4 DISEÑO DE UN SOFTWARE PARA EL CONTROL DE DISPOSITIVOS DOMÓTICOS SIGUIENDO EL ESTÁNDAR EIB.

En este capítulo se explica cuál es el método seguido en el diseño de un software para el control de dispositivos domóticos siguiendo el estándar EIB. Se abordan temas como: las características previas, configuraciones de los dispositivos, partes del sistema, funcionalidad y características tecnológicas.

4.1 MONTAJE Y CONFIGURACIÓN DE LA INSTALACIÓN DE PRUEBAS.

En este apartado, se detallará el método seguido para la puesta a punto de la instalación de prueba. Se explicará su cableado y conexionado, dejando para el anexo2 la explicación de la configuración de la placa con el software ETS.

4.1.1 Conexionado de dispositivos.

Lo primero que se realizo fue conectar los equipos y comprobar que estos funcionan correctamente de forma manual. Los pasos que se realizaron fueron los siguientes:

1. Se colocaron y sujetaron los dispositivos domóticos (NTA6F16H+COM y TS2) en una placa.
2. En la parte trasera de la placa, se coloco el cable bus tratando de maximizar la longitud de este, para lo que se procedió enrollándolo. Tal y como se muestra en la siguiente figura.

Figura 4.1. Cableado del bus en la placa de pruebas.

3. Se conecta la parte de alimentación del dispositivo NTA6F16H+COM a la red eléctrica común. Para ello, se ha de unir la fase, el neutro y la tierra del dispositivo con los de la red, mediante un cable con enchufe común. En la figura 2.1, puede verse cual es esta conexión en el dispositivo.

4. Se conecta el fase de la red eléctrica a la bombilla y el neutro a uno de los canales del NTA6F16H+COM. Tal y como puede verse en la figura 4.2.

5. Se conecta el NTA6F16H+COM al bus a través del conector que incorpora este dispositivo. Este conector tiene dos partes a la que se conectan cada uno de los conductores del cable bus[1].

6. El otro extremo del cable bus se conecta el interfaz TS2 de la misma forma que se conecta el dispositivos anterior. En la figura 4.2, se puede ver donde están los conectores al bus de los dos dispositivos.

7. El TS2 tiene dos canales y cuatro cables. A cada canal le pertenecerán dos cables. Dos de los cuatro cables son iguales (negro y blanco), con lo cual pertenecerán cada uno de ellos a un canal y serán los cable de señal. Los otros dos cables serán los cables de tierra y cada uno de ellos se relaciona en proximidad a los anteriores. Los cables de un canal se conectarán a los elementos del interruptor normal para que, haciendo uso de este y con la configuración adecuada, se pueda utilizar como los interruptores comunes.

8. Una vez que se conectan los equipos EIB al bus habrá que conectar la parte de interfaz RS-232 del NTA6F16H+COM al puerto serie de un PC.

La siguiente figura nos muestra como quedó el montaje de la instalación de pruebas.

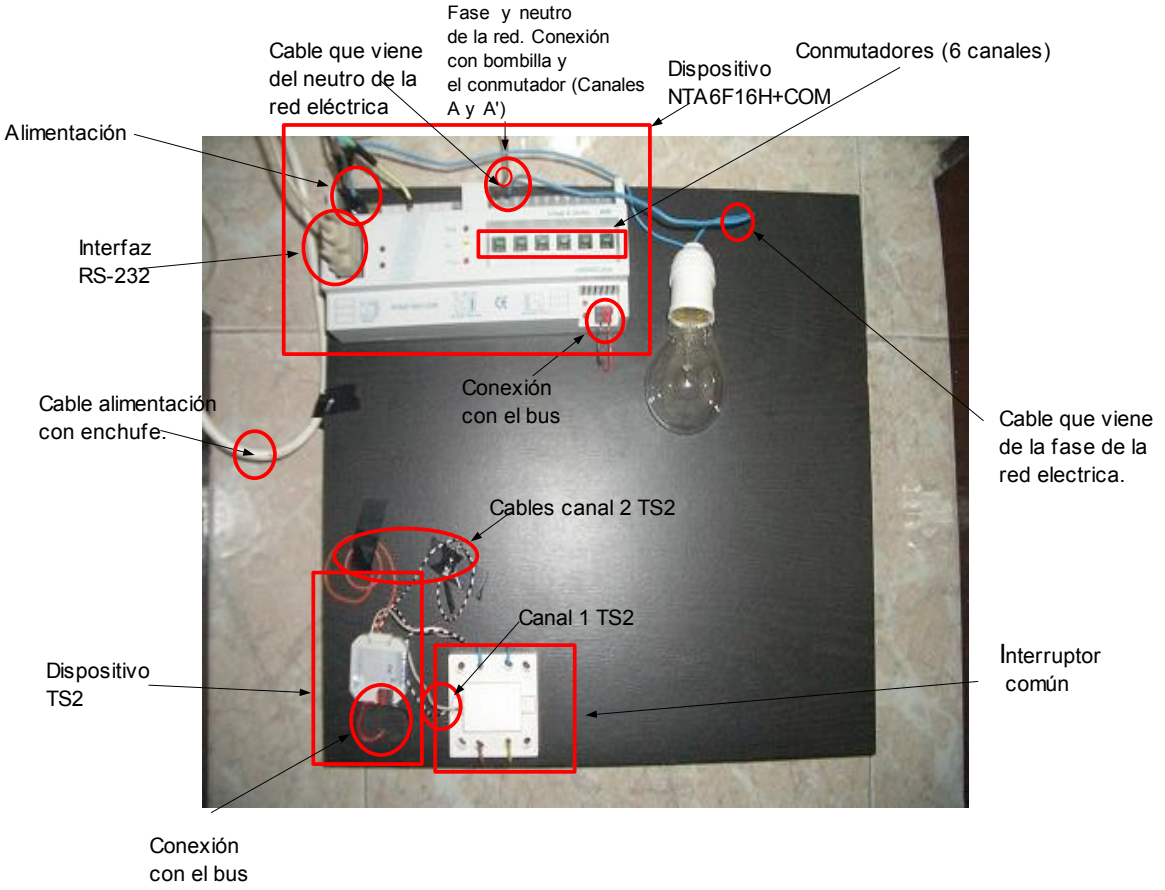

Figura 4.2: Montaje del instalación de pruebas.

1 El cable bus es un cable de pares, con un total de dos pares. De esos dos pares, se usaran para señal dos, el rojo y el negro, siendo el rojo el positivo y el negro el negativo. Los otros dos (amarillo y blanco) se utilizarán para apantallar.

Con estos pasos se ha logrado el montaje del instalación de pruebas. Esta instalación no funciona manualmente, es decir, si se actúa sobre el interruptor este no accionará el contacto de la bombilla. Esto es así porque no se han configurado los dispositivo, por lo que no se han establecido las relaciones necesarias entre ellos. La alimentación llega correctamente pero, hasta que no se establezcan las relaciones, no se sabrá a que dispositivo se han de mandar los datagramas y, por tanto, no se sabrá quien ha de actuar ante los mismos. Sí, funciona actuando sobre el relé del conmutador. Lógico por otra parte ya que es éste el que permite que pase la corriente o se corte, y sobre el que actúa el interruptor cuando el TS2 convierte la señal de éste a un datagrama de encendido o apagado.

Como puede observarse en la figura 4.4, hasta la bombilla llega la fase de la corriente eléctrica. En una instalación normal ese cable llegaría hasta el interruptor y este se encargaría de abrir o cerrar el circuito con el neutro. En este sistema el que se encarga de abrir y cerrar este circuito es el relé, sobre el que puede actuar un interruptor normal a través del TS2 o un datagrama que nosotros mandemos hasta el NTA6F16H+COM.

4.1.2 Configuración de la instalación con software ETS3.

Una vez hecho el conexionado de los equipos, para que estos puedan funcionar de forma automática y pueda dotárseles de la automaticidad que los convierten en sistemas domóticos, han de ser configurados es decir, hay que establecer las relaciones entre ellos para que puedan enviarse datagramas ante los que actuar. Una vez configurados, los sensores podrán enviar mensajes EIB a los actuadores y estos ejecutaran las ordenes para las que han sido programados.

Para realizar la configuración de los dispositivos EIB, la EIBA ha creado un software, llamado ETS. Este software permite establecer relaciones entre los diferentes dispositivos, les asigna las direcciones para que se graben en su memoria y los configura en función de sus posibilidades.

Los fabricantes de dispositivos EIB, junto con los equipos, distribuyen unas bases de datos estandarizadas para el software ETS y del las que éste obtiene todas sus características y posibilidades de configuración. En estas bases de datos se incluyen: el nombre del fabricante, el tipo de BCU[2] que posee el equipo, los modos de configuración y en definitiva todos los parámetros del mismo.

El software ETS relaciona los elementos a través de las direcciones de grupo y permite, además, presentar un esquema de la instalación EIB a través de su interfaz gráfica.

Lo que se habrá de hacer es crear un proyecto ETS en el que se configurarán los dispositivos de los que consta la instalación según las posibilidades de los mismos. Estas posibilidades se obtienen a partir de las bases de datos suministradas por el fabricante. Una vez creado el proyecto, la configuración es pasada a los dispositivos a través del puerto serie y la interfaz RS-232 de la instalación.

El software ETS posee un módulo de chequeo de proyectos o de test. Éste, nos permite chequear si el proyecto creado es válido, tiene todas las fuentes de alimentación

2 El estándar define dos tipos de BCU. Es conveniente tener esto muy en cuenta porque, en función del tipo de BCU que posea el dispositivo, el protocolo que el estándar define para la conexión RS-232 es diferente.

necesarias y establecidas, y en general, si se cumplen las restricciones físicas que establece el estándar EIB. En esta etapa de diagnosis, es donde la dirección física cobra su importancia ya que es ésta la que se utiliza para la realización de los diferentes test.

Una vez realizada esta programación con el ETS tendremos una instalación de pruebas en la que los dispositivos tendrán direcciones físicas y de grupos. Los dos equipos estarán relacionados a través de las direcciones de grupo y, por tanto, el interruptor podrá encender y apagar la luz. Sin embargo, seguiremos sin poder hacer programaciones de encendido y apagado o crear escenarios típicos de una forma intuitiva y sencilla. Ésta es la principal funcionalidad que incorporará la plataforma software que se ha diseñado a lo largo del proyecto.

En el anexo2 se muestra como se ha de hacer la configuración del sistema de pruebas. Este constituye un ejemplo muy básico de instalación por lo que, para instalaciones de mayor emvergadura, convendría consultar referencia más detallada y que contemplen mayor numero de dispositivos y configuraciones.

4.2 EL SISTEMA LINCE.

El sistema LINCE se conecta a la instalación EIB a través de la interfaz serie RS-232. Se encargará de mandar a ésta, los comandos necesarios para que se realicen las actuaciones que el usuario desee. Además, este sistema se conectará a un PC a través de una interfaz de red por la que recibirá comandos mediante protocolo SSH. Estos comandos serán los que el usuario introduzca mediante el uso de la plataforma software.

El uso de esta configuración mediante una interfaz red y comandos SSH es debida a posibles ampliaciones futuras. Si se desea ampliar el sistema a través de Internet habría que crear una interfaz web que presente una estructura similar a la del software aquí desarrollado. Sin embargo, los comandos se seguirían transmitiendo mediante el protocolo SSH hasta el LINCE, por lo que la parte de control de la instalación no habría que modificarla y no habría que implementarle seguridad externa, algo muy importante cuando se habla de internet y aplicaciones web.

El sistema LINCE es un ordenador como otro cualquiera, pero con la salvedad de que funciona con un sistema operativo empotrado. Este sistema operativo es una distribución Debian la cual, al estar empotrada, se descarga al inicio sobre una memoria flash. A este Linux incorporaremos un programa que ejecutará comandos EIB y los transmitirá a través del puerto serie hacia la instalación. Según ésto, serán necesarios un compilador y un linkador así como un editor básico para poder hacer pequeñas modificaciones sobre este software. Aunque, esto último, no es excesivamente necesario ya que mediante comandos SSH podremos copiar los archivos ya compilados y/o crearlos en otra plataforma distinta. Evidentemente, el sistema, desde el punto de vista hardware, habrá de tener tarjetas de red y conexión RS-232.

4.2.1 El software de control de dispositivos EIB.

Este es el software encargado de enviar los comandos necesarios a los dispositivos de la instalación, a través de la interfaz serie. Se divide en dos partes fundamentales:

- Las librerías para el control de la interfaz RS-232.
- El programa de ejecución de comandos.

4.2.1.1 Las librerías para el control de la interfaz RS-232

Las librerías utilizadas para el control de la interfaz RS-232 están desarrolladas por el alemán Martin Koegler. Estas librerías no cumplían con lo que en un principio se deseaba pues, son un módulo de kernel para linux. De tal forma, que su funcionamiento consiste en desactivar el puerto serie normal del PC (Controlado a través del archivo /dev/ttyS0), insertando un módulo nuevo que permite introducir comandos EIB en dicho puerto (Cambia el fichero y el puerto serie pasará a ser controlado con el fichero /dev/eib). Estos comandos han de introducirse al nivel de enlace por lo que habrán de escribirse como dígitos hexadecimales en el archivo que controla el driver. Esto supuso un incremento considerable del trabajo y por ello tuvo que hacerse una reestructuración del proyecto, limitándose éste al control exclusivo de dispositivos EIB, dejando para una futura incorporación el poder aunar otros estándares con un mismo software.

Las librerías de Martin Koegler, como se ha dicho, cambiaban el módulo de control que por defecto se usa en el kernel de linux para el control del puerto serie. El nuevo módulo permite mandar a través de éste comandos EIB, siempre y cuando estos se introduzcan como comandos a nivel de enlace. Así, por ejemplo, el comando que habría que introducir para que pase a on el conmutador con dirección de grupo 0/0/1 es; \x11\x00\x00\x00\xe1\x00\01\x00\x81, mientras que el que lo hace conmutar a off es; \x11\x00\x00\x00\xe1\x00\01\x00\x80

A la hora de instalar estas librerías en nuestro equipo y poder utilizarlas los pasos a realizar son los siguientes:

Pasos para instalar las librerías.

1. Descargar el archivo eib-0.2.6.4.tar.gz de la pagina web http://www.auto.tuwien.ac.at/~mkoegler/index.php/eibdriver .
2. Descomprimir el documento bajado con la instrucción

   tar -xvzf eib-2.0.6.4.tar.gz

Esta es la última versión de las librerías, en el momento de la realización del proyecto. Sus características y ventajas se explican en la pag web. Además, en ella se pueden descargar otros programas de libre distribución para dispositivos EIB bajo licencia GPL.

3. Una vez que tenemos el archivo descomprimido se ejecuta el comando

   ```
   make
   ```

4. A continuación se ejecuta con permisos de superusuario el comando:

   ```
   setserial /dev/ttyS1 uart none
   ```

Este comando da de baja el modulo que controla el puerto serie, el cual tenía como archivo para el intercambio el /deb/ttyS1.

5. Mediante el comando siguiente se incorpora el módulo creado por las librerías para que sea el que controle el puerto serie.

   ```
   isnmod /home/usuario/eib-0.2.6.4/eib.ko
   ```

Con esto, el archivo para el control del puerto serie en el cual tendremos que escribir

los comandos será el /dev/eib0. Junto con este archivo, el módulo creará otros archivos que son /dev/eib1, /dev/eib2 y /dev/eib3 y que se utilizarían para el resto de puertos serie, en el caso de que haya mas de uno en nuestro PC.

6. Se puede comprobar que el módulo se instala correctamente visualizando los módulos instalados en el sistema. Para ello usaremos el comando:

```
lsmod
```

Con este comando aparecerán todos los módulos instalados en el kernel de Linux. Entre ellos ha de aparecer uno con el nombre eib que es el creado por las librerías.

Estos pasos hay que hacerlos con permisos de root. Para evitar tener al sistema durante un excesivo tiempo en este modo, debido a cuestiones de seguridad, se ha optado por ejecutar dichos comandos anteponiéndoles la palabra *sudo*. Este comando (*sudo*) permite ejecutar una instrucción como superusuario siempre y cuando en el archivo *sudoers*, que esta situado en */etc/*, se especifique que el usuario en concreto puede hacer uso de determinados comandos en modo root. Según esto, ha sido necesario modificar el archivo *sudoers* añadiéndole la siguiente linea:

```
#MANEJADORSERIE


usuario equipo=NOPASSWD: /bin/setserial, /sbin/insmod
```

Donde usuario es la cuenta y equipo es el nombre del PC.

Los pasos explicados han de realizarse cada vez que se arranque el sistema pues, por defecto, el sistema no utiliza como manejador del puerto serie el driver de las librerías. Para solucionar este problema lo que se ha hecho es crear un pequeño script de inicio con estos comandos escritos. A este script se le llama cada vez que se arranca el software de interfaz gráfica, que es el que manejará el usuario. De esta forma, nos aseguramos que el módulo que se utiliza es el correcto.

En realidad, en el archivo *makefile* que acompaña a las librerías viene la opción *make insert,* que realiza el comandos *insmod*. Por tanto*,* bastaría hacer este comando en lugar de los dos expuestos anteriormente (Pasos 4 y 5). Sin embargo, al ejecutar el comando *make insert* el sistema pide el password de usuario. Es debido a esto que es mejor ejecutar el script de inicio o en su defecto las dos instrucciones desde el otro PC al sistema LINCE. Así, la comunicación solo es necesario realizarla en un sentido y no en los dos, lo cual simplifica en gran medida el programa.

Conviene señalar, que antes de poder instalar el módulo y poder utilizarlo, hay que tener compilado un kernel propio. Por lo que es necesario tener las fuentes de un kernel y compilarlo para hacer una configuración particular, de otra forma, las librerías no pueden instalarse correctamente.

También, es conveniente señalar que el sistema linux sobre el que se han hecho las pruebas es una distribución Vidalinux. Vidalinux es una distribución Gentoo con la diferencia de que la instalación lleva el motor anaconda de RedHat. De esta forma, es posible que haya algunas pequeñas variaciones para otro tipo de distribuciones.

Comandos ante los que responden las librerías.

El funcionamiento después de instalar las librerías consiste en escribir tramas a nivel de enlace en el archivo /deb/eibX (X representa al archivo en función del puerto serie que escojamos, en nuestro caso será el 0, que se corresponde con el puerto COM 1 del PC). Estas tramas son enviadas por el driver al dispositivos interface EIB y este las mandará a la dirección de grupo indicada mediante el comando EIS correspondiente.

Los comandos han de introducirse como tramas del nivel de enlace y además, estas tramas deben ser entendibles por una BCU tipo 1. Por todo esto, es necesario estudiar el estándar KNX, que recormemos es totalmente válido para dispositivos EIB, para ver cual es el formato de trama y el tipo, en función de la labor a realizar.

El tipo de trama que se ha de utilizar para introducir los comandos es conocida como L_Data.rep. Es una primitiva de servicio para enviar datos y su formato es el que se detalla a continuación.

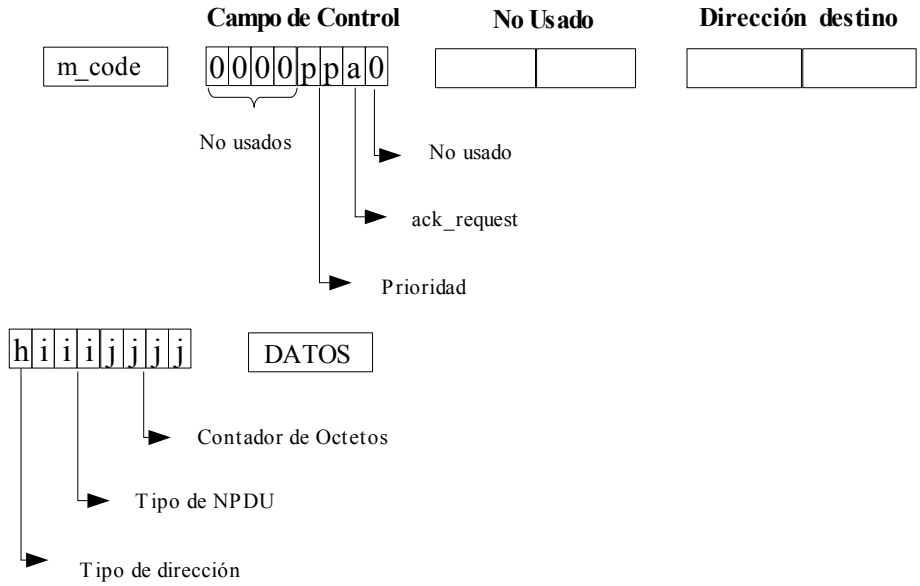

Figura 4.3: Formato de tramas EMI.

Los campos de esta trama significan:

m_code: Es un código que indica el tipo de trama. Para el caso de una L_Data.rep es 0x11.

Prioridad: Se utiliza para indicar la prioridad de la trama que se envía. Las distintas prioridades son las que se muestran en la siguiente tabla.

Bits de Prioridad	*Tipo de prioridad*	*Uso*
00	Sistema	Reservado para alta prioridad, configuración del sistema y gestión de procedimientos
01	Normal	Es la que se usa por defecto para tramas cortas.
10	Urgente	Reservado para tramas urgentes
11	Baja	Obligatorio para tramas largas, trafico de ráfaga...

Tabla 4.1: Tipos de prioridad.

ack_request: Se usa para indicar si se requiere asentimiento de respuesta. Hay dos posibles valores.

0 = No se requiere asentimiento explicito por parte de la capa superior, pero se puede mandar.

1 = No se requiere asentimiento de capa 2.

Dirección de destino: Indicara la dirección a la que se envía la trama.

Tipo de dirección: Se utilizará par indicar el tipo de dirección es a la que se manda la trama así, tendremos dos posibilidades.:

0 = Dirección individual o física.

1 = Dirección de grupo.

Contador de Octetos: Indica el número de octetos de datos que se mandan. 0 indicaría un octeto, 1 dos y así sucesivamente.

Así, por ejemplo, para el caso de la instrucción puesta anteriormente tendríamos que el valor de los campos es el siguiente.

m_code = 0x11.

Prioridad = 00.

ack_request = 0.

Dirección destino = 0/0/1.

Tipo de dirección = 1.

Contador de octetos = 1.

Este comando pasaba a on el conmutador con dirección de grupo 0/0/1. Para el resto de comandos lo que habrá que variar será el la dirección destino y el tipo de comando a enviar. Según esto el programa que escriba los comandos en el archivo /dev/eibX[3] ha de

3 A partir de ahora se tomará como archivo el /dev/eib0 que se corresponde con el puerto serie COM 1.

contemplar estos dos parámetros.

4.2.1.2 Programa de ejecución de comandos.

Este programa, haciendo uso del driver para linux explicado en el apartado anterior, escribe el comando que el usuario indica en el archivo antes mencionado para que pase a través de la interfaz serie a la instalación EIB.

Las librerías están escritas en lenguaje de programación C. Si además tenemos en cuenta que la mayoría de las distribuciones de Linux incorporan un compilador para este lenguaje de programación, tenemos el principal motivo por el cual se utiliza dicho lenguaje para esta parte del software. Por otra parte, la propia arquitectura del sistema hace que C sea una opción óptima ya que este programa únicamente se encarga de recibir un comando y una dirección de grupo, ofreciendo a cambio, la trama a nivel de enlace que hay que escribir en el archivo /dev/eib0. Visto desde otro punto, puesto que los requisitos son muy simples C, que es un lenguaje robusto y potente con el que ya se ha trabajado antes, constituye una muy buena opción para desarrollar esta parte.

En realidad no es un único programa sino que son varios programas pequeños. Así, el usuario introducirá el nombre de un programa o de otro en función de lo que quiera hacer. Por ejemplo, si el usuario simplemente quiere encender o apagar una luz, el programa a utilizar se llamará *onoff*, si lo que quiere es programar para encender a una determinada hora, llamará a un programa que se llamará *timer* y si lo que se desea es regular una luz, llamará a un programa *dimming*. Evidentemente, esto que se ha explicado para una luz es aplicable a cualquier dispositivo domótico, ya sea una luz, una persiana, el aire acondicionado o el sistema de riego del jardín.

El usuario lo que ha de saber es la acción a ejecutar además de la dirección de memoria a la que quiere enviar el comando. En definitiva, este programa es un programa en el que el usuario, por la línea de comandos, introducirá una acción a ejecutar y la dirección de grupo del dispositivo al que va dirigida la instrucción. También conviene señalar que, dependiendo del tipo de instrucción a ejecutar, habrá de introducir determinados parámetros como son: el tiempo de espera hasta encender o apagar la luz o el valor de regulación, si lo que se desea es regular un elemento físico.

El usuario de estos programas es el software que se ejecuta en el PC y que envía la línea de comandos mediante protocolo SSH al dispositivo LINCE. De esta manera, es el software de gestión quién tendrá que saber cual es el comando que habrá que mandar según lo que desee hacer el usuario final, el cual actúa sobre la interfaz gráfica.

Como ya se ha mencionado, este programa está formado por pequeños programas que ejecutan un comando cada uno. Se ha decidido hacer de esta forma, porque los comandos que se van a manejar no van a ser muchos, si acaso tres o cuatro. La parte que es común a todos ellos se codifica en un archivo aparte que se incluye en todos ellos. La que es diferente se programa en archivos diferentes. Así, se mantiene una estructura consecuente y solo supone saberse tres o cuatro comandos mientras que ahorra tener que testear número de parámetros en la linea de comandos y otras opciones que emborronarían el código y lo harían mucha más difícil de entender. Además, esta estrategia permite una ampliación con nuevos comandos muy sencilla y simple. Bastaría con tener documentado cuales son las funciones comunes creadas y crear un programa con las diferentes y que no

Este es el puerto con el que se han hecho las pruebas y con el que se ha funcionado. No obstante también se han hecho pruebas con otros puertos serie (COM2) y también han funcionado.

serán comunes.

Actualmente en esta primera versión del software, el numero de programas es dos. A continuación detallaremos cada uno de ellos, su funcionamiento y estructura.

Programa de encendido y apagado.

Los posibles comandos de ejecución de este programas son los siguientes:

```
onoff /0/0/1 on

onoff /0/0/1 off
```

donde:

onoff: Es el nombre del programa.

0/0/1: Es la dirección de grupo a la que se le envía el comando.

on/off: Es el comando a ejecutar, on para encender y off para apagar.

Este programa se encargará de conmutar a on o a off un relé que tenga esas dos posiciones. También, podrá utilizarse para reguladores que toleren comandos de encendido y apagado. Podrá utilizarse, por tanto, para encender y apagar toda clase de dispositivos.

En las pruebas realizadas se han encendido y apagado luces, pero por extensión puede encenderse y apagarse otra serie de elementos como son: persianas, aspersores o electrodomésticos. Estos últimos requieren de una interfaz adecuada.

Desde otro punto de vista, lo que hace el programa es recibir una dirección de memoria y un comando y como resultado escribe en el archivo /dev/eib0 una trama del nivel de enlace. Esta trama será entendida por el dispositivo interfaz RS-232 de la instalación EIB y transmitida al elemento con dirección de grupo indicada.

Como se ha explicado, las tramas entendibles por los equipos EIB son tramas del nivel de enlace por lo que, la dirección de grupo recibida como una cadena de caracteres, ha de traducirse a dos bytes en hexadecimal. En estos dos bytes el primer cero indica el grupo principal, el segundo cero el grupo medio, y el 1 el subgrupo (Mirar direccionamiento de tres niveles apartado 2.2.4.2).

Programa para encender/apagar en un determinado instante.

En este programa los posibles comandos de ejecución son los siguiente.

```
timer /0/0/1 10 s on

timer /0/0/1 10 s off

timer /0/0/1 10 m on

timer /0/0/1 10 m off

timer /0/0/1 10 h on

timer /0/0/1 10 h off
```

donde:

timer: Es el nombre del programa.

0/0/1: Es la dirección de grupo a la que enviar el comando.

10: Es el tiempo a esperar para ejecutar el comando.

s/m/h: Indicará si el tiempo se da en segundos, minutos u horas.

on/off: Indicará si lo que se desea es encender o apagar.

Este programa recibe como parámetros una dirección de grupo a la que enviar el comando, el comando a enviar y un tiempo de espera para que se envíe. El funcionamiento interno consiste en que una vez conocido el tiempo el programa se hecha a dormir y se queda dormido tanto tiempo como indique el parámetro introducido. Transcurrido este tiempo se envía un comando del mismo tipo que el del programa anterior.

Se contemplan diferentes posibilidades de tiempo (segundos, minutos y horas) por aumentar la versatilidad del programa, aunque, en la realidad, solo se utilice una o a lo sumo dos de estas opciones. En concreto, a la hora de programar los dispositivos el usuario de este programa (Plataforma software en el otro PC) solo utiliza la opción de los minutos.

Lo que se ha visto es el funcionamiento de estos dos programas. Desde el punto de vista estructural, estarán formados por cuatro archivos.

- **onoff.c**: Es el fichero con la función principal del comando para encender y apagar. En él se toma de la entrada los parámetros explicados antes y como resultado ofrece una trama que escribe en el archivo /dev/eib0.
- **timer.c**: Es el fichero con la función principal del comando para encender y apagar en un determinado tiempo. Toma de la entrada los parámetros explicados anteriormente. Como resultado ofrece una trama que escribe en el archivo /dev/eib0 después de haber pasado un tiempo determinado.
- **comun.c**: En este fichero se definen las funciones de uso común en ambos programas.
- **comun.h:** Este fichero incluye los prototipos de las funciones del anterior. Se incluye en los dos primeros y servirá para llevar un programación estructurada de esta parte del software.

4.2.2 Arquitectura de directorios del sistema LINCE.

A continuación, se muestra una figura del sistema de directorios para el LINCE. Se ha tratado de estructurar este sistema de tal forma que, futuras ampliaciones se puedan hacer de una manera simple.

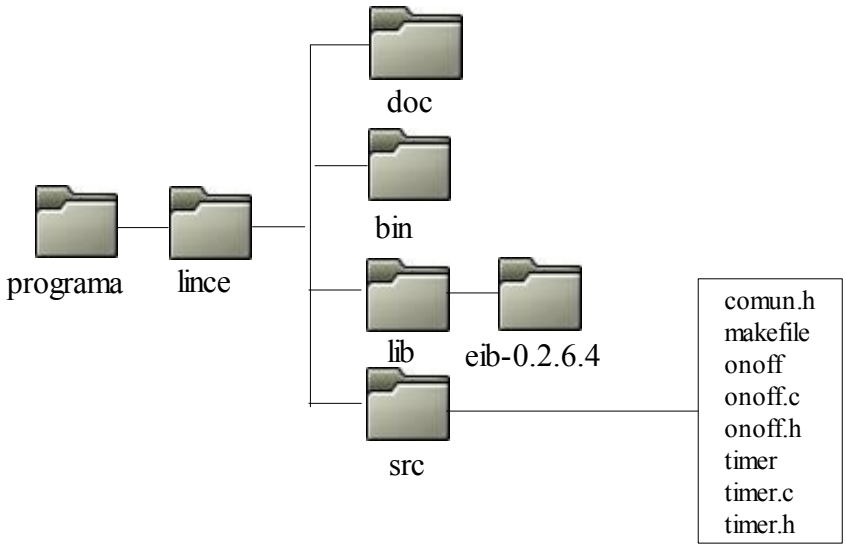

Figura 4.4: Sistema de directorio del equipo LINCE.

En esta estructura pueden distinguirse los siguientes directorios:

- **programa**:Es para estructurar las dos partes software (Interfaz gráfica y LINCE) de una forma común.
- **lince**: Representa a esta parte en concreto.
- **doc**: Contendrá la documentación concerniente a toda esta parte. En el se ubicarán tanto la parte correspondiente de esta memoria como otro tipo de documentación que se considerase oportuna.
- **bin**: En el se ubicarán los diferentes programas binarios pertenecientes a este software.
- **lib**: En el se incluyen las librerías necesarias para trabajar con esta parte. Por ello, se incluye dentro del mismo las librerías del driver para comunicación con el puerto serie.
- **src**: Se incluye dentro de este directorio el código fuente de esta parte del programa.

4.2.3 Resumen del Sistema LINCE.

El LINCE será el sistema que se conecte a la interfaz RS-232 de la instalación EIB. En el habrá instalado un sistema Debian empotrado, las librerías para comunicación con un dispositivo EIB con BCU tipo 1 y unos programas, a los que introduciéndoles una serie de parámetros por la línea de comandos, ofrecen como resultado la ejecución de una instrucción EIB.

Por otro lado, el LINCE tendrá como usuario de sus programas a otra plataforma software, la cual le hace las peticiones a través de una comunicación de red y mediante comandos de protocolo seguro SSH.

El software de actuación funciona con todo tipo de dispositivos EIB siempre y cuando la interfaz RS-232 de la instalación domótica tenga una BCU tipo 1. Además, su

estructura es sencilla y fácil de ampliar, así como también permite depurar errores de forma rápida y sin excesivas complicaciones.

Es importante recordar que el protocolo definido en el estándar para comunicación por el puerto serie recibe el nombre de EMI y no coincide con el definido para la comunicación entre dispositivos, el cual recibe el nombre de EIS.

Con este sistema se podrían encender y apagar toda clases de dispositivos dómoticos que fueran controlados por relés del tipo on/off. También será capaz de hacer que se enciendan y/o apaguen después de pasar un determinado tiempo. Sin embargo, la programación de los mismos se realizará a través de la interfaz gráfica instalada en el otro PC y mediante un proceso que se explicará en un apartado posterior.

El estándar EIB define una serie de adaptadores de electrodomésticos las cuales se conectan a aparatos como: lavadoras, microondas, lavavajillas, etc y permiten actuar sobre estos como si fueran dispositivos EIB. Según esto, este sistema podrá actuar también sobre este tipo de elementos.

En cuanto a la estructura de directorios de esta parte diremos que, se ha tratado que sea jerárquica para que futuras ampliaciones puedan llevarse a cabo de una forma lo más sencilla posible. Hay que señalar que esta parte del sistema, como la parte de la interfaz gráfica, se desean incorporar a un sistema de repositorios que permitan descargarla e instalarla de una forma sencilla y eficaz.

4.3 LA INTERFAZ GRÁFICA.

Esta es la segunda parte del software desarrollado en este proyecto. La interfaz gráfica irá instalada en un PC que se conectará al sistema LINCE a través de una conexión de red ethernet. En principio a través de cable. Se comunicará con el sistema LINCE mediante comandos SSH. Estos comandos son del tipo explicado anteriormente y nos permitirán actuar sobre el sistema LINCE o sobre la instalación, según sea el tipo de comando enviado.

Este software esta implementado en Java. Se ha elegido este lenguaje de programación por varios motivos. Los principales son los fundamentales de Java y la programación orientada a objetos.

- Herencia.
- Encapsulamiento.
- Polimorfismo

Por otro lado, tendremos otras propias de Java como son su portabilidad o su legibilidad. Será ésta, la portabilidad de Java, la verdadera razón por la que se ha escogido este lenguaje. En la actualidad hay muchos y muy diferentes tipos de equipos los cuales incorporan una maquina virtual de Java, la cual les permite ejecutar pequeñas aplicaciones para este tipo de lenguaje. De esta forma, podremos implementar o desarrollar implementaciones de nuestro software para este tipo de dispositivos. Teniendo así una gran versatilidad a la hora de ejecutar nuestra aplicación en sistemas móviles.

Por otro lado, la carga de este software no es excesiva, ni se prevé que lo sea tal y como se ha desarrollado el sistema, por lo que no se hace necesario el uso de otros

lenguajes orientados a objetos como son C++, más robustos en este sentido.

Otra razón es que, tal y como se ha pensado el sistema no tiene por qué haber una relación entre las dos plataformas software, ya que están perfectamente desacopladas a través de los comandos SSH.

Por último, y no por ello es una razón menos importante, es que se posee un conocimiento de este lenguaje pues ya se ha trabajado con el con anterioridad. Todas estas razones en conjunto hacen que Java sea un lenguaje muy apto para realizar la interfaz gráfica de nuestro sistema.

Debido a la amplitud del trabajo a realizar se ha utilizado para el desarrollo de la plataforma gráfica el entorno de desarrollo Neetbeans. Este tipo de software facilita enormemente la programación, depuración y prueba del sistema. Se ha decidido usar este y no otro entorno de desarrollo porque es un entorno optimizado para el lenguaje de programación Java, además de ser muy versátil y fácil de utilizar.

4.3.1 Características generales de la plataforma gráfica.

En muchas ocasiones, uno de los principales inconvenientes con el que se encuentran las tecnologías es las reticencias que presenta el usuario ante elementos que, mas que facilitarle la vida, se la complican. El desarrollo de esta plataforma ha tratado de hacerse de la manera mas simple, sencilla e intuitiva posible. De tal forma, que un usuario totalmente inexperto en sistemas domóticos pueda usarla sin necesitar adquirir grandes conocimientos. Le bastará con simples cliks de ratón o de dedo, si la pantalla es táctil, para poder ejecutar ordenes complejas.

Bajo este pensamiento, la plataforma software se ha desarrollado para permitir a cualquier usuario actuar sobre una instalación EIB de una forma sencilla e intuitiva. Le permitirá actuar con los dispositivos mediante el uso de iconos y sistemas gráficos simples y de una manera coherente. Permitiéndole realizar tareas que de otra manera serían muy complejas.

La plataforma permitirá, entre otras cosas, insertar nuevos elementos a la instalación desde el punto de vista lógico, teniendo en cuenta que, estos dispositivos físicamente han de estar instalados y configurados con anterioridad. Así, si se inserta un dispositivo de iluminación en la interfaz gráfica y se le asigna una dirección de grupo, este dispositivo debe existir físicamente y ha de estar previamente configurado con el software ETS.

Permitirá, además, interactuar con los dispositivos enviándoles comando ante los que estos respondan, realizar programaciones de los mismos en función de las necesidades de los usuarios, configurar escenarios típicos para crear ambientes y situaciones deseados y cargar instalaciones desde un archivo fuente.

4.3.2 Tecnología utilizada.

El lenguaje de programación utilizado para el desarrollo de la plataforma ha sido Java y principalmente sus clases Swing y Awt. Pero también, se han usado unas librerías llamadas jgraph, en su versión 5.9, que permiten dotar a los elementos gráficos de movilidad y versatilidad.

Estas jgraph, han sido desarrolladas por la empresa JGraph Ltd. Se pueden descargar

de la pagina web http://www.jgraph.com, de la cual también nos podremos descargar la documentación oportuna, así como ejemplos de su utilización.

Además de las jgraph, se han utilizado las librerías jsch en la versión 0.1.28 para las funciones de comunicación mediante comandos SSH.

Como se ha comentado anteriormente, para facilitar la programación y el desarrollo de la plataforma objetivo, se ha utilizado el entorno de desarrollo Netbeans. Este entorno esta especialmente diseñado para ser usado con el lenguaje Java. En la siguiente figura se muestra una captura con el entorno gráfico que presenta esta herramienta. Entre alguna de sus facilidades destacan: la importación automática de paquetes, el autocompletado de métodos y atributos, el autotabulado de las clases, etc. Todas estas herramientas facilitan la edición del programa. Sin embargo, la herramienta más potente que se ha utilizado es su depurador el cual ha facilitado enormemente la corrección de los errores y malfuncionamientos de la plataforma. Sin duda, Netbeans es una herramienta potente y optimizada para Java muy recomendable siempre y cuando se utilice este lenguaje de programación.

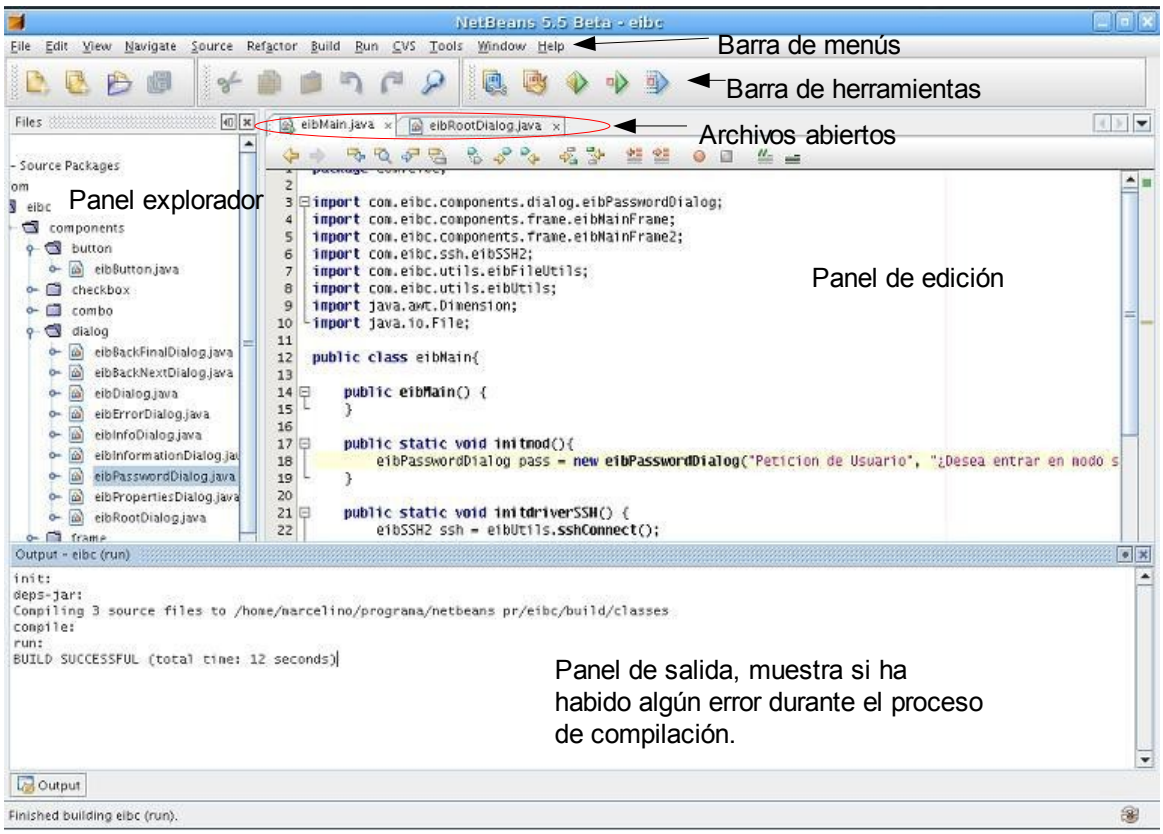

Figura 4.5: Captura del entorno Netbeans.

4.3.3 Arquitectura interna de la interfaz gráfica.

4.3.3.1 Diagramas UML.

En las siguientes figuras, se muestran diagramas de los paquetes de la interfaz

gráfica y las clases que contienen. Con ellos podremos hacernos una idea de la estructuración que se le ha dado a la plataforma gráfica. Se ha tratado que esta estructura sea lo mas jerarquizada posible ya que, de esta forma, posteriores ampliaciones serán mucho mas fáciles de acometer y desarrollar.

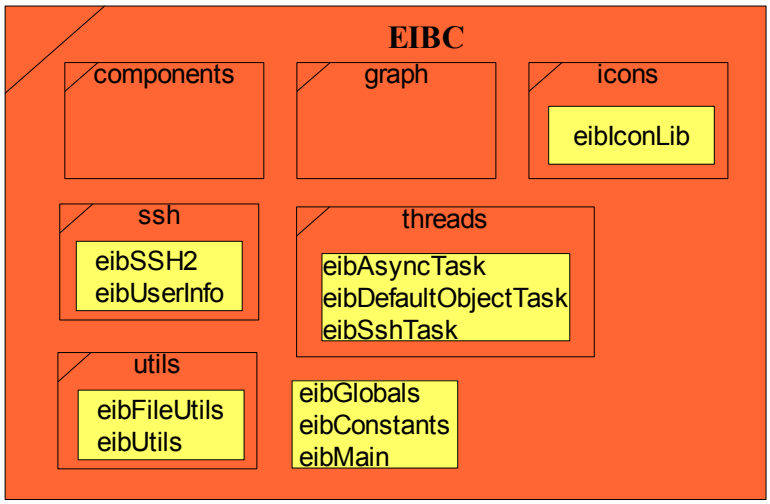

Figura 4.6: Estructura de paquetes de la interfaz gráfica.

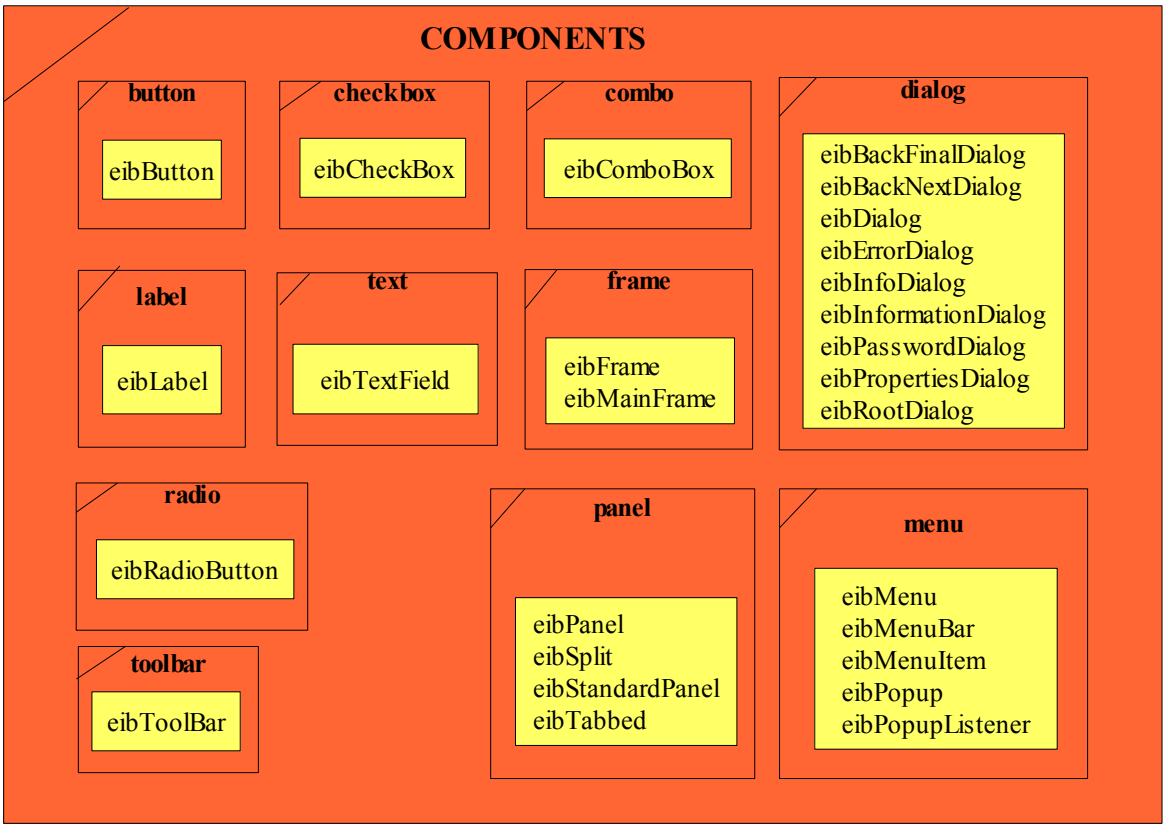

Figura 4.7: Estructura de paquetes del paquete components.

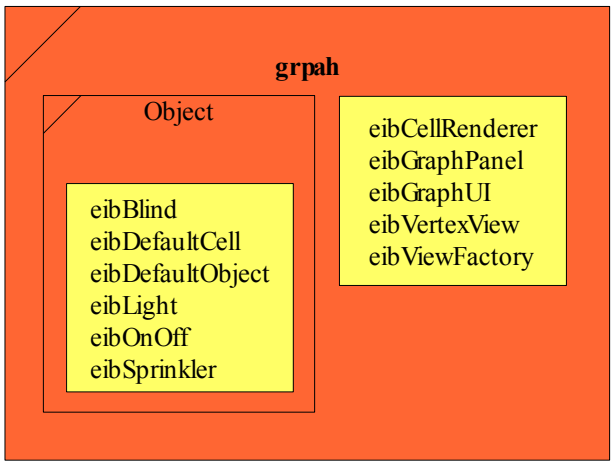

Figura 4.8: Estructura de paquetes del paquete graph.

A continuación se muestra un diagrama con todas las clases y las relaciones entre ellas. En el se puede apreciar como se ha partido de las clases Swing y Awt la cuales se han extendido para crear clases hijas. Esto permite hacer uso de una de las herramientas más potentes de la programación orientada a objetos (POO), la herencia. De esta forma, de la clase padre heredaremos todas sus propiedades y ventajas y, además, nos permitirá particularizar las clases hijas para el software desarrollado, sobrescribiendo métodos, incorporando algunos nuevos y creando nuevos elementos más versátiles en función de nuestras necesidades.

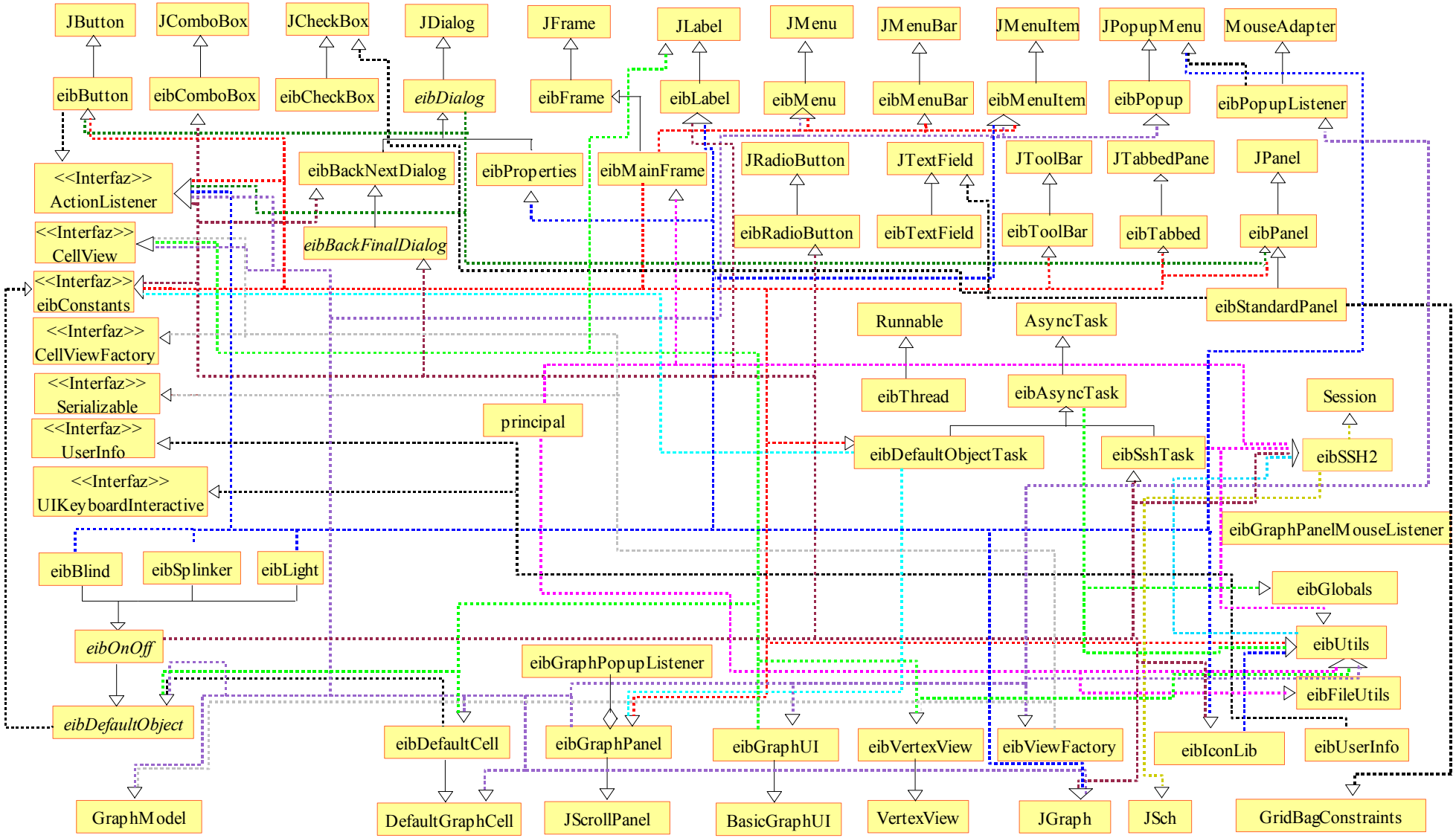

Figura 4.9: Diagrama de clases de la interfaz gráfica.

Como puede observarse, se ha decidido implementar la interfaz gráfica haciendo uso de las clases básicas de las librerías mencionadas, en lugar de utilizar el entorno de desarrollo para programar interfaces mediante el uso de botones predefinidos, barras predefinidas, etc. Esto ha sido así porque, el uso de esta herramienta introduce en el software mucho código inútil o código basura, que no solo carga en gran medida el sistema, sino que, además, lo hace mucho más difícil de depurar y manejar. Con esta forma de trabajar el código queda mucho más claro, y acompañado de la documentación necesaria, lo convierten en un sistema sencillo de depurar y ampliar.

4.3.3.2 Diagrama del sistema de archivos.

A continuación se muestran algunas figuras que aclaran la estructura de archivos seguido para la implementación de la interfaz gráfica. En la primera de ellas se muestra la estructura seguida hasta las clases definidas para la interfaz. La segunda muestra la estructura bajo la cual se han definido las clases.

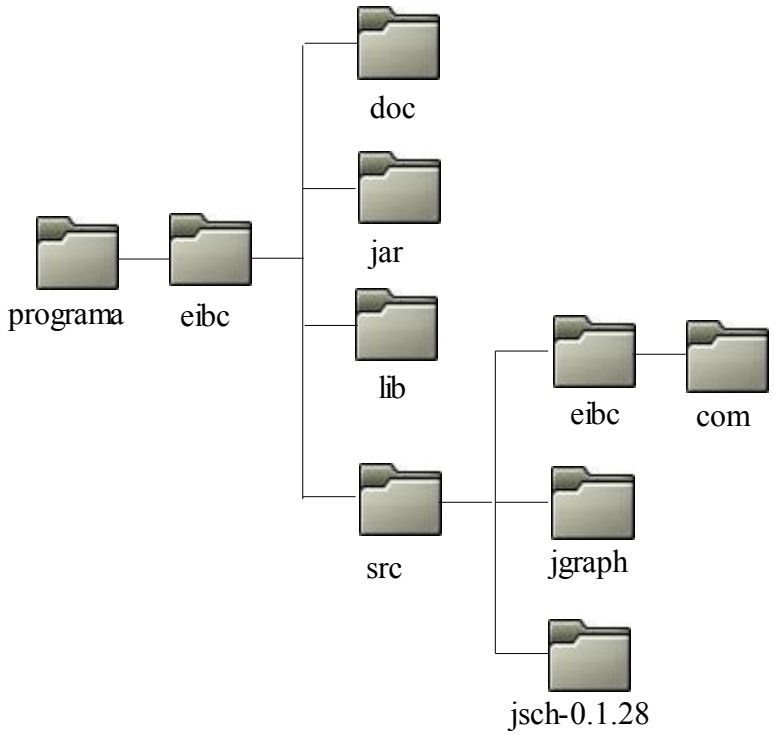

Figura 4.10: Sistema de directorios de la interfaz gráfica.

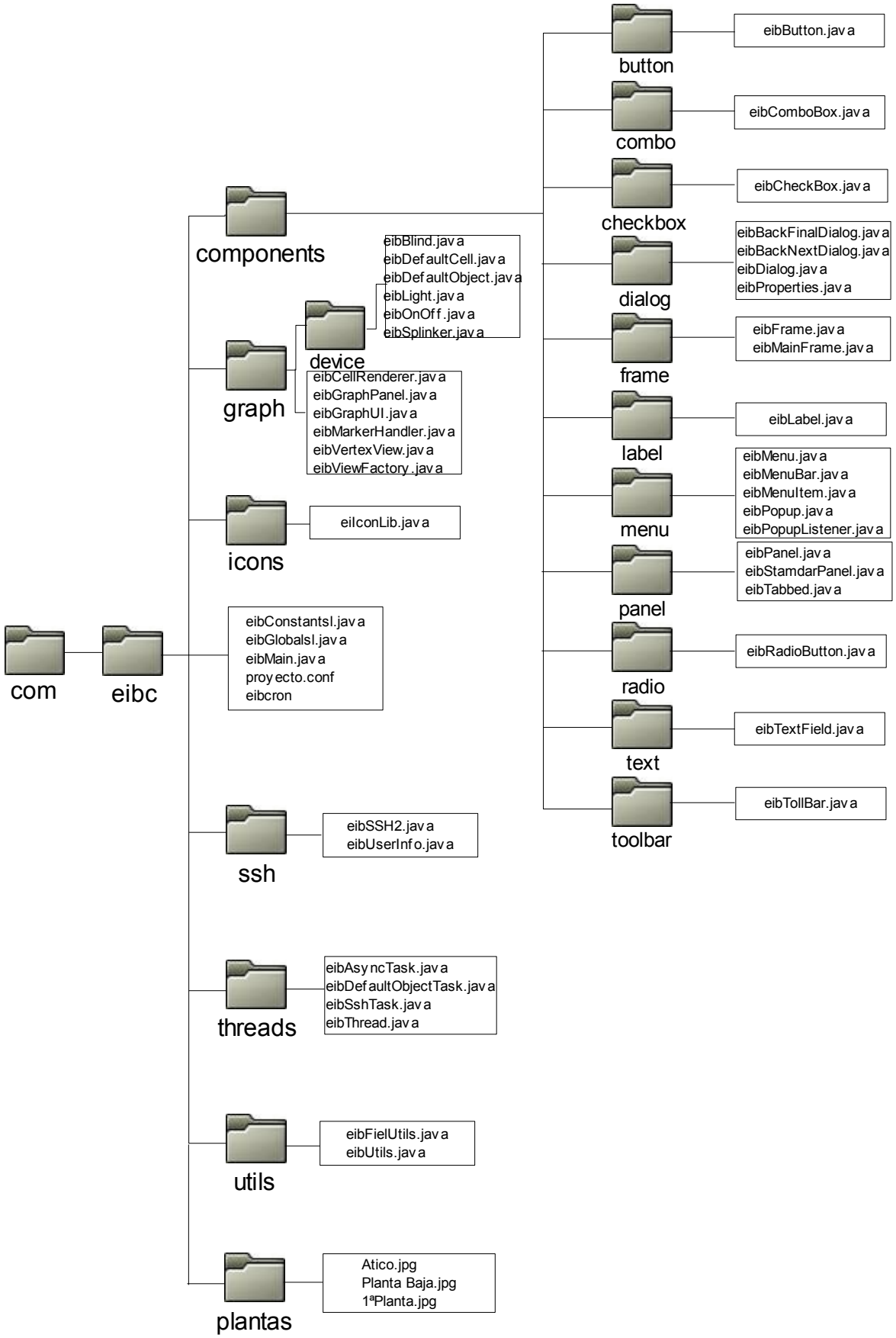

Figura 4.11: Estructura de directorio de las clases de la interfaz.

En la primera de las figuras podemos observar los siguientes directorios.

- **programa**: Este es el directorio con el que comienza la parte de la interfaz. Se crea para partir de un punto común respecto a la otra parte del software.
- **eibc**: Este directorio distingue a esta parte de la anterior.
- **doc**: Incluirá toda la documentación que se considere necesaria acerca de esta parte del software.
- **jar**: Incluye todos los archivos .jar necesarios para el correcto funcionamiento de la interfaz gráfica.
- **lib**: En este directorio se incluirán todas las librerías necesarias para el funcionamiento de esta parte del software.
- **src**: En él se incluirán los códigos fuentes de las diferentes partes de esta GUI. Por ello dentro de este directorio encontramos:
  - **eibc**: Incluye el sistema de directorios mostrado en la figura 4.11. En él se incluyen todas las clases programadas para el funcionamiento de la interfaz gráfica.
  - **jgraph**: En este directorio se incluyen las fuentes de las librerías jgraph.
  - **jsch-0.1.28**: Incluye todas las fuentes de las librerías jsch.

En la segunda figura, se muestra la estructura de directorios de la interfaz propiamente dicha. Se ha tratado que el sistema tenga una estructura jerárquica en función de las relaciones de dependencia de las diferentes clases. De la misma forma, se ha querido que este sistema de archivos sea lo más estructurado posible, para que a la hora de una futura ampliación ésta pueda realizarse de una forma mas cómoda.

De esta arquitectura, mostrada en las figuras 4.10 y 4.11, puede distinguirse la estructura de paquetes del sistema y con la que se completa la visión ofrecida por los diagramas del apartado anterior.

4.3.3.3 Análisis de clases.

En este apartado nos dedicaremos a explicar, una a una, las diferentes clases, sus atributos y métodos. Así como su funcionamiento y uso.

La clase eibButton.

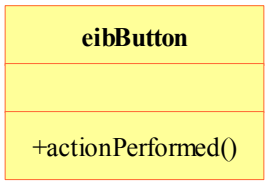

Figura 4.12: Clase eibButton.

Siguiendo la política mencionada de extender las clases dadas por los paquetes Java Swing y Awt, para poder dotar a estas de mayor funcionalidad y particularizarlas para

nuestro software si fuera preciso, ésta clase no es más que una extensión de la clase JButton del paquete javax.swing . No introduce ninguna particularidad, salvo la de extender la interfaz ActionListener, la cual permite introducir un evento cuando se pulsa el botón creado. Solamente define los constructores de la clase con llamadas a los constructores de la clase padre.

- *actionPerformed()*: Este método indicará la acción a ejecutar cuando se pulsa el botón. No se define ninguna acción en particular y tendrá que ser definido para cada uno de los botones creados, si así se estimase oportuno.

La clase eibComboBox.

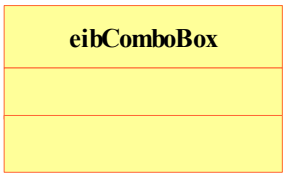

Figura 4.13: Clase eibComboBox.

Al igual que la anterior no es más que una extensión de la clase JComboBox definida por el paquete javax.swing. En este caso no se crea ningún método y tan solo se definen algunos constructores llamando al constructor de la superclase.

La clase eibCheckBox.

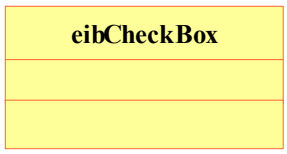

Figura 4.14: Clase eibCheckBox.

Esta clase se extiende de JCkeckBox definida en el paquete javax.swing y tan sólo define el constructor de la clase mediante la llamada al superconstructor.

La clase eibDialog.

Figura 4.15: Clase eibDialog.

Esta clase extiende de la clase JDialog definida en el paquete javax.swing. Es una clase abstracta que nos va a permitir crear cuadros de diálogos base para el software. De tal forma que, todos ellos sigan un formato común. Según esto, podremos crearlos de una sola vez definiendo tan solo los métodos abstractos para cada uno de ellos.

- *initComponents()*: Este método nos permite crear un cuadro de dialogo base en función de los atributos y cuyo valor se pasará en el constructor de la clase. Se llama una vez ejecutado el superconstructor de la clase padre.
- *getCenterPanel()*: Este método devuelve un elemento *eibStandardPanel*, que crea con los parámetros *title* y *bigComp*.
- *centerDialog()*: Centra el cuadro de dialogo para que este aparezca en el centro de la pantalla.
- *setVisible()*: Hace visible el cuadro de dialogo una vez que ha centrado.
- *getButtonNames()*: Este método obtiene el nombre de los botones que se quieren insertar en el cuadro de dialogo y que serán ofrecidos como parámetros mediante el parámetro *title*. Puesto que el nombre de los botones a insertar en el dialogo son particulares de cada caso, éste será un método abstracto a definir para cada caso.
- *buttonAction()*: Permite definir la acción a ejecutar por cada botón del cuadro de dialogo. Como el anterior, éste ha de ser un método abstracto.

La clase eibBackNextDialog.

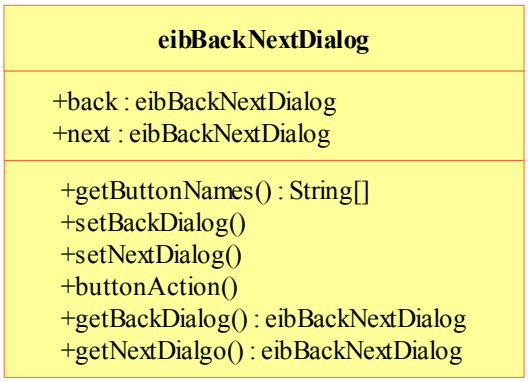

Figura 4.16: Clase eibBackNextDialog.

Esta clase se extiende de eibDialog. Define un caso particular de cuadro de dialogo del tipo que se muestra en la siguiente figura.

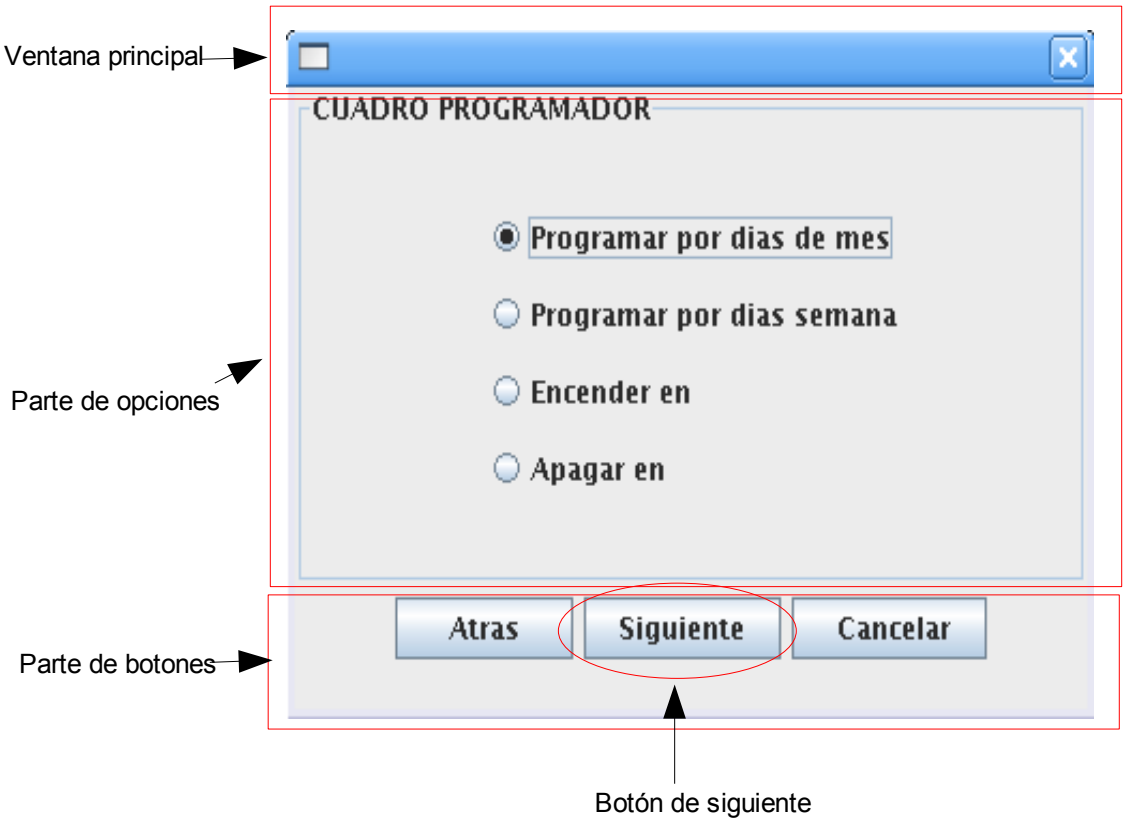

Figura 4.17: Cuadro de diálogo estándar.

En este tipo de cuadro de dialogo lo que permanece siempre igual es el panel que contiene los botones creados. Estos siempre recibirán los nombres mostrados en la figura. El resto del cuadro dependerá del caso en el que lo queramos utilizarlo.

- *getButtonNames()*: Define los nombres de los botones para este tipo de cuadro de dialogo, así como devuelve el nombre de los mismos. Este método necesita ser sobrescrito por ser abstracto en la superclase.
- *setBackDialog()*: Nos permite dictaminar el anterior cuadro de dialogo que se mostró y que podrá volverse a mostrar. Éste será del tipo *eibBackNextDialog*.
- *setNextDialog()*: Nos permite dictaminar el siguiente cuadro de dialogo a mostrar y que podrá ser del tipo *eibBackNextDialog* o *eibBackFinalDialog*.
- *buttonAction()*: En este método se indicarán los eventos a realizar al pulsar cada uno de los botones. Este método necesita ser sobrescrito por ser abstracto en la superclase.
- *getBackDialog()*: Este método devuelve el anterior cuadro de dialogo mostrado.
- *getNextDialog()*: Este método devuelve el siguiente cuadro de dialogo que se muestra.

La clase eibBackFinalDialog.

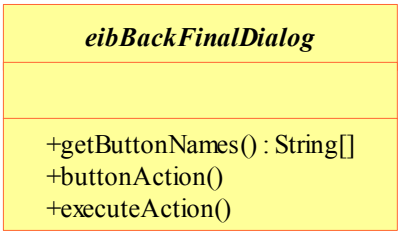

Figura 4.18: Clase eibBackFinalDialog.

Esta clase es una clase abstracta que hereda de la clase *eibBackNextDialog*. Este tipo de clase nos permitirá pintar un cuadro de dialogo como el que se muestra en la siguiente figura. Se divide en las mismas partes que el anterior, pero con la salvedad de que en lugar de tener un botón para avanzar hasta otro cuadro tiene uno que finaliza.

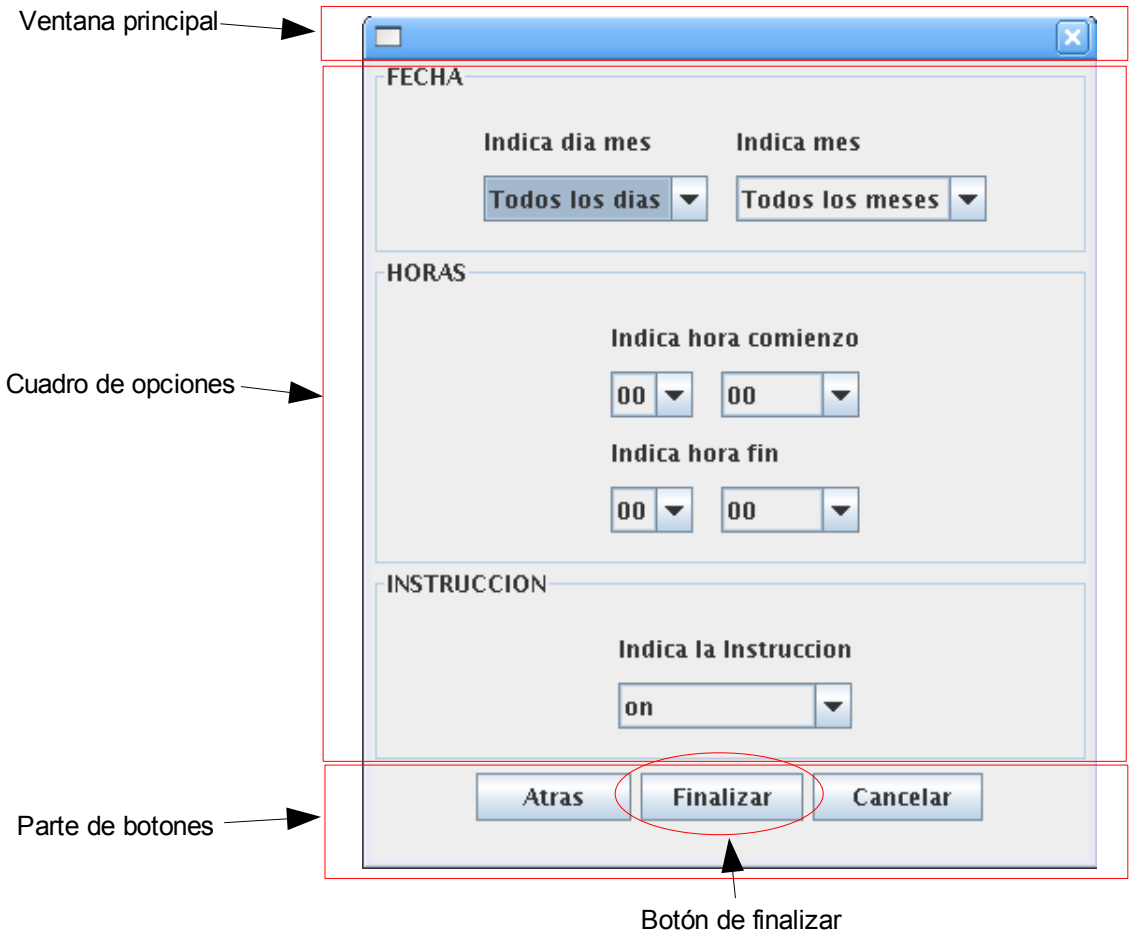

Figura 4.19: Cuadro del tipo eibBackFinalDialog.

En este tipo de cuadro la principal característica es que los botones cambian y, por tanto, la acciones a ejecutar. Los métodos que define son:

- *getButtonNames()*: Sobrescribe el método de la superclase que se encarga de definir los nombres de los botones para este tipo de cuadro de dialogo. Devuelve el nombre de los mismos.
- *buttonAction()*: Sobrescribe el método de la superclase para dictaminar cuales han de ser las nuevas funciones a ejecutar. Llamará a la función *executeAction()* para dictaminar cuales serán las diferencias con el método de la superclase.
- *executeAction()*: Este es un método abstracto el cual ha de ser implementado para este tipo de cuadros de dialogo. Permitirá ejecutar las acciones para los botones que son diferentes a los de la clase *eibBackNextDialog*.

La clase eibPropertiesDialog.

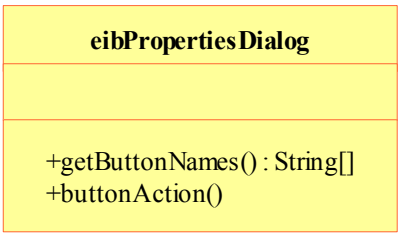

Figura 4.20: Clase eibPropertiesDialog.

Esta clase nos permitirá crear un cuadro de dialogo del tipo que se muestra en la siguiente figura. Tienen la misma estructura que los anteriores. Sin embargo, no posee una parte de botones para avanzar hacia adelante o hacia atrás.

Figura 4.21: Cuadro del tipo eibProperties.

Esta clase hereda de la clase *eibDialog*, por lo que ha de sobrescribir los métodos abstractos de esta. La principal diferencia con los cuadros de diálogos anteriores, es que los de este tipo no poseen botones, tal y como se muestra en la figura anterior. En este caso los métodos se caracterizan por no devolver ningún nombre para los botones (*getButtonNames()*) y no realizar ninguna acción para los mismos (*buttonAction()*).

La clase eibInformationDialog.

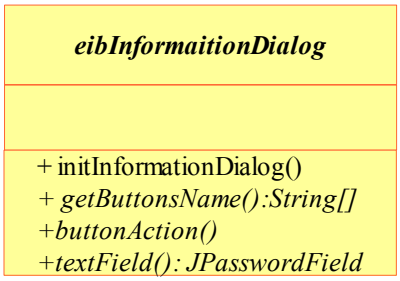

Figura 4.22: Clase eibInformationDialog.

Esta clase es una clase abstracta cuya finalidad es la de ser la base sobre la que se construyan los diferentes cuadros de mensajes de la interfaz. Nos servirá para detallar mensajes de error, información, etc. Esta clase hereda de la clase *Jdialog* que se encuentra dentro del paquete javax.swing. Consta de los siguiente métodos:

- *initInformationDialog()*: Se encarga de inicializar el cuadro de dialogo que se ha de mostrar.
- *getButtonsName()*: Cada uno de los cuadros tendrá una serie de botones, cuyos nombres se han de definir en este método. Estos nombres serán devueltos como parámetro. Este será un método abstracto
- *buttonAction()*: Define la acción asociada a cada botón del cuadro. Al igual que el anterior será un método abstracto que tendrá que ser definido para cada cuadro en cuestión.
- *textField()*: Dependiendo del tipo de cuadro, este método podrá pedir un parámetro como un campo de texto. Es el caso del cuadro de petición de password. Como los anteriores es un método abstracto.

La clase eibErrorDialog

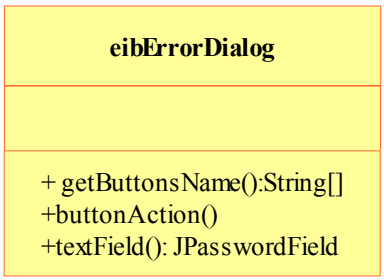

Figura 4.23: Clase eibErrorDialog.

Esta clase se encarga de mostrar un cuadro de error en el que se indica el tipo de error que se ha cometido. Además, si el error producido se considera crítico, entonces, se cerrará el programa y se saldrá de éste. Los métodos que define son los métodos abstractos que ha de implementar por heredar de la clase *eibInformationDialog*. La funcionalidad de

estos métodos es la descrita para la clase padre. En este caso el método *textField()* no es necesario y por tanto no hará nada.

La clase eibInfoDialog

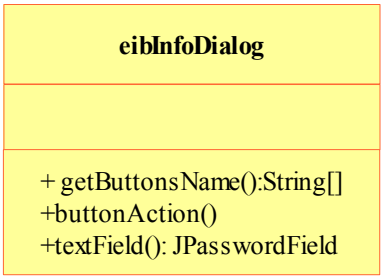

Figura 4.24: Clase eibInfoDialog.

Esta clase extiende de la clase abstracta *eibInformationDialog* por lo que ha de sobrescribir los métodos que esta última define. Sólamente define los métodos abstractos de la superclase, cuya funcionalidad ya se ha explicado en ésta. Se encarga de mostrar mensajes de información que no son de error y que sirven para decir al usuario que es lo que esta sucediendo. Así podrá indicar si le falta algún dato que incorporar al sistema, la dirección introducida no es correcta, etc.

La clase eibPasswordDialog.

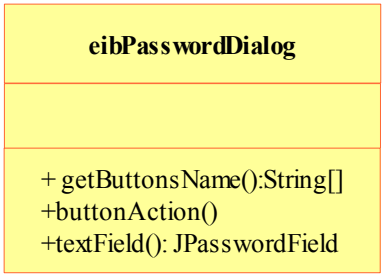

Figura 4.25: Clase eibPasswordDialog.

Esta clase se utiliza para mostrar el cuadro de dialogo que ofrece la posibilidad de cambiar de modo. Hereda de la clase *eibInformationDialog* y, por tanto, ha de sobrescribir los métodos abstractos que define esta última. Con la funcionalidades que para ellos se han definido.

La clase eibRootDialog.

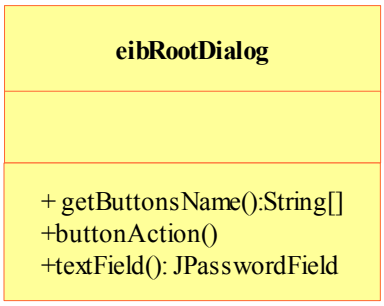

Figura 4.26: Clase eibRootDialog.

Este cuadro pide la contraseña de root, verifica si esta es correcta y en caso de ser así cambia al modo administrador. Esta clase hereda de la clase *eibInformationDialog* por lo que ha de sobrescribir los métodos abstractos de ésta. La diferencia entre este cuadro y los anteriores es que en este caso el método *textField()*, devuelve un elemento *JPasswordField* que no es null.

La clase eibFrame.

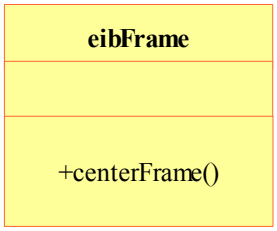

Figura 4.27: Clase eibFrame.

Esta clase extiende de la clase JFrame que se localiza en el paquete javax.swing y hereda todas las propiedades de ésta. Define el método *centerFrame()* y su creación responde a la política de aprovechar la herramienta de la herencia ofrecida por la programación orientada a objetos.

- *centerFrame()*: Permite centrar el objeto *eibFrame* que se crea.

La clase eibMainFrame.

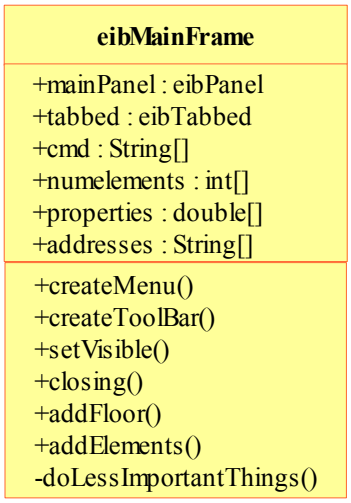

Figura 4.28: Clase eibMainFrame.

Esta clase hereda de la clase *eibFrame*. Será la clase que permita crear los Frame sobre los que se insertarán los elementos jgraph, los cuales representarán los dispositivos EIB. Los métodos que define son los siguientes.

- *createMenu()*: Crea los menús característicos de las interfaces gráficas como son los menús *Archivo* o *Herramientas*. Mirar figura 5.5 para ver dichos menús.
- *createToolBar()*: Crea una barra de herramientas como las que usualmente aparecen en las interfaces gráficas. Mediante ella podremos insertar todo tipo de elementos que representarán dispositivos EIB. Mirar figura 5.5 para ver este tipo de barra de herramientas.
- *setVisible()*: Permite hacer visible el Frame.
- *closing()*: Este método permitirá salir del programa de forma estructurada.
- *addFloor()*: Este método permitirá añadir diferentes Frames. Se utilizará cuando carguemos la configuración desde un archivo. Cada Frame representará a una planta del edificio, las cuales han de aparecer en dicho archivo de configuración.
- *addElements()*: Con este método cargaremos, en cada una de las plantas representadas por cada Frame, los elementos que en esta estuvieran y que harían referencia a los diferentes dispositivos EIB localizados en la instalación real. El numero de elementos y tipo, así como otras propiedades, aparecerían en el archivo de configuración del que habría que leer.
- *doLessImportantThings()*: Este método se encarga de dictaminar que es lo que hay que hacer, en función de si se cierra o se abre un Frame. También se encargará de realizar otras acciones comunes menos importantes.

La clase eibLabel.

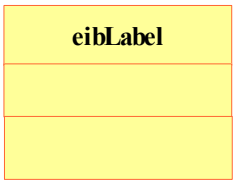

Figura 4.29: Clase eibLabel.

Esta clase extiende de la clase *JLabel* del paquete javax.swig. En este caso no se crea ningún método y tan solo define el constructor llamando al constructor de la superclase.

La clase eibMenu.

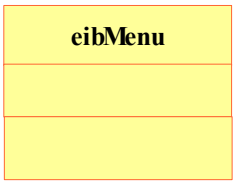

Figura 4.30: Clase eibMenu.

Esta clase extiende de *JMenu*, una clase perteneciente al paquete javax.swing. Permite crear menús y, como en el caso anterior, *eibMenu,* tan sólo define el constructor llamando al constructor de la superclase.

La clase eibMenuBar.

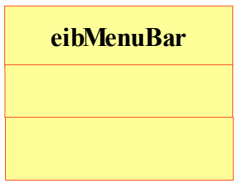

Figura 4.31: Clase eibMenuBar.

La clase *eibMenuBar* extiende de *JMenuBar*, una clase perteneciente al paquete javax.swing. Permite crear barras de menús y tan sólo define el constructor llamando al constructor de la superclase.

La clase eibMenuItem.

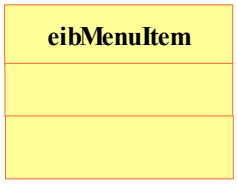

Figura 4.32: Clase eibMenuItem.

La clase *eibMenuItem* extiende de *JMenuItem*, la cual se encuentra dentro del paquete javax.swing. Al igual que las anteriores solo define su constructor, el cual se construye llamando al de la superclase, mediante el comando *super*.

La clase eibPopup.

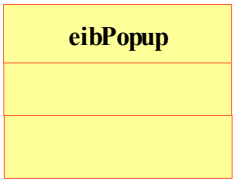

Figura 4.33: Clase eibPopup.

La clase *eibPopup* extiende de *JPopupMenu*, la cual está dentro del paquete javax.swing. Actualmente solamente define el constructor de esta clase llamando al de la superclase.

La clase eibPopupListener.

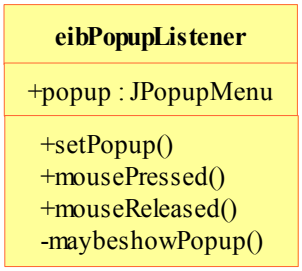

Figura 4.34: Clase eibPopupListener.

Esta clase se define para dictaminar los eventos que se producen cuando se pulsa el ratón sobre un punto de la interfaz. Esta clase extiende de la clase *MouseAdapter* y va a sobrescribir algunos métodos de ésta. Parte de un elemento *JPopupMenu* y los métodos que define son los siguientes.

- *setupPopup()*: Permite elegir un popup(menú desplegable) sobre el que realizar los eventos.
- *mousePressed()*: Este método se invoca cuando se pulsa un botón del ratón. Cuando se produce un evento de este tipo se llama al método *maybeshowPopup()*.
- *mouseReleased()*: Este método se invoca cuando se libera un botón del ratón y, como el anterior método, lo que hace es llamar al método *maybeshowPopup()*.
- *maybeshowPopup()*: Este método lo que hace es mostrar un menú desplegable en el lugar en el que se pica con el ratón.

La clase eibPanel.

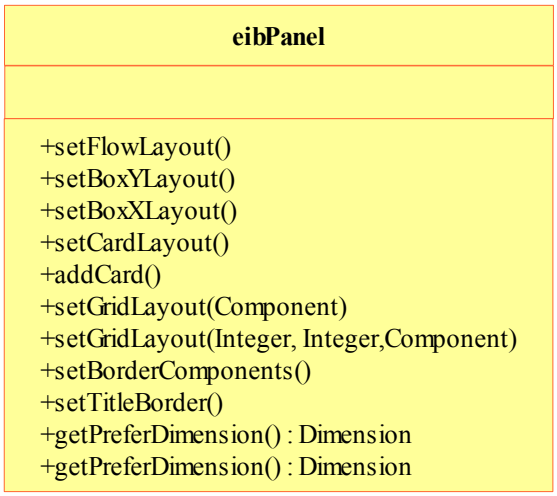

Figura 4.35: Clase eibPanel.

Esta clase extiende de la clase *JPanel* del paquete javax.swing. Además de heredar todas las características de *JPanel* define los siguientes métodos:

- *setFlowLayout()*: Este método permite elegir como layout del panel el tipo FlowLayout.
- *setBoxYLayout()*: Con este método se indicará como layout del panel el tipo BoxLayout con orientación el eje Y.
- setBoxXLayout(): Con este método se indicará como layout del panel el tipo BoxLayout con orientación el eje X.
- *setCardLayout()*: Este método indica que el layout del panel será del tipo CardLayout.
- *setGridLayout()*: Indica que el layout del panel será del tipo GridLayout. En esta primera forma del método recibe como parámetro una matriz de elementos *Component*. El método crea un layout a partir de las dimensiones del elemento *Component [][]* .
- *setGridLayout()*: En esta segunda forma, el método realiza lo mismo que la anterior. Sin embargo, ahora recibe como parámetros dos elementos del tipo *Integer* y un elemento del tipo *Component [][]*.
- *setBorderComponents()*: Este método se utilizará para indicar que el layout del panel será del tipo BorderLayout, y colocará los elementos introducidos como parámetros en el lugar oportuno.
- *setTitleBorder()*: Permite introducir un titulo en cada uno de los cuadros en los que se divide el panel.
- *getPreferDimension()*: Este método devuelve las dimensiones óptimas que serán las mínimas del panel actual. En este primer caso el método no recibe ningún parámetro.
- *getPreferDimension()*: Este método devuelve las dimensiones óptimas del

elemento que se le introduce como parámetro, el cual es del tipo *JPanel*.

La clase eibStarndardPanel.

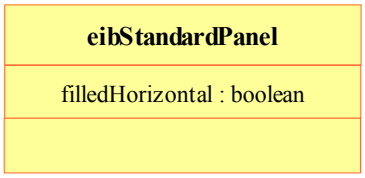

Figura 4.36: Clase eibStandardPanel.

Esta clase extiende de la clase *eibPanel* y lo que nos va a crear es un panel que denominaremos estándar. Esta clase solamente define el constructor de la misma en el que recibe como parámetros un elemento del tipo *String[]* y otro del tipo *Component[][][]*. Estos objetos son los que se utilizan en la clase *eibDialog* para construir los cuadros de dialogo (referencia a los métodos *getCenterPanel()* e *initComponents()* de la clase *eibDialog*). Esta clase realiza los pasos necesarios para crear un panel, que incluido en el cuadro de dialogo, separa a este en diferentes partes. Cada una de estas partes con un encabezado que se indica con *setTitleBorder()* y con una serie de objetos. Estos objetos podrán ser de cualquier tipo que herede de *Component.*

La clase eibTabbed.

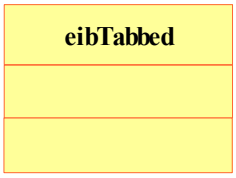

Figura 4.37: Clase eibTabbed.

La clase *eibTabbed* extiende de *JTabbedPane*, la cual se encuentra dentro del paquete javax.swing. Esta clase solamente define su constructor llamando al de la superclase mediante el comando *super.*

La clase eibRadioButton.

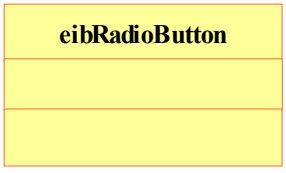

Figura 4.38: Clase eibRadioButton.

La clase *eibRadioButton* no es más que una extensión de *JRadioButton* definida por el paquete javax.swing. En este caso no se crea ningún método y tan sólo se define el constructor llamando al de la superclase.

La clase eibTextField.

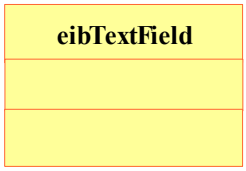

Figura 4.39: Clase eibTextField.

Ésta es una extensión de la clase *JTextField* definida por el paquete javax.swing. En este caso, al igual que en algunos de los anteriores, no se crea ningún método y tan sólo se definen algunos constructores llamando al constructor de la superclase.

La clase eibToolBar.

Figura 4.40: Clase eibToolBar.

La clase *eibToolBar* hereda de la clase *JToolBar*, que se encuentra en el paquete javax.swing. Como en otras ocasiones, para esta clase, solamente se han definido algunos constructores, mediante la llamada a los constructores de la clase padre.

La clase eibDefaultCell.

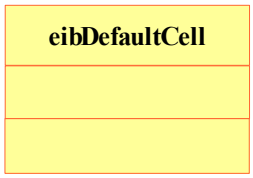

Figura 4.41: Clase eibDefaultCell.

Esta clase hereda de la clase *DefaultGraphCell* que se encuentra dentro del paquete org.jgraph.graph. Nos permitirá crear los objetos graph que se utilizarán en la interfaz gráfica para representar los dispositivos EIB. Estos objetos se insertarán en una posición determinada que por defecto será el centro de la pantalla. De la misma forma que hasta ahora esta clase extiende de una clase general, facilitándose de esta forma la particularización para nuestro software. En ella tan sólo se define el constructor llamando al de la superclase e incorporándole la capacidad para insertarse por defecto en un determinado lugar de la pantalla.

La clase eibDefaultObject.

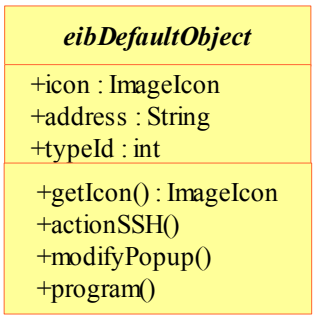

Figura 4.42: Clase eibDefaultObject.

Esta clase es la que constituirá la base para implementar gráficamente los dispositivos físicos. De ella se derivan los elementos y se particularizarán para los distintos casos. Por esto tiene una serie de atributos y definirá unos métodos abstractos que variarán según hagan referencia a un tipo de elemento u otro. Esta clase, además, implementa la interfaz *eibConstants,* que nos permitirá discernir entre los diferentes tipos de dispositivos. Los métodos que define esta clase son:

- *getIcon()*: Devuelve un elemento *ImageIcon* con el icono que representa al objeto.
- *actionSSH()*: Este método se encarga de ejecutar el comando SSH que se introduce como parámetro. Éste es un método abstracto que ha de ser sobrescrito por las clases hijas que hereden de *eibDefaultObject*.
- *modifyPopup()*: Este método nos va a permitir modificar el menú desplegable, de tal forma que, cuando picamos sobre un dispositivo. se añadan o quiten ciertos elementos de dicho menú. Así, particularizaremos menús para los dispositivos. También éste es un método abstracto que se ha de sobrescribir.
- *program()*: Este es un método abstracto que se ha de sobrescribir. Se encargará de configurar los dispositivos para que se programen y ejecuten comandos cuando los usuarios así lo requieran.

La clase eibOnOff.

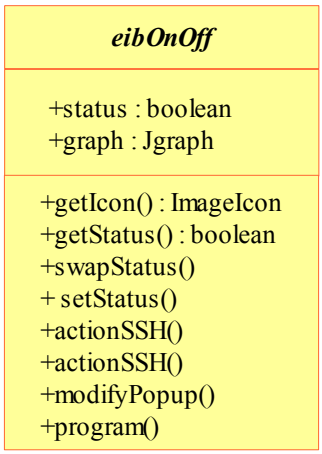

Figura 4.43: Clase eibOnOff.

Esta clase hereda de *eibDefaultObject.* Es una clase abstracta por lo que no ha de definir todos los métodos de la clase padre. Define algunos métodos característicos de los dispositivos que funcionan en la forma On/Off. Estos, si recordamos, eran el tipo de dispositivos que teníamos físicamente en la instalación de pruebas. Actualmente esta clase de elementos gráficos es el único tipo de elementos del que se dispone. Sin embargo, pueden existir elementos que además puedan regularse, para los que habría que crear una clase ya que funcionarían de forma diferente. Por otra parte, esta clase deja particularidades para que se definan los dispositivos concretos.

- *getIcon()*: Éste es un método abstracto que devolverá la imagen que representa un elemento controlado por un dispositivo EIB.
- *getStatus()*: Devuelve un valor booleano con el estado en el que se encuentra el elemento. Así, si este éste está on se devolverá true y false si esta a off.
- *swapStatus()*: Método que conmuta el estado del elemento, es decir, si este esta a on cambia el valor de la variable s*tatus* a off.
- *setStatus()*: Permite poner un valor en la variable *status*.
- *actionSSH()*: En esta primera forma, este método recibe como parámetros una cadena de caracteres en la que se indica el tipo de acción a ejecutar y se encargará de ejecutar la acción dentro de dos posibilidades, conmutarlo en el momento o programarlo.
- *actionSSH()*: En la segunda forma del método, se reciben como parámetro dos cadenas de caracteres una que indicando el comando y otra el tiempo. En este caso, este método se encargará de programar el dispositivo para que actúe transcurrido un tiempo predefinido por el usuario.
- *modifyPopup()*: Éste es un método abstracto, que ha de definir cada elemento concreto.
- *program()*: Este método permitirá programar los dispositivos que tengan la forma de funcionamiento On/Off.

La clases eibBlind, eibLight, eibSplinker.

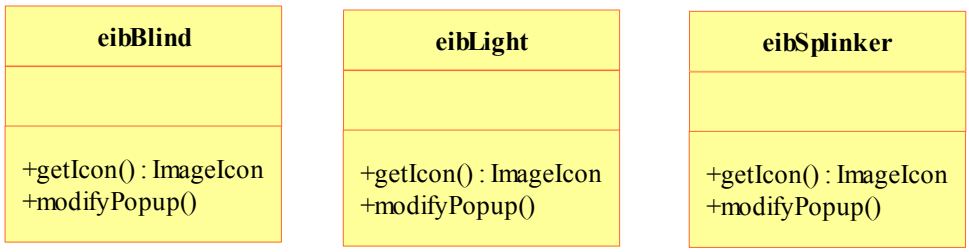

Figura 4.44: Clases eibBlind, eibLight, eibSplinker.

Estas clases son las que van a representar a los elementos comunes existentes en el hogar y que están relacionados con dispositivos EIB. Define dos métodos que ya se han explicado:

- *getIcon()*: Devuelve el icono que representa al elemento del hogar (una bombilla, una persiana,...).
- *modifyPopup()*: Para modificar el desplegable que aparece al picar sobre uno de estos elementos.

La clase eibGraphPanel

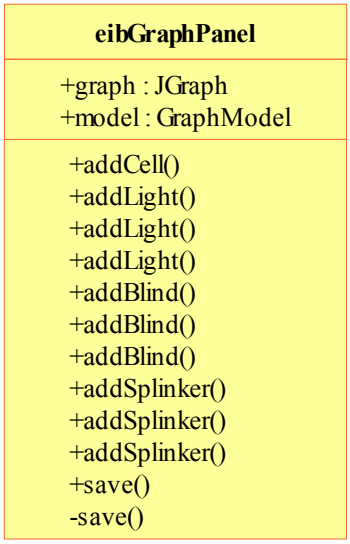

Figura 4.45: Clase eibGraphPanel.

Esta clase hereda de la clase *JScrollPanel* que se encuentra dentro del paquete javax.swing. Define un panel del tipo mencionado y sobre el que añade un elemento del tipo graph. Constituirá cada uno de los paneles sobre los que se añadirán los elementos relacionados con los dispositivos EIB. Como se muestra en la representación UML requiere el graph y un modelo. Los métodos que define esta clase son:

- *addCell()*: Añade una celda al panel del tipo que se indica como parámetro y que será *eibDefaultObjec.* Además, se insertará en una posición que también se pasa como parámetro.
- *addLight()*: Este método inserta una luz para que pueda ser controlada por el usuario final mediante el control del dispositivo EIB al que se conecta. En este primer caso, recibe como parámetro una dirección de grupo que identificará al aparato. La luz se insertará en el panel sobre el que hayamos picado y en el centro del mismo. No obstante, gracias a las propiedades de los elementos graph podremos mover el elemento por todo el panel.
- *addLight()*: En esta segunda forma además de pasar como parámetro la dirección de grupo del elemento físico, se pasará la posición en la que se desea insertar la luz.
- *addLight()*: En esta última forma además de la posición y la dirección de memoria, se pasa como parámetro el elemento graph en el que se desea insertar.
- *save()*: Permite guardar una configuración de un panel grahp, es decir, guarda en un fichero los dispositivos con sus propiedades (direcciones de grupo, posición, tipo de elemento, etc). En este primera forma recibe como parámetros el fichero y el nombre del panel graph en el que se encuentra.. Esta primera forma lo que hace es llamar a la segunda.
- *save()*: Esta segunda forma es la que escribe en el fichero. La anterior se encargaba de identificar el tipo de elemento graph y llamar a esta segunda forma. Las dos en conjunto realizan el proceso de guardar la configuración de una instalación particular.

Los métodos *addBlind()* y *addSplinker()* en sus otras tres formas hacen exactamente lo mismo que hacen los métodos *addLight, c*on la salvedad de que, lo que insertan son persianas, que se identificarán con dispositivos de subida y bajada de persiana (Estos dispostivos permiten regular, al igual que las luces, por lo que en realidad habrían de heredar de una futura clase llamada *eibDimming* y no de *eibOnOff*) y aspersores, que se identificarán con los dispositivos EIB destinados al control del riego (Estos dispositivos son los únicos, de los que hemos visto hasta ahora, que puramente pertenecen al grupo de elementos que funcionan con un sistema On/Off).

La clase eibGraphPopupListener.

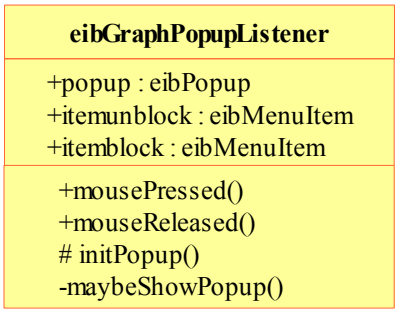

Figura 4.46: Clase eibGraphPopupListener.

Esta clase hereda de la clase adaptadora *MouseAdapter* y se encarga de que haya menús desplegables dentro del panel graph. Se instancia un objeto de esta clase para cada panel graph(*eibGraphPanel*). La forma de hacer esto es mediante una llamada al constructor de esta clase desde el constructor de la clase *eibGraphPanel*. Los métodos que define son los siguientes:

- *mousePressed()*: Este método es el que lanza el evento de que aparezca un popup cuando se pulsa un botón del ratón.
- *mouseReleased()*: Este método se encarga de lanzar el evento cuando se libera un botón del ratón.
- *initPopup()*: Este método se encarga de crear un popup o menú desplegable inicial, independiente y valido para todo el panel.
- *maybeShowPopup()*: Este método se encarga de modificar el menú desplegable si cuando pulsamos lo hacemos sobre un elemento *eibDefaultObject*. Para ello, se encarga de llamar al método *maybeShowPopup()* de estos objetos.

La clase eibGrpahPanelMouseListener

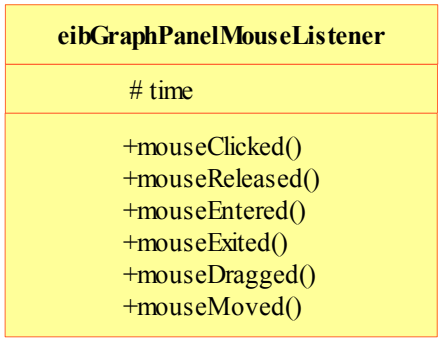

Figura 4.47: Clase eibGraphPanelMouseListener.

Esta clase hereda de *MouseMotionListener* y *MouseListener*. Se encarga de sobrescribir los métodos de éstas que son abstractos. Define que acciones se van a ejecutar al actuar sobre el ratón. Se ha creado para simular el botón derecho del ratón en pantallas táctiles. En definitiva, se encarga de ejecutar una serie de acciones de tal forma que, cuando se pulse una vez sobre un elemento del panel y se mantenga pulsado dicho elemento durante un cierto tiempo aparezca un popup (Simula el botón derecho del ratón).

La clase eibGraphUI

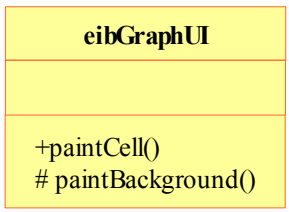

Figura 4.48: Clase eibGraphUI.

Esta es la clase que se va a encargar de definir los métodos que pinten los elementos de la interfaz gráfica. Extiende de BasicGraphUI que se localiza en el paquete jgraph.plaf.basic y los métodos que define son los siguientes:

- *paintCell()*: Este método se encarga de definir como se ha de pintar una celda para los diferentes tipos de celdas que definen las librerías jgraph.
- *paintBackground()*: Este método es el que se encarga de pintar la celda que representa el fondo y que en nuestro software serán las diferentes plantas.

La clase eibVertexView

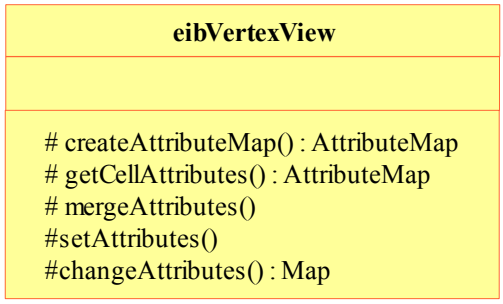

Figura 4.49: Clase eibVertexview.

Las librerias jgraph definen tres tipos de elementos, los vértices, los puentes y los puertos. En el sistema creado se utilizan los vértice. Esta clase extiende de VertexView y se encarga de definir la vista de este tipo de elementos, así como otras propiedades de los mismos.

La clase eibViewFactory.

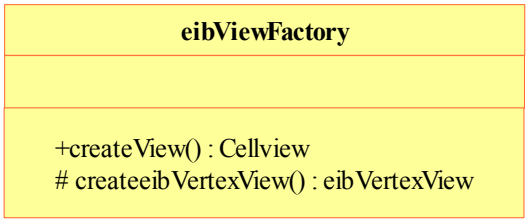

Figura 4.50: Clase eibViewFactory.

Esta clase define los métodos que se encargan de crear los elementos del tipo vértice. Implementa las interfaces *CellViewFactory* y *Serializable* por lo que tendrá que sobrescribir los métodos que definen éstas.

- *createView()*: Este método se encarga de llamar al método que crea un vértice y que no es otro que *createeibVertexView()*.
- *createeibVertexView()*: Se encarga de instanciar un objeto *eibVertexView*.

La clase eibIconLib.

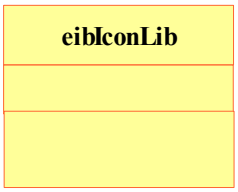

Figura 4.51: Clase eibIconLib.

Esta clase se encarga de definir una serie de constantes que relacionarán a éstas con los icono que representarán a los elementos controlados por los dispositivos EIB.

La clase eibSSH2.

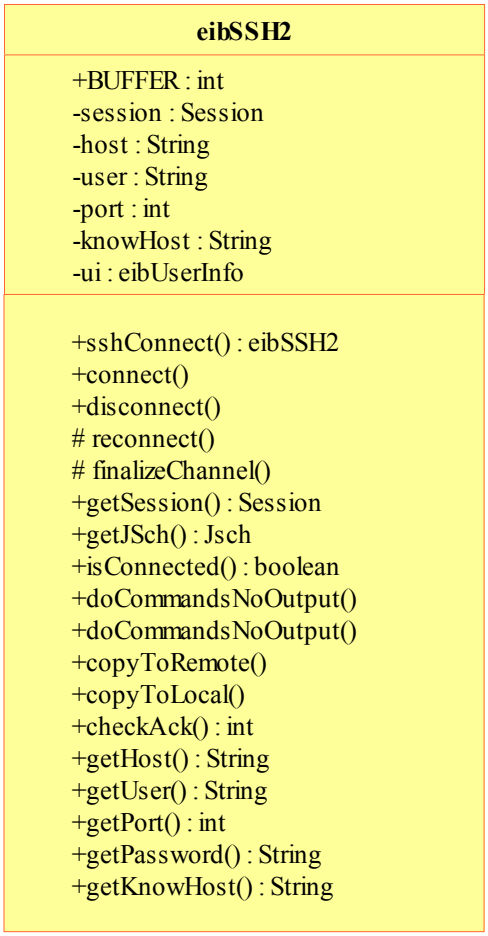

Figura 4.52: Clase eibSSH2.

Esta clase se encarga de crear una conexión SSH con el equipo remoto. También se encarga de definir los métodos para que se puedan enviar los comandos. Si recordamos, las comunicaciones entre el equipo de la interfaz gráfica y el equipo LINCE se realizaban mediante comandos ssh. Esta es la clase que se encarga de definir los métodos que van a posibilitar el llevar a cabo estas comunicaciones. Los métodos que define son los siguientes:

- *sshConnect()*: Este método crea una conexión SSH llamando al método *connect()* y devuelve dicha conexión.
- *connect()*: Se encarga de crear la conexión con lo atributos propios de la clase.
- *disconnect()*: Desconecta la sesión SSH establecida.
- *reconect()*: Este método vuelve a intentar la conexión SSH con los atributos de la clase en caso de que falle un intento de conexión anterior.
- *finalizeChannel()*: Se encarga de finalizar el canal de comunicaciones abierto.
- *getSession()*: Devuelve la sesión abierta.
- *getJSch()*: Devuelve el elemento JSch que crea la conexión.
- *isConnected()*: Devuelve un valor booleano diciendo si la sesión abierta esta activa.
- *doCommandsNoOutput()*: En este primer formato recibe como parámetro un *String* y llama a la segunda forma de este mismo método.
- *doCommandsNoOutput()*: Esta segunda forma se encarga de abrir un canal y de mandar el comando que se le introduce mediante un parámetro del tipo *String []*.
- *copyToRemote()*: Este método se encarga de copiar el fichero local en el equipo remoto. Las rutas completas de ambos ficheros se introducen como parámetros.
- *copyToLocal()*: Copia el fichero remoto en el local. Recibe como parámetros las rutas completas de ambos archivos.
- *checkAck()*: Este método devuelve un entero con el tipo de error que ha acontecido en un *InputStream* y lo muestra por la salida estándar. Devuelve 0 ó -1 en caso de que no se produzca ninguno.
- *getHost()*: Devuelve una cadena con el nombre del host remoto al que se intenta acceder.
- *getUser()*: Devuelve una cadena con el nombre del usuario que intenta acceder al host.
- *getPort()*: Devuelve un entero que representa al puerto por el que se intenta acceder al host.
- *getPassword()*: Devuelve una cadena con el password del usuario que intenta acceder al host.
- *getKnowHost()*: Devuelve una cadena con la ruta en la que se encuentra el programa.

La clase eibUserInfo.

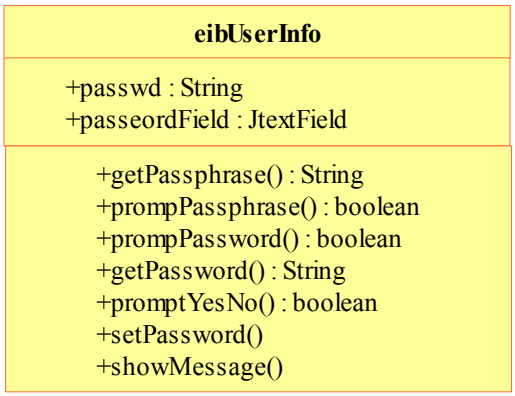

Figura 4.53: Clase eibUserInfo.

Esta clase es la que se encarga de verificar si la información de usuario, que se está introduciendo para crear las conexiones SSH, es la correcta. Implementa las interfaces *UserInfo* y *UIKeyboardInteractive* por lo que ha de sobrescribir todos los métodos de éstas.

- *getPassphrase()*: Este método ha de ser sobrescrito y en nuestro caso no realizará ninguna acción.
- *promptPassphrase()*: Como el anterior, este método ha de ser sobrescrito, y puesto que no se utiliza, devuelve el valor false.
- *PromptPassword()*: Muestra un cuadro de dialogo en caso de que el password introducido como parámetro al hacer la conexión no sea el correcto.
- *getPassword()*: Devuelve un *String*, con el valor del atributo passwd.
- *promptYesNo()*: Devuelve un valor booleano para indicar si el password introducido es o no correcto.
- *detPassword()*: Asigna el valor del password a la variable passwd.
- *showMessage()*: Muestra un cuadro de información con el mensaje que se le pasa como parámetro.

La clase eibAsyncTask.

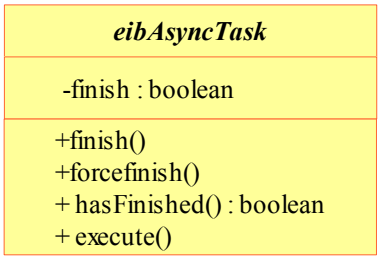

Figura 4.54: Clase eibAsyncTask.

Esta clase hereda de *AsynckTask*, la cual se encuentra en el paquete foxtrop. Esta clase se encarga de crear los hilos que se ejecutan en paralelo al programa, y que surgen como consecuencia de las acciones que se van desarrollando en éste. Es una clase abstracta, por tanto, sus métodos abstractos se han de sobrescribir. Los métodos que define *eibAsyncTask* son:

- *finish()*: Este método se sobrescribe de la clase padre y no realiza ninguna acción.
- *forcefinish()*: Este método fuerza el valor del atributo *finish* a true y se utilizará para cuando sea necesario forzar una salida del hilo.
- *hasFinished()*: Devuelve un valor booleano indicando si se el hilo ha terminado.
- *execute()*: Incorpora el hilo a la cola del gestor de hilos.

La clase eibDefaultObjectTask.

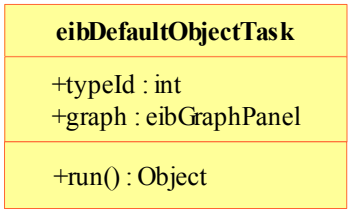

Figura 4.55: Clase eibDefualtObjectTask.

Esta clase hereda de la clase *eibAsyncTask*. Se encarga de insertar elementos controlados por dispositivos EIB dentro de un panel *eibGraphPanel*. Esta inserción se hace como como un hilo independiente. Tan solo define el constructor y el método *run()* que se encarga de la inserción mencionada.

La clase eibSshTask.

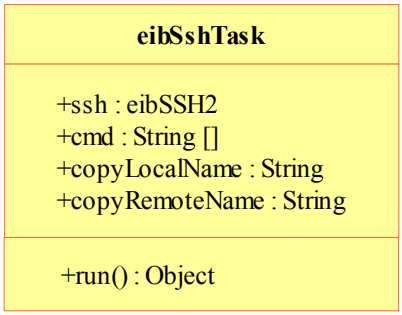

Figura 4.56: Clase eibSshTask.

Esta clase hereda de la clase *eibAsyncTask* y se encarga de lanzar los comandos SSH para interactuar con los dispositivos EIB. Esta acción también se ejecuta como un hilo independientes. Define el método *run()* que se encargará de llamar a las funciones para lanzar los comandos mencionados.

La clase eibFileUtils.

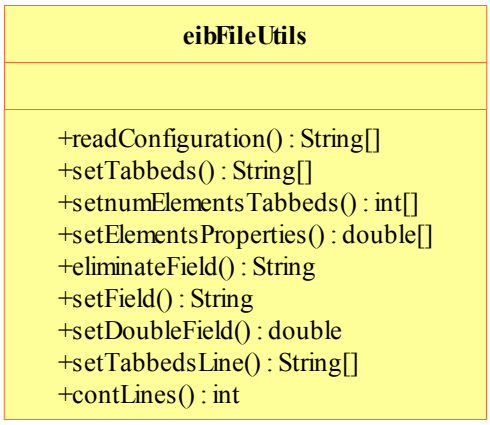

Figura 4.57: Clase eibFileUtils.

Esta clase define una serie de métodos, los cuales se utilizan cuando se lee la configuración de una instalación EIB desde un archivo. Los métodos que define son los siguientes.

- *readConfiguration()*: Este métodos devuelve un *String[]* en el que se almacena todas las características de la instalación. Las características de la instalación se encuentran almacenadas en el archivo que recibe como parámetro, el cual ha de seguir un formato específico.
- *setTabbeds()*: Recibe como parámetro lo devuelto por el método anterior. De éste objeto *String[]* obtiene el nombre de los *tabbeds* que representarán cada una de las plantas del edificio en el que está la instalación.
- *setnumElementsTabbeds()*: Devuelve un *array* de enteros en el que cada elemento representa el número dispositivos que hay en cada planta del edificio. Distingue entre los diferentes tipos de objetos(luces, persianas y aspersores) y para cada uno de ello utiliza un elemento del *array*.
- *setElementsProperties()*: Este método devuelve un *double[]*, el cual contiene las propiedades (posición y tipo de elemento) de todos los elementos de la instalación.
- *eliminateField()*: Devuelve una cadena a la que se le ha quitado uno de los campos. Cada uno de estos campos representa una característica de un elemento de la instalación.
- *setField()*: Devuelve un *String* con el primer campo de la cadena que identifica a un elemento individual.
- *setDoubleField()*: Hace lo mismo que la anterior, pero en este caso el campo devuelto es del tipo *double.*
- *setTabbedsLine()*: Devuelve un *array* de elementos *String* con los nombres de las plantas. Se llama desde el método *setTabbeds()*.
- *contLines()*: Cuenta las lineas que hay en un fichero.

La clase eibUtils.

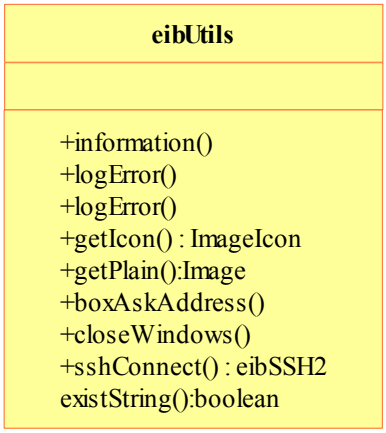

Figura 4.58: Clase eibUtils.

Esta clase define algunos métodos de uso general los cuales se usan a lo largo del programa. Los métodos que define son lo siguientes.

- *Information()*:Este método se encarga de mostrar un cuadro de información. Dicha información se le introduce como una cadena.
- *logError()*: Se encarga de mostrar por pantalla un error. Al igual que el anterior, dicho error se le introduce como parámetro.
- *logError()*: en esta segunda forma además recibe como parámetro un objeto *Exception*.
- *getIcon()*: Devuelve el icono cuyo nombre coincide con el que se introduce como parámetro.
- *getPlain()*: Devuelve un objeto *Image* que representa al plano que lleva por nombre el que recibe como parámetro.
- *boxAskAdrres()*: Esta método devuelve un cuadro de dialogo que se utiliza para introducir las direcciones de grupo.
- *closeWindows()*: Este método se encarga de sobrescribir los métodos de la interfaz *WindowsListener*.
- *sshConnect()*: Se encarga de abrir una conexión y de cerrarla en caso de que cierre el programa de forma inesperada.
- *existString()*: Devuelve true si el valor de un *String* se encuentra dentro de una cadena de *Strings* y false en caso contrario.

La interfaz eibConstants.

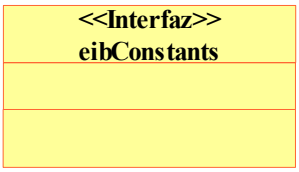

Figura 4.59: Interfaz eibConstants.

Esta interfaz define una serie de constantes que se utilizan a lo largo del programa y que nos facilitan la programación del mismo.

La clase eibGlobals.

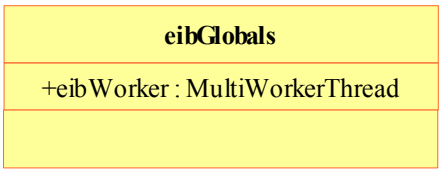

Figura 4.60: Clase eibGlobals.

Esta clase define las variables que serán de uso global a lo largo del programa y que podrán ser utilizadas en cualquier momento por los distintos objetos del mismo. Entre ellas merece especial mención la variable que gestiona los hilos.

La clase principal.

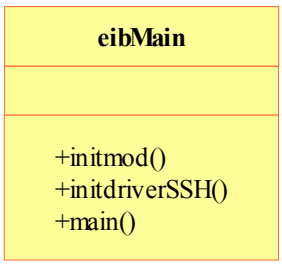

Figura 4.61: Clase eibMain.

Esta clase define el método *main()* que carga el programa y lo ejecuta. Define tres métodos:

- *initmod()*: Este método se encarga de crear un cuadro que nos permite elegir el modo en el que vamos a entrar al programa.
- *initdriverSSH()*: Inicializa el driver para dispositivos EIB con los comandos vistos en un apartado anterior.
- *main()*: Es el método principal, que ejecuta el programa y lo lanza. Lo primero que hace es preguntar el modo en el que queremos iniciarlo. Después lanza el driver y a continuación, comprueba si existe fichero de configuración. En función de si este último existe o no se arrancará de una manera o de otra.

4.3.4 Resumen de la interfaz gráfica.

Para finalizar con el sistema de clases que representa la interfaz diremos que: se fundamenta en las propiedades aportadas por las diferentes librerías utilizadas para su desarrollo. Así, desde un punto de vista gráfico, estará dotada con las propiedades aportadas por las clases Swing y Jgraph. Mientras que, desde un punto de vista más interno, se caracterizan por el uso de las librerías Jsch usadas para la comunicación mediante protocolo seguro.

Esta parte del sistema se comunica a través de comandos del tipo linea de ejecución. Estos son pasados al sistema LINCE codificados en protocolo SSH, el cual los interpreta y los convierte en comandos a nivel de enlace. La forma de comunicarse con el LINCE es a través de una tarjeta ethernet, por tanto, el equipo en el que se instale esta parte del software ha de constar de un elemento de este tipo.

El sistema se ha diseñado para ser instalado en una pantalla de tipo táctil. Así, podrá funcionar sin ratón ni teclado y, de tal forma, que no sea necesario un sistema con este tipo de periféricos.

En cuanto a su estructura, la interfaz ha sido programada mediante el uso de las clases básicas especificadas por las librerías que utiliza. Nunca mediante el uso de interfaces de desarrollo, las cuales permiten especificar los elementos de forma automática. Sin embargo, este tipo de interfaces aumentan en gran medida el número de líneas de código y, por tanto, el número de recursos que se utilizarían a la hora de la ejecución. Además, con la estrategia utilizada se hace uso de una de las herramientas más potentes que da la POO y que no es otra que la herencia. Teniendo así, clases particularizadas para este software, fácilmente ampliables y que permitirán caracterizar y diferenciar unas instalaciones de otras.

La estructura de archivos en la que se incorporan las clases es uno de los pilares de la interfaz ya que ésta contempla la posibilidad de ampliaciones futuras, relaciones entre clases y dependencias de las mismas.

Los usuarios de esta interfaz son los beneficiarios de la instalación domótica y ésta se ha desarrollado para que dichos usuarios no necesiten de grandes conocimientos informáticos para poder utilizarla.