la interrupción
; *************************************************************
        pagesel Process_ISR

Process_ISR
        btfsc   INTCON,T0IF                ; Timer 0
        goto continuaISRTMR0
        goto terminaISRTMR0

continuaISRTMR0
        BANKSEL    INTCON
        bcf      INTCON,T0IF                        ;eliminamos la señalizacion de
interrupcion  de TMR0

        btfsc flag_motores,6
        goto terminaISRTMR0
```

```
        BANKSEL contador1           ;evita entradas erróneas
        movf contador1,w
        addlw 0x00
        btfsc STATUS,Z
        goto terminaISRTMR0
        decfsz contador1,1          ;comprobamos si llegó a cero
        goto reiniciaTimer1
        goto conto8

reiniciaTimer1                      ;volvemos a asignar los datos al timer
        PAGESEL iniciaTimer0
        call iniciaTimer0
        PAGESEL terminaISRTMR0
        goto terminaISRTMR0

conto8
        BANKSEL contador1
        movlw 0x40
        movwf contador1             ;reponemos el valor del contador
        pagesel IntMotor1
        call IntMotor1
terminaISRTMR0

        BANKSEL    PIR1             ;USB
        pagesel    ServiceUSBInt
        btfsc   PIR1,USBIF
        call    ServiceUSBInt

        btfsc   PIR1,TMR1IF         ;TMR1
        goto continuaISRTMR1
        goto terminaISRTMR1

continuaISRTMR1
        BANKSEL    PIR1
```

```
        bcf     PIR1,TMR1IF         ;eliminamos la señalizacion de interrupcion de
                                    ;TMR1
        btfsc flag_motores,2
        goto terminaISRTMR1

        BANKSEL contador2
        pagesel conto82
        decfsz contador2,1          ;comprobamos si llegó a cero
        goto reiniciaTimer2
        goto conto82

reiniciaTimer2
        pagesel iniciaTimer1
        call iniciaTimer1
        pagesel terminaISRTMR1
        goto terminaISRTMR1

conto82
        banksel contador2
        movlw 0x08
        movwf contador2                 ;reponemos el valor del contador
        pagesel IntMotor2
        call IntMotor2
terminaISRTMR1

; *********************************************************************
; End ISR, restaura el contexto guardado y vuelve al programa
; principal
; *********************************************************************
EndISR
        clrf    STATUS
        movf    FSR_save,w          ; restaura FSR
        movwfFSR
        movf    PCLATH_save,w       ; restaura PCLATH
```

```
        movwfPCLATH
        movf   Status_save,w      ; restaura Status
        movwfSTATUS
        swapf W_save,f            ; restaura W manteniendo STATUS
        swapf W_save,w
        retfie
code

;/*****************RUTINAS DEL MOTOR*****************************/

;gestión del motor1
IntMotor1                               ;su salida será por la parte alta del puerto
        BANKSEL flag_motores
        btfsc flag_motores,4      ;comprobamos que no es el último estado
        goto ultimo_paso_motor_1

        btfss flag_motores,5      ;atendemos a si se nos pide un paso ó un medio-
                                  ;paso
        goto paso_motor1

medio_paso_motor1
        BANKSEL    PORTB
        movf PORTB,w
        andlw 0x0F                      ;despejamos la parte alta del puerto
        movwf PORTB                             ;lo sacamos al puerto B

        BANKSEL giro1             ;calculo del medio-paso
        movf giro1+0,w
        andwf giro1+1,w

        BANKSEL PORTB
        iorwf PORTB,1             ;lo sacamos al puerto B

        BANKSEL giro1
```

```
        movlw 0xF0
        xorwf giro1+1,1                 ;almacenamos el valor del sentido

        BANKSEL flag_motores
        bcf flag_motores,5          ;indicamos que hemos hecho un medio-paso

        goto reasignar_Tiempo1

paso_motor1
        BANKSEL    PORTB
        movf PORTB,w
        andlw 0x0F                      ;despejamos la parte alta del puerto
        movwf PORTB                             ;lo sacamos por el puerto B

        BANKSEL giro1                   ;calculo del paso
        movf giro1+0,w
        xorwf giro1+1,w
        movwf giro1+0                   ;almacenamos el valor

        BANKSEL PORTB
        iorwf PORTB,1                   ;lo sacamos al puerto B

        BANKSEL flag_motores
        bsf flag_motores,5          ;indicamos que hemos hecho un paso

descuenta1
        BANKSEL Npasos_motor1               ;descuenta el paso realizado (2
bytes)
        movlw 0x00
        addwf Npasos_motor1+1,w
        BANKSEL STATUS
        btfsc STATUS,2
        goto descuenta_es_cero1
        decfsz Npasos_motor1+1,1
```

```
        goto reasignar_Tiempo1

descuenta_es_cero1                      ;hemos llegado a 0 en la parte baja
del número de pasos
        movf Npasos_motor1+0,w
        addlw 0
        BANKSEL STATUS
        btfss STATUS,2
        goto descuenta_reempieza1       ;la parte alta no es cero, por tanto, seguir
                                        ;contando

        BANKSEL flag_motores            ;tanto la parte baja como la alta son cero:
        bsf flag_motores,4              ;indicamos que es el último paso
        goto reasignar_Tiempo1

descuenta_reempieza1
        BANKSEL Npasos_motor1
        decf Npasos_motor1+0,1          ;decrementamos el número de pasos
(nivel alto)
        movlw 0xFF
        movwf Npasos_motor1+1           ;reiniciamos la parte baja del contador
        goto reasignar_Tiempo1

ultimo_paso_motor_1
        BANKSEL INTCON          ;deja el motor en reposo
        bcf     INTCON,T0IE     ;desabilitamos las interrupciones de este timer

        BANKSEL    PORTB
        movf PORTB,w
        andlw 0x0F
        movwf PORTB

        BANKSEL flag_motores    ;la operación en este motor ha terminado
        bsf flag_motores,6
```

```
        goto fin_IntMotor1

reasignar_Tiempo1       ;volvemos a asignar los datos al timer
        BANKSEL outbuffer
        movf outbuffer+1,w ;velocidad
        BANKSEL    TMR0
        movwf TMR0

        movlw b'00000111' ;asignación de prescaler
        BANKSEL    OPTION_REG
        iorwf OPTION_REG,1
fin_IntMotor1
        return

;gestión del motor 2
IntMotor2                               ;su salida será por la parte baja del puerto
        BANKSEL flag_motores
        btfsc flag_motores,0        ;comprobamos que no es el último estado
        goto ultimo_paso_motor_2

        btfss flag_motores,1        ;atendemos a si se nos pide un paso ó un medio-
                                    ;paso
        goto paso_motor2

medio_paso_motor2
        BANKSEL    PORTB
        movf PORTB,w
        andlw 0xF0                  ;despejamos la parte baja del puerto
        movwf PORTB                 ;lo sacamos por el puerto B
        BANKSEL giro2               ;cálculo del medio-paso
        movf giro2+0,w
        andwf giro2+1,w
```

```
        BANKSEL PORTB
        iorwf PORTB,1                   ;lo sacamos al puerto B

        BANKSEL giro2
        movlw 0x0F
        xorwf giro2+1,1                 ;almacenamos el valor

        BANKSEL flag_motores
        bcf flag_motores,1          ;indicamos que hemos hecho un medio-paso

        goto reasignar_Tiempo2

paso_motor2
        BANKSEL    PORTB
        movf PORTB,w
        andlw 0xF0                  ;despejamos la parte baja del puerto
        movwf PORTB                     ;lo sacamos por el puerto B

        BANKSEL giro2               ;cálculo del paso
        movf giro2+0,w
        xorwf giro2+1,w
        movwf giro2+0               ;almacenamos el valor

        BANKSEL PORTB
        iorwf PORTB,1               ;lo sacamos al puerto B

        BANKSEL flag_motores
        bsf flag_motores,1  ;indicamos que hemos hecho un paso

descuenta2                          ;descuenta el paso realizado (2 bytes)
        BANKSEL Npasos_motor2
        movlw 0x00
        addwf Npasos_motor2+1,w
        BANKSEL STATUS
```

```
        btfsc STATUS,2
        goto descuenta_es_cero2         ;la parte baja ya es cero
        decfsz Npasos_motor2+1,1
        goto reasignar_Tiempo2

descuenta_es_cero2                      ;hemos llegado a 0 en la parte baja
                                        ;del número de pasos
        movf Npasos_motor2+0,w
        addlw 0
        BANKSEL STATUS
        btfss STATUS,2
        goto descuenta_reempieza2       ;la parte alta no es cero, por tanto, seguir
                                        ;contando

        BANKSEL flag_motores            ;tanto la parte baja como la alta son cero:
        bsf flag_motores,0
        goto reasignar_Tiempo2

descuenta_reempieza2                    ;la parte alta no es cero, por tanto, seguir
                                        ;contando
        BANKSEL Npasos_motor2
        decf Npasos_motor2+0,1          ;decrementamos el número de pasos
(nivel alto)
        movlw 0xFF
        movwf Npasos_motor2+1           ;reiniciamos el nivel bajo
        goto reasignar_Tiempo2

ultimo_paso_motor_2
        BANKSEL T1CON           ;deja el motor en reposo
        bcf     T1CON,TMR1ON    ;desactivamos el timer

        BANKSEL    PIE1
        bcf PIE1,TMR1IE         ;deshabilitamos las interrupciones del Timer1
```

```
        BANKSEL    PORTB
        movf PORTB,w
        andlw 0xF0
        movwf PORTB

        BANKSEL flag_motores    ;la operación en este motor ha terminado
        bsf flag_motores,2

        goto fin_IntMotor2

reasignar_Tiempo2
        banksel outbuffer    ;volvemos a asignar los datos al timer
        movf outbuffer+4,w ;velocidad
        BANKSEL    TMR1H
        movwf TMR1H
        BANKSEL    TMR1L
        movlw 0xFF
        movwf TMR1L

        BANKSEL    T1CON
        movlw b'00110101' ;Prescaler
        movwf T1CON

fin_IntMotor2
        return

; *****************************************************************
;                                   Programa principal
; *****************************************************************

main
        movlw.30                ; retardo de 16us antes de comenzar la
configuración
        movwfW_save
```

```
        decfszW_save,f
        goto   $-1

        BANKSEL    OPTION_REG
        bsf     OPTION_REG,NOT_RBPU        ; deshabilita los pull-ups

        BANKSEL    OPTION_REG
        bcf OPTION_REG,T0CS          ;TMR0 modo Timer
        bcf OPTION_REG,PSA           ;divisor de frecuencia al TMR0

        BANKSEL    INTCON
        bcf INTCON,T0IE              ;deshabilitar las interrupciones del TMR0

        BANKSEL    PIE1
        bcf PIE1,TMR1IE              ;deshabilitar las interrupciones del TMR1

        BANKSEL    INTCON
        bcf     INTCON,T0IF                ;eliminamos la señalizacion de
interrupcion del TMR0
        BANKSEL PIR1
        bcf     PIR1,TMR1IF                ;eliminamos la señalizacion de
interrupcion del TMR1

        BANKSEL giro1                ;asignamos un paso inicial a ambos
motores y un sentido
        movlw 0xA0                   ;motor 1
        movwf giro1+0
        movlw 0xC0
        movwf giro1+1                ;sentido
        BANKSEL giro2
        movlw 0x0A                   ;motor 2
        movwf giro2+0
        movlw 0x0C
        movwf giro2+1                ;sentido
```

```
        BANKSEL contador1
        clrf contador1              ;asignamos valores conocidos inicialmente
        clrf contador2

        clrf ultimoSentido

        BANKSEL flag_motores    ;configuracion inicial de los motores
        movlw 0x00
        movwf flag_motores

        BANKSEL    TRISB            ;todos los pines del puerto B como salida
        clrf PORTB
        BANKSEL    PORTB                ;inicialmente apagado
        movlw 0
        movwf PORTB

        pagesel    InitUSB
        call    InitUSB             ;inicializa el sistema USB
        ConfiguredUSB               ;espera hasta que se complete el proceso
de configuración

        BANKSEL    INTCON
        bcf    INTCON,T0IF

LoopForData
        pagesel    GetEP1
        banksel outbuffer
        bankisel outbuffer
        movlw outbuffer
        movwf FSR
        movlw 0x8
        call    GetEP1              ;intentammos obtener datos del PC
```

```
        pagesel     ProcessOUTBuffer
        btfsc   STATUS,C            ;SI llegó un paquete
        call    ProcessOUTBuffer;lo procesamos

compruebaEnviar             ;comprueba si terminó el proceso de ambos motores
        pagesel LoopForData
        BANKSEL flag_motores
        btfss flag_motores,6        ;¿ha terminado el primer motor?
        goto LoopForData

        btfss flag_motores,2        ;¿ha terminado el segundo motor?
        goto LoopForData            ;NO ha terminado

;En caso de contar con dispositivos con los que saber cómo se ha comportado
;cada motor, aquí se incluirían las rutinas para obtener esta info y enviarla al PC
;en nuestro caso repetiremos lo que nos fue enviado

Pkt2PC
        pagesel     PutEP1
        bankisel outbuffer
        banksel outbuffer
        movlw outbuffer
        movwf FSR                   ;localización del buffer de datos

        movlw 0x8                   ;8 bytes

        call    PutEP1              ;envía los datos al sistema principal
        pagesel LoopForData

        BANKSEL flag_motores    ;desactivamos la opcion de volver a enviar
        bcf     flag_motores,2
        goto LoopForData
```

```
ProcessOUTBuffer
; outbuffer+0 = Indica el sentido de giro (Motor 1/Motor 2) Sentido 0:Izquierda,
        ;1:Derecha
; outbuffer+1 = velocidad motor 1
; outbuffer+2 = Parte alta del número de pasos motor 1
; outbuffer+3 = Parte baja del número de pasos motor 1
; outbuffer+4 = velocidad motor 2
; outbuffer+5 = Parte alta del número de pasos motor 2
; outbuffer+6 = Parte baja del número de pasos motor 2

Timermotor1             ;configuramos el timer para el primer motor
            ;Temporizacion = 4 · Tosc · (valor cargado en TMR0{en
            ;complemento A2}) · (Rango del divisor)
            ;Valor a cargar en TMRO = (Temporizacion/4 · Tosc) · (Rango del
            ;Divisor), siendo Tosc = 1/6MHz

        BANKSEL INTCON
        bcf     INTCON,T0IE     ;deshabilitamos las interrupciones de este timer

        BANKSEL contador1
        movlw 0x40
        movwf contador1                 ;número de interrupciones TMR0
necesarias

        banksel outbuffer
        movf outbuffer+2,w      ;parte alta de los pasos del motor 1
        banksel Npasos_motor1
        movwf Npasos_motor1+0
        banksel outbuffer
        movf outbuffer+3,w      ;parte baja de los pasos del motor 1
        banksel Npasos_motor1
        movwf Npasos_motor1+1

        BANKSEL flag_motores    ;configuracion inicial de los motores
```

```
        bcf flag_motores,4          ;NO es la ultima iteracion
        bsf     flag_motores,5              ;fijamos que lo anterior fue un paso
        bcf flag_motores,6          ;NO enviar

        BANKSEL outbuffer           ;asignación de sentido
        movf outbuffer,w
        BANKSEL ultimoSentido
        movwf tempSentido
        movf ultimoSentido,w
        addwf tempSentido,F
        btfss tempSentido,4         ;si el 4º bit es un "1" es que hay que cambiar de
                                    ;sentido
        goto configuraGiro1

        movlw 0xF0
        BANKSEL giro1
        xorwf giro1+1,1                     ;invertimos el sentido
        movlw 0x10
        BANKSEL ultimoSentido
        xorwf ultimoSentido,1       ;invertimos el valor del último sentido (para la
                                    ;próxima)
configuraGiro1
        BANKSEL giro1               ;sacamos por el puerto el último paso dado
        movf giro1+0,w
        BANKSEL    PORTB
        iorwf PORTB,1

        banksel outbuffer           ;comprobamos si hay "algo" que mover
        movf outbuffer+2,w
        addwf outbuffer+3,w
        btfss STATUS,C
        btfss STATUS,Z
        goto iniciarPrimerTimer
        goto noiniciarPrimerTimer
```

```
iniciarPrimerTimer
        PAGESEL iniciaTimer0
        call iniciaTimer0
        goto Timermotor2

noiniciarPrimerTimer
        bsf flag_motores,6          ;indicamos que termino

Timermotor2
        BANKSEL contador2
        movlw 0x08
        movwf contador2             ;número de interrupciones TMR1 necesarias

        banksel outbuffer
        movf outbuffer+5,w          ;parte alta del paso del motor 2
        banksel Npasos_motor2
        movwf Npasos_motor2+0
        banksel outbuffer
        movf outbuffer+6,w          ;parte baja del paso del motor 2
        banksel Npasos_motor2
        movwf Npasos_motor2+1

        BANKSEL flag_motores        ;configuracion inicial de los motores
        bcf flag_motores,0          ;NO es la ultima iteracion
        bsf     flag_motores,1              ;fijamos que la anterior fue un paso
        bcf flag_motores,2          ;NO enviar

        BANKSEL outbuffer           ;sentido
        movf outbuffer,w
        BANKSEL ultimoSentido
        movwf tempSentido
        movf ultimoSentido,w
        addwf tempSentido,F
```

```
        btfss tempSentido,0     ;si el primer bit es un "1" es que hay que cambiar
                                ;de sentido
        goto configuraGiro2

        movlw 0x0F
        BANKSEL giro2
        xorwf giro2+1,1                 ;invertimos el sentido
        movlw 0x01
        BANKSEL ultimoSentido
        xorwf ultimoSentido,1   ;invertimos el valor del último sentido (para la
                                ;próxima)
configuraGiro2
        BANKSEL giro2           ;sacamos por el puerto el último paso dado
        movf giro2+0,w
        BANKSEL    PORTB
        iorwf PORTB,1

        banksel outbuffer       ;comprobamos si hay "algo" que mover
        movf outbuffer+5,w
        addwf outbuffer+6,w
        btfss STATUS,C
        btfss STATUS,Z
        goto iniciarSegundoTimer
        goto noiniciarSegundoTimer

iniciarSegundoTimer
        PAGESEL iniciaTimer1
        call iniciaTimer1
        goto final
noiniciarSegundoTimer
        bsf flag_motores,2

final
        return
```

```
;funciones:
iniciaTimer1                ;motor2
        banksel outbuffer
        movf outbuffer+4,w ;velocidad
        BANKSEL    TMR1H
        movwf TMR1H
        BANKSEL    TMR1L
        movlw 0xFF
        movwf TMR1L

        BANKSEL    T1CON
        movlw b'00110101' ;prescaler
        movwf T1CON

        BANKSEL    PIE1
        bsf PIE1,TMR1IE          ;Habilitamos las interrupciones del Timer1
        return

iniciaTimer0                ;motor1
        BANKSEL outbuffer
        movf outbuffer+1,w ;velocidad
        BANKSEL    TMR0
        movwf TMR0

        movlw b'00000111'
        BANKSEL    OPTION_REG
        iorwf OPTION_REG,1      ;prescaler

        BANKSEL    INTCON
        movlw b'10100000'
        iorwf INTCON,1          ;Habilitamos las interrupciones del Timer0
        return
end
```

descript.asm

```
;                       DESCRIPT.ASM
;                       DESCRIPTORES USB
;
; Contiene los descriptores del proyecto
;
;               Author:         Ruperto Hernández Sáiz
;               Revision:       2.1
;               Fecha:          Junio 2004
;
;       incluye:
;               P16C7X5.inc     Rev 1.00
;               usb_defs.inc Rev 1.10

 ifdef __16C765
  #include <p16c765.inc>
 endif
 ifdef __16C745
  #include <p16c745.inc>
 endif

#include "usb_defs.inc"

USBBANK    code
        global Config_desc_index
        global Report_desc_index
        global Descriptions
        global string_index
        global DeviceDescriptor
        global ReportDescriptor
        global ReportDescriptorLen
        global String0
        global String0_end
```

```
        global StringDescriptions
        global HID_Descriptor

        extern EP0_start
        extern          temp   ; temp var used in get config index
        extern temp2 ; another temp, in bank2

; *****************************************************************
; Dado el índice del descriptor de configuración, devuelve la
; dirección en la que este comienza
; *****************************************************************
Config_desc_index
        movwftemp
        movlw HIGH CDI_start
        movwfPCLATH
        movlw low  CDI_start
        addwf temp,w
        btfsc   STATUS,C
        incf    PCLATH,f
        movwfPCL
CDI_start      ; calcula el offset para cada descriptor de configuración
        retlw   low  Config1  ; desde el principio de la tabla
        retlw   high Config1

; *****************************************************************
; Dado el índice del descriptor de informe, devuelve la
; dirección en la que este comienza
; *****************************************************************
Report_desc_index
        movwftemp
        movlw HIGH RDI_start
        movwfPCLATH
        movlw low RDI_start
        addwf temp,w
```

```
        btfsc   STATUS,C
        incf    PCLATH,f
        movwf   PCL
RDI_start       ; calcula el offset para cada descriptor de informe
        retlw   low  ReportDescriptorLen  ;desde el principio de la tabla
        retlw   high ReportDescriptorLen

; *****************************************************************
; Esta tabla es solicitada por el sistema principal una vez ha sido
; reiniciado el periférico USB. Esta tabla define el tamaño máximo
; del paquete que EP0 puede utilizar. Los valores asignados a estos
; campos son dependientes de la aplicación a realizar.
; *****************************************************************

Descriptions
        banksel         EP0_start
        movf    EP0_start+1,w
        movwf   PCLATH
        movf    EP0_start,w
        movwf   PCL

DeviceDescriptor
StartDevDescr
        retlw   0x12            ; bLengthLength de este descruptor
        retlw   0x01            ; bDescType El descriptor de este dispositivo
        retlw   0x00            ; bcdUSBUSB revision 1.10 (byte bajo)
        retlw   0x01            ; byte alto
        retlw   0x00            ; bDeviceClasszero indica que cada interfaz trabaja de
                                ;forma independiente
        retlw   0x00            ; bDeviceSubClass
        retlw   0x00            ; bDeviceProtocol
        retlw   0x08            ; bMaxPacketSize0
        retlw   0x00            ; idVendor
        retlw   0x00            ;   byte alto
```

```
        retlw   0x00            ; idProduct
        retlw   0x00
        retlw   0x00            ; bcdDevice
        retlw   0x00
        retlw   0x00            ; iManufacturer
        retlw   0x00            ; iProduct
        retlw   0x00            ; iSerialNumber
        retlw   NUM_CONFIGURATIONS          ; bNumConfigurations

; ******************************************************************
; Esta tabla es solicitada por el sistema principal una vez que el
; dispositivo ha sido direccionado
; ******************************************************************
Config1
        retlw   0x09            ; bLengthLength de este descruptor
        retlw   0x02            ; bDescType2 = CONFIGURATION
        retlw   EndConfig1 - Config1
        retlw   0x00
        retlw   0x01            ; bNumInterfacesNumber de interfaces
        retlw   0x01            ; bConfigValue
        retlw   0x04            ; iConfigString Indice de la cadena para esta
configuración
        retlw   0x80            ; bmAttributesattributes - obtiene la potencia del cable
USB
        retlw   0x32            ; MaxPowerself
Interface1
        retlw   0x09            ; longitud del descriptor
        retlw   INTERFACE
        retlw   0x00            ; número de interfaz
        retlw   0x00            ; configuracion alternativa
        retlw   0x02            ; número de interfaces utilizados en esta configuración
        retlw   0x03            ; interface class - asignado por USB
        retlw   0x00            ; boot device
        retlw   0x00            ; interface protocol - ninguno
```

```
        retlw   0x00            ; indice de la cadena que describe esta configuración
HID_Descriptor
        retlw   (Endpoint1 - HID_Descriptor)     ; tamaño del descriptor
        retlw   0x21            ; tipo de descriptor (HID)
        retlw   0x00
        retlw   0x01            ; HID class release number (1.00)
        retlw   0x00            ; Pais de locaclización (ninguno)
        retlw   0x01            ; número de descriptores de la clase HID
        retlw   0x22            ; Tipo de descriptor de informe (HID)
        retlw   low  (end_ReportDescriptor - ReportDescriptor)
        retlw   high (end_ReportDescriptor - ReportDescriptor)
Endpoint1
        retlw   0x07            ; longitud del descriptor
        retlw   ENDPOINT
        retlw   0x81            ; EP1, Entrada
        retlw   0x03            ; Interrupt (modo Interrupcion)
        retlw   0x08            ; tamaño máximo del paquete (8 bytes), byte bajo
        retlw   0x00            ; tamaño máximo del paquete (8 bytes), byte alto
        retlw   0x0A            ; intervalo de polling (10ms)
Endpoint2
        retlw   0x07            ; longitud del descriptor
        retlw   ENDPOINT
        retlw   0x01            ; EP1, Salida
        retlw   0x03            ; Interrupt (modo Interrupcion)
        retlw   0x08            ; tamaño máximo del paquete (8 bytes), byte bajo
        retlw   0x00            ; tamaño máximo del paquete (8 bytes), byte alto
        retlw   0x0A            ; intervalo de polling (10ms)
EndConfig1

ReportDescriptorLen
        retlw   low  (end_ReportDescriptor-ReportDescriptor)
ReportDescriptor
      dt 006h, 000h, 0ffh         ; USAGE_PAGE (Vendor Defined Page 1)
      dt 009h, 001h               ; USAGE (Vendor Usage 1)
```

```
        dt 0a1h, 001h           ; COLLECTION (Application)
        dt 019h, 001h           ;   USAGE_MINIMUM (Vendor Usage 1)
        dt 029h, 008h           ;   USAGE_MAXIMUM (Vendor Usage 8)
        dt 015h, 000h           ;   LOGICAL_MINIMUM (0)
        dt 026h, 0ffh, 000h      ;   LOGICAL_MAXIMUM (255)
        dt 075h, 008h           ;   REPORT_SIZE (8)
        dt 095h, 008h           ;   REPORT_COUNT (8)
        dt 081h, 002h           ;   INPUT (Data,Var,Abs)
        dt 019h, 001h           ;   USAGE_MINIMUM (Vendor Usage 1)
        dt 029h, 008h           ;   USAGE_MAXIMUM (Vendor Usage 8)
        dt 091h, 002h           ;   OUTPUT (Data,Var,Abs)
        dt 0c0h                 ; END_COLLECTION
end_ReportDescriptor

StringDescriptions
        banksel     EP0_start
        movf   EP0_start+1,w
        movwfPCLATH
        movf   EP0_start,w
        movwfPCL

; *****************************************************************
; Dado el índice del descriptor de configuración, devuelve la
; la dirección de inicio del descriptor dentro de la tabla de
; descriptores de texto
; *****************************************************************
string_index
        movwftemp
        bcf     STATUS,C
        rlf     temp, f
        pagesel     langid_index
        call    langid_index
        movwftemp2
        incf    temp, f
```

```
        pagesel     langid_index
        call    langid_index
        movwftemp

        movf   temp, w
        movwfPCLATH
        movf   temp2,w
        addwf EP0_start+1,w
        btfsc   STATUS,C
        incf    PCLATH, f
        movwfPCL

langid_index
        movlwhigh langids
        movwfPCLATH
        movlwlow langids
        addwf temp, w
        btfsc   STATUS,C
        incf    PCLATH,f
        movwfPCL

langids
        retlw   low lang_1
        retlw   high lang_1
        retlw   low lang_2    ; indices de cadenas para diferentes idiomas
        retlw   high lang_2

lang_1
        retlw   low  String0  ; LangIDs
        retlw   high String0
        retlw   low  String1_l1
        retlw   high String1_l1
        retlw   low  String2_l1
        retlw   high String2_l1
```

```
        retlw   low  String3_l1
        retlw   high String3_l1
        retlw   low  String4_l1
        retlw   high String4_l1
        retlw   low  String5_l1
        retlw   high String5_l1

lang_2
        retlw   low  String0
        retlw   high String0
        retlw   low  String1_l2
        retlw   high String1_l2
        retlw   low  String2_l2
        retlw   high String2_l2
        retlw   low  String3_l2
        retlw   high String3_l2
        retlw   low  String4_l2
        retlw   high String4_l2
        retlw   low  String5_l2
        retlw   high String5_l2

String0
        retlw   low (String1_l1 - String0)   ; longitud de la cadena
        retlw   0x03   ; tipo de descriptor
        retlw   0x09   ; identificador del idioma (definido por MS 0x0409)
        retlw   0x04
        retlw   0x04   ; para otros lenguajes
        retlw   0x08
String0_end
String1_l1                      ;identificador textual del nombre de dispositivo
        retlw   String2_l1-String1_l1        ; longitud de la cadena
        retlw   0x03   ; descriptor de cadena tipo 3 (definido antes)
        retlw   'P'
        retlw   0x00
```

```
        retlw   '.'
        retlw   0x00
        retlw   'F'
        retlw   0x00
        retlw   '.'
        retlw   0x00
        retlw   'C'
        retlw   0x00
        retlw   '.'
        retlw   0x00
        retlw   ' '
        retlw   0x00
        retlw   'U'
        retlw   0x00
        retlw   'S'
        retlw   0x00
        retlw   'B'
        retlw   0x00
String2_l1                      ;identificador textual del autor
        retlw   String3_l1-String2_l1
        retlw   0x03
        retlw   'R'
        retlw   0x00
        retlw   'u'
        retlw   0x00
        retlw   'p'
        retlw   0x00
        retlw   'e'
        retlw   0x00
        retlw   'r'
        retlw   0x00
        retlw   't'
        retlw   0x00
        retlw   'o'
```

```
        retlw   0x00
        retlw   ' '
        retlw   0x00
        retlw   'H'
        retlw   0x00
        retlw   'e'
        retlw   0x00
        retlw   'r'
        retlw   0x00
        retlw   'n'
        retlw   0x00
        retlw   'a'
        retlw   0x00
        retlw   'n'
        retlw   0x00
        retlw   'd'
        retlw   0x00
        retlw   'e'
        retlw   0x00
        retlw   'z'
        retlw   0x00
String3_l1
        retlw   String4_l1-String3_l1
        retlw   0x03
        retlw   'V'
        retlw   0x00
        retlw   '1'
        retlw   0x00
        retlw   '.'
        retlw   0x00
        retlw   '0'
        retlw   0x00
        retlw   '0'
        retlw   0x00
```

```
String4_l1
        retlw   String5_l1-String4_l1
        retlw   0x03
        retlw   'U'
        retlw   0x00
        retlw   'L'
        retlw   0x00
        retlw   'P'
        retlw   0x00
        retlw   'G'
        retlw   0x00
        retlw   'C'
        retlw   0x00
String5_l1
        retlw   String6_l1-String5_l1
        retlw   0x03
        retlw   'E'
        retlw   0x00
        retlw   'U'
        retlw   0x00
        retlw   'I'
        retlw   0x00
        retlw   'T'
        retlw   0x00
        retlw   'T'
        retlw   0x00
String6_l1

String1_l2
        retlw   String2_l2-String1_l2
        retlw   0x03
        retlw   'M'
        retlw   0x00
        retlw   'i'
```

```
        retlw   0x00
        retlw   'c'
        retlw   0x00
        retlw   'r'
        retlw   0x00
        retlw   'o'
        retlw   0x00
        retlw   'c'
        retlw   0x00
        retlw   'h'
        retlw   0x00
        retlw   'i'
        retlw   0x00
        retlw   'p'
        retlw   0x00
String2_l2
        retlw   String3_l2-String2_l2
        retlw   0x03
        retlw   'P'
        retlw   0x00
        retlw   'i'
        retlw   0x00
        retlw   'c'
        retlw   0x00
        retlw   '1'
        retlw   0x00
        retlw   '6'
        retlw   0x00
        retlw   'C'
        retlw   0x00
        retlw   '7'
        retlw   0x00
        retlw   '4'
        retlw   0x00
```

```
        retlw   '5'
        retlw   0x00
        retlw   ' '
        retlw   0x00
        retlw   'U'
        retlw   0x00
        retlw   'S'
        retlw   0x00
        retlw   'B'
        retlw   0x00
        retlw   ' '
        retlw   0x00
        retlw   'P'
        retlw   0x00
        retlw   'F'
        retlw   0x00
        retlw   'C'
        retlw   0x00
String3_l2
        retlw   String4_l2-String3_l2
        retlw   0x03
        retlw   'V'
        retlw   0x00
        retlw   '1'
        retlw   0x00
        retlw   '.'
        retlw   0x00
        retlw   '0'
        retlw   0x00
        retlw   '0'
        retlw   0x00
String4_l2
        retlw   String5_l2-String4_l2
        retlw   0x03
```

```
        retlw   'U'
        retlw   0x00
        retlw   'L'
        retlw   0x00
        retlw   'P'
        retlw   0x00
        retlw   'G'
        retlw   0x00
        retlw   'C'
        retlw   0x00
String5_l2
        retlw   String6_l2-String5_l2
        retlw   0x03
        retlw   'E'
        retlw   0x00
        retlw   'U'
        retlw   0x00
        retlw   'I'
        retlw   0x00
        retlw   'T'
        retlw   0x00
        retlw   'T'
        retlw   0x00
String6_l2

        end
```