UNIVERSIDAD POLITÉCNICA DE CARTAGENA

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES. ESPECIALIDAD TELEMÁTICA.

PROYECTO FIN DE CARRERA.

CONTROL A TRAVÉS DE WIIMOTE DE APLICACIONES DOMÓTICAS (WIIHOME)

ALUMNO: Fuensanta Blanco Pascual.

PROFESOR: Mª Francisca Rosique Contreras.

ÍNDICE.

ÍNDICE DE IMÁGENES.

CAPÍTULO 1. Presentación del proyecto.

1.1. Introducción.

La evolución de los sistemas de telecomunicaciones, informática y electrónica, junto con la integración de los sistemas de información en la vida cotidiana han permitido la consolidación de los sistemas domóticos en la actualidad. Estos sistemas ofrecen a las personas la posibilidad de controlar y gestionar un conjunto de servicios que mejoran la seguridad, el ahorro energético, el confort, las comunicaciones, y la calidad de vida de las personas que habitan dichas viviendas.

Para la gestión de estos sistemas existen numerosos dispositivos e interfaces hombre-máquina (HMI), que permiten controlar el sistema domótico de una forma fácil e intuitiva, ya que, suelen proporcionar una interfaz de comunicación con el usuario bastante clara y la manera de hacer uso de las mismas es cada vez más sencilla. Debido a esto, estas interfaces tienen una alta aceptación entre la sociedad y es por esta razón, entre otras, por la que se ha decido a desarrollar una interfaz de este estilo en este proyecto.

Además, el sector de la domótica ha evolucionado considerablemente en los últimos años, y en la actualidad ofrece una oferta más consolidada, por lo que, el desarrollo y por tanto, uso de aplicaciones domóticas está cada vez más instaurado en la vida cotidiana y por consiguiente cada vez se desarrollan más métodos para manejar aplicaciones que den el control de los diferentes dispositivos domóticos.

Entre las interfaces hombre-máquina existentes, para este proyecto se ha buscado una interfaz de bajo coste que permita una gestión inalámbrica del sistema domótico, por lo que el presente proyecto consistirá en el desarrollo de una aplicación para el control inalámbrico de sistemas domóticos, empleando para ello el mando controlador de la videoconsola *Wii* de Nintendo, ***WiiMote***.

1.2. Motivación.

El principal motivo que impulsa el desarrollo de este proyecto es la creación de una interfaz usuario-máquina que permita el control automatizado del hogar y que sea **accesible** para los usuarios y de **bajo coste**. Además, como se ha comentado, lo que se lleva a cabo en este proyecto proporciona una gran **originalidad** a la hora de manejar las diferentes luces o aparatos del hogar.

La aplicación aquí descrita ofrece las tres características anteriormente mencionadas, puesto que es **accesible**, ya que cualquier persona puede acceder a ella y hacer uso de la misma **sin** que conlleve un **gasto excesivo** ya que únicamente será necesario adquirir los dispositivos X10 (tecnología domótica usada) y el mando de la videoconsola *Wii***, *WiiMote***, al cual se puede acceder sin necesidad de adquirir dicha videoconsola debido a su carácter autónomo que después explicaremos. Y por último su **originalidad** reside en emplear dicho mando como interfaz entre el usuario y la red domotizada creada por el usuario.

Al no existir demasiadas aplicaciones desarrolladas en el ámbito anteriormente descrito y que proporcionen las características requeridas por la aplicación creada, se va a proceder al estudio de las diferentes posibilidades que se ofrecen.

1.3. Objetivos.

Como ya se ha mencionado, se va a emplear como dispositivo de entrada para la aplicación del PC que aquí se va a desarrollar, el mando de la videoconsola *Wii* de Nintendo, ***WiiMote,*** el cual está dotado de conectividad *bluetooth* (lo que proporciona una gran facilidad para la conexión del mando con el PC). Por tanto se va a usar como **interfaz de usuario** y por consiguiente como **herramienta** para **gestionar** el **sistema X10.**

El principal objetivo que impulsa el desarrollo del presente proyecto es la creación de una aplicación que aproveche y ponga a prueba la gran mayoría de funcionalidades que ofrece la consola *Wii* de Nintendo®, la cual emplea tanto la pulsación de botones como los movimientos del usuario (los cuales son detectados a través del mando/controlador ***WiiMote***) como forma de interacción con el sistema, y que al mismo tiempo proporcione una gran originalidad a la hora de manejar las luces y aparatos del hogar. Además, a pesar de ser ésta, una videoconsola con gran éxito en el ámbito comercial puesto que es de las más vendidas en todo el mundo, existen pocas aplicaciones desarrolladas para PC que obtengan el máximo rendimiento de los recursos de la misma, por tanto, esto se convierte en otro de los motivos para llevar a cabo este proyecto, la creación de una aplicación innovadora en lo que a este aspecto se refiere.

Por tanto, como consecuencia del empleo de esta videoconsola tan particular y al mismo tiempo sencilla de manejar se pretende desarrollar una **aplicación domótica innovadora** y **fácil** de usar que ofrezca al usuario el control de los diferentes módulos de su vivienda empleando como única interfaz el mando de la consola *Wii* en el cual se basa el desarrollo del proyecto (***WiiMote***). La gran ventaja que ofrece el dispositivo ***WiiMote***, que no ofrecen otros dispositivos en el mercado similares, es que consigue unificar en un mismo dispositivo distintas tecnologías a un precio bastante asequible, gracias a su producción a gran escala. Además, como se ha comentado, no es necesaria la adquisición de una videoconsola *Wii* para hacer uso del ***WiiMote*** con las aplicaciones desarrolladas para PC dado su carácter autónomo.

Por tanto, se fija como objetivo principal del proyecto el desarrollo de la aplicación denominada ***WiiHome*** (*Wii + domótica*).

Por supuesto para llegar a conseguir este objetivo y poder por tanto llevar a cabo y finalizar este proyecto se han fijado una serie de objetivos secundarios aunque necesarios para el desarrollo del proyecto:

- Identificación y estudio de las aplicaciones desarrolladas actualmente tanto para el ***WiiMote*** como en el ámbito de la **domótica**.

- Búsqueda y posterior utilización de librerías desarrolladas en java para el uso de ***WiiMote***.

- El objetivo anterior y en general el objetivo principal del proyecto conlleva tanto el estudio previo del lenguaje de programación empleado (java), como el entorno de desarrollo utilizado en este proyecto (Eclipse).

- Estudio y desarrollo del protocolo de comunicación para la transmisión y recepción de información hacia/desde los dispositivos domóticos (x10) empleados en el proyecto.

Finalmente, para la aplicación que se va a proceder a desarrollar en este proyecto serán necesarios principalmente, entre otros, los siguientes elementos:

- ***WiiMote.***

- Comunicación bluetooth (para establecer la conexión entre el PC y el WiiMote).

- Tecnología x10 para el hardware de domótica.

Con todos estos objetivos fijados y llevados a cabo comprobaremos finalmente que el resultado obtenido es el programa para la implementación de la aplicación ***WiiHome*** para el control de lámparas y persianas entre otros módulos del hogar.

1.4. Recursos utilizados.

Emplearemos una serie de recursos para llevar a cabo el proyecto, tanto hardware como software algunos de los cuales detallaremos más en profundidad en capítulos posteriores.

1.4.1. Hardware.

- **PC portátil.** Necesario para el desarrollo y las pruebas de la aplicación.

- ***WiiMote*.** Dispositivo que se empleará como interfaz entre el usuario y la aplicación para la interacción con los dispositivos de la vivienda.

- **Dongle Bluetooth.** Este aparato es necesario para realizar la conexión ***bluetooth*** entre el ***WiiMote*** y el PC. La forma de realizar ésta conexión quedará detallada en el **Anexo 1**.

- **Módulos X10.** Será necesario un controlador CM11A y diferentes actuadores para el encendido/apagado de las luces a través de la red eléctrica, los cuales se detallarán más adelante.

1.4.2. Software.

- **Windows 7.** Se utiliza este sistema operativo para el desarrollo del proyecto aunque en algunos momentos ha presentado incompatibilidades que han sido solventadas.

- **Eclipse.** Se usará este entorno de programación, en el cual, con el lenguaje de programación ***java*** se establecerá la comunicación tanto con el mando como los módulos domóticos.

- **WiiuseJ.** Librería java para el control del ***WiiMote***.

- **ActiveHome Pro.** Software empleado para la programación del escenario de dispositivos que deseamos. Este programa también proporciona la base del protocolo que usaremos para la comunicación con los dispositivos.

1.5. Distribución de la memoria.

En este subapartado se va a detallar cómo está dividida la memoria y cuáles son los aspectos más significativos que se van a detallar en cada capítulo y anexo, para tener de esta manera una visión global de lo que en el presente documento se explica.

La memoria está dividida en 6 capítulos y X anexos. A continuación se va a detallar lo que contiene cada uno de ellos.

- En el primer capítulo, como se ha visto se da una breve introducción de las bases del proyecto y lo que ha llevado a desarrollar el mismo, así como los objetivos fijados.

- En el capítulo 2 se realiza un estudio sobre la domótica y sobre el controlador ***WiiMote***, es decir, las dos partes principales de las que está formado el proyecto. En lo referente a la domótica, se hace referencia, entre otras cosas, a lo que es, su historia, los beneficios que nos aporta, etc…y en lo que al mando ***WiiMote*** se refiere se detallan sobre todo las funcionalidades que aporta, así como su conectividad y los diferentes accesorios que se pueden añadir, entre otras cosas.

- En el tercer capítulo se da una explicación del software utilizado, tanto en lo referente al controlador ***WiiMote*** (librería *WiiUseJ*), como en lo referente a la domótica, para lo cual se ha empleado como interfaz de comunicación con los dispositivos X10 el programa **Active Home Pro**.

- En el capítulo 4 se detalla el funcionamiento de la aplicación java creada, revisando el código y explicando lo que se ha utilizado para cada función que se quería llevar a cabo.

- En el capítulo 5, se explica con detalle el caso de estudio de nuestro proyecto, en el que se detalla el material del que se disponía para realizar las pruebas y cómo se va creando la interfaz de comunicación con el usuario.

- En el capítulo 6 se detallan las conclusiones y los trabajos futuros a realizar.

- Y finalmente en el capítulo 7, la bibliografía usada.

- Por último en los anexos se describe lo siguiente: en el anexo 1 cómo realizar el enlace a través de la conexión *bluetooth* del ***WiiMote*** con el PC. En el anexo 2 se detalla cómo se comienza a crear el proyecto y lo necesario para dicha tarea. En el anexo 3 cómo se establecen las comunicaciones con los dispositivos x10 (a través del archivo *ahcmd.exe*). En el anexo 4 se detalla la documentación de la API *wiiusej*. En el anexo 5 se presenta el manual de configuración y uso del software utilizado para el manejo de los dispositivos domóticos. En el anexo 6 se presenta todo el código del proyecto desarrollado.

CAPÍTULO 2.

Marco teórico.

En este segundo capítulo del proyecto ofreceremos un análisis de la base teórica tanto del aspecto domótico como de la videoconsola *Wii*, particularmente del controlador o mando ***WiiMote***.

Todo lo que aquí se explica es necesario conocerlo tanto para el desarrollo de la aplicación como para entender cada uno de los objetivos fijados anteriormente y por tanto llegar a comprender el cómo y el porqué del desarrollo del proyecto aquí presentado.

Se ofrece por tanto, en este apartado, una introducción a alguno de los componentes que posteriormente analizaremos y trataremos más en detalle como es el caso de los dispositivos X10 empleados en el proyecto.

2.1. Introducción a la domótica.

2.1.1. ¿Qué es la domótica?

La domótica consiste en una serie de sistemas tecnológicos que aportan diferentes servicios al hogar, estos servicios pueden ser de seguridad, bienestar, comunicación, de gestión energética, etc. La domótica está integrada por redes de comunicación tanto interiores cómo exteriores ya sea de forma inalámbrica o alambrada. Ésta no sólo va dirigida a las viviendas, sino también a los comercios, edificios, granjas, etc. La domótica se ha implantado desde hace décadas, pero, desde que se creó Internet, éste ha dado un giro controversial, los modelos tecnológicos relacionados con Internet han progresado y forman parte del futuro de la domótica. Las tecnologías inalámbricas WiFi y las redes de Internet, se han constituido como las tecnologías del entorno digital que evolucionarán, y sobre las cuales la domótica debería mantenerse para poder aumentar el uso de las tecnologías en los hogares. Las ventajas y los servicios que proporcionan este tipo de sistemas son los siguientes:

- El incremento del confort.
- Climatización del control de luces, ventanas, cortinas, persianas y enchufes.
- Uso de energía renovable como la energía solar, la geotérmica y la eólica.

Figura 2.1. Ilustración vivienda domótica [1].

2.1.2. Historia de la domótica.

Desde los inicios formales de la automatización de hogares han pasado ya bastantes años. Los cambios han sido enormes y el crecimiento mucho más, sin embargo, afortunadamente quedan números aspectos que tienen que evolucionar mucho más. Hoy en día, aún no existen unas normas oficiales específicas sobre este tipo de tecnología.

No obstante es imposible no recalcar el crecimiento que ha tenido la implementación de la domótica a lo largo de su breve historia en la vida diaria de las personas, aquellas que tomaron la decisión de convertir su vivienda en un hogar digital. En la actualidad encontramos pantallas táctiles por menos de 500 euros, tenemos una importante cantidad de teléfonos móviles inteligentes (smartphones) que sirven como controladores domóticos para automatizar todo el hogar y además en la actualidad todas las viviendas están dotadas de cómo mínimo un ordenador. Las cosas han cambiado y con ellas la historia de la domótica aún se está escribiendo.

2.1.3. Evolución de la domótica en España.

Los orígenes de la domótica a España se pueden situar alrededor del año 1990, fecha en la que se empiezan a llevar a cabo las primeras iniciativas e investigaciones. Al principio, el mercado se caracterizaba por un gran desconocimiento de la domótica tanto en el ámbito tecnológico como de posibilidades y aplicaciones, por lo que el interés que suscitaba este adelanto tecnológico era muy limitado y su investigación mínima.

Los primeros sistemas estaban poco integrados y las áreas de gestión que se cubrían eran, a duras penas, el aspecto de la confortabilidad y la seguridad, aunque también cabe destacar que había otras aplicaciones más aisladas tales como la gestión de las comunicaciones y la energía.

A continuación se muestra una clasificación de las características de aquel mercado al que se enfrentaban los sistemas domóticos españoles:

- Generalmente, los productos estaban fabricados atendiéndose a las normativas europeas y destinados a mercados extranjeros más desarrollados.
- Había ciertas dificultades a la hora de diseñar e instalar dispositivos, al carecer de suficiente personal cualificado.
- El coste de las instalaciones era muy elevado y éstas resultaban poco productivas.
- No había entidades públicas o privadas especializadas en instalaciones de este tipo ni interés por abrir un mercado.
- Había desconfianza y reticencia por parte de los usuarios al encontrarse delante de algo que podía poner en riesgo la seguridad de los edificios o viviendas (o incluso los mismos usuarios) debido a una excesiva automatización.

A pesar de que actualmente la situación se diferencia notablemente respecto otros países, no hay lugar a dudas de que en los próximos diez años las instalaciones automatizadas serán un valor añadido de las construcciones, ya que el mercado actual se caracteriza por los siguientes aspectos:

- Creación de nuevas empresas dedicadas a la fabricación e instalación de sistemas automatizados.

- Avances en la normalización y homologación de determinados productos así cómo el rechazo a otros que no cumplen la normativa tecnológica española y/o europea.

- Desarrollo de nuevos sistemas por parte de las empresas del sector electrónico.

- Creación de organismos de investigación y desarrollo (I+D). El progreso de la domótica española no habría podido llevarse a cabo sin los organismos e instituciones dedicadas a su óptimo desarrollo que, asumiendo un cierto riesgo económico considerable han apostado por esta nueva tecnología.

- La apuesta por el progreso y la innovación mediante sistemas domóticos y la financiación por parte de las instituciones públicas junto con la Comunidad Económica Europea de proyectos I+D.

El mercado español cuenta con más de 25 sistemas (casas equipadas con componentes automatizados) domóticos y un gran número de productos con prestaciones cada vez más atractivas y asequibles para los usuarios no familiarizados con el sector. Por otro lado, dichos sistemas se están implantando en el 60% de hogares de nueva construcción y en el 40% de hogares ya existentes, lo que implica una paulatina normalización (aún estando en periodo de crecimiento) de los dispositivos automatizados a nuestras casas.

Hasta la actualidad el usuario de una instalación eléctrica convencional se conformaba sencillamente con iluminarse, calentarse y disponer de acceso al tendido eléctrico para conectar los electrodomésticos y demás componentes eléctricos. Los requisitos de una instalación se limitaban a proteger las líneas y las personas contra los riesgos eléctricos.

Pero posteriormente, a las funciones tradicionales se han añadido nuevas funciones y productos que gestionan la energía y el confort como aparatos que permiten aplicaciones especificas como programar la calefacción, regular la temperatura ambiental, gestionar el consumo de energía, etc.

La incorporación de estas instalaciones singulares ha supuesto mayor complejidad, aumentando el cableado interno de la vivienda y provocando que una ampliación y/o modificación de dicha instalación se traduzca en largas y costosas intervenciones por parte del instalador.

2.1.4. Beneficios.

Los beneficios aportados por la domótica son múltiples. Los agruparemos en:

- El ahorro energético gracias a una gestión tarifaria e "inteligente" de los sistemas y consumos.

- La potenciación y enriquecimiento de la propia red de comunicaciones.

- La más contundente seguridad personal y patrimonial.

- La tele-asistencia.

- La gestión remota (vía teléfono, radio, Internet, etc.) de instalaciones y equipos domésticos.

- Aumento del bienestar y en definitiva, del confort.

2.1.5. Aplicaciones.

La domótica busca el aprovechamiento al máximo de la energía y luz solar adecuando su comportamiento a nuestras necesidades.

Las posibles aplicaciones son innumerables dadas las posibilidades que ofrece la domótica. A continuación se describirán las más importantes y las que son más relevantes en este proyecto:

En el ámbito del nivel de confort.

- Control de todos los dispositivos instalados y operativos desde un dispositivo central simplificando su gestión y optimizando su uso.

- Apagado general de todas las luces de la vivienda.

- Automatización del apagado/encendido en cada punto de luz. La forma de encender y apagar la iluminación de la vivienda puede ser automatizada y controlada de formas complementarias al control tradicional a través del interruptor clásico. La iluminación puede ser regulada en función del nivel de luminosidad ambiente, evitando su encendido innecesario o adaptándola a las necesidades del usuario. La activación de ésta se realiza siempre cuando el nivel de luminosidad supera un determinado umbral, ajustable por parte del usuario. Esto garantiza un nivel de iluminación mínimo, que puede ser especialmente útil para por ejemplo un pasillo o la iluminación exterior.

- Regulación de la iluminación según el nivel de luminosidad ambiente.

- Automatización de todos los distintos sistemas/ instalaciones / equipos dotándolos de control eficiente y de fácil manejo.

En el ámbito de las comunicaciones.

- Control remoto.

- **Dentro de la vivienda.** A través de un esquema de comunicación con los distintos equipos (mando a distancia, bus de comunicación, etc.). Reduce la necesidad de moverse dentro de la vivienda, este hecho puede ser particularmente importante en el caso de personas de la tercera edad o movilidad reducida.

- **Fuera de la vivienda.** Presupone un cambio en los horarios en los que se realizan las tareas domésticas (por ejemplo: la posibilidad de que el usuario pueda activar la cocina desde el exterior de su vivienda, implica que previamente ha de preparar los alimentos) y como consecuencia permite al usuario un mejor aprovechamiento de su tiempo.

- Transmisión de alarmas.

- Intercomunicación entre las habitaciones.

Existen otros ámbitos cómo son el ahorro energético y la seguridad que también son de gran importancia en la domótica. Cualquier sistema domótico debe proporcionar estos cuatro servicios.

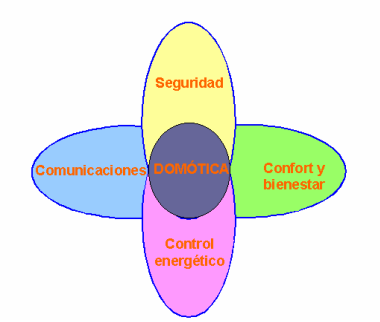


Figura 2.2. Aspectos integrados en la domótica.

2.1.6. Estándares domóticos.

En un sistema domótico la interacción con el entorno físico del sistema es un factor muy importante. En el desarrollo de un sistema domótico es necesario establecer mecanismos para que pueda extraer información del entorno y realizar acciones sobre él. Por otra parte, hay que destacar que un aspecto importante de los sistemas domóticos es la integración de los distintos tipos de servicios que debe ofrecer: automatización, seguridad, comunicaciones, multimedia, etc. y para ello se valdrá tanto de elementos hardware (sensores, actuadores, cableado...) como de software (video bajo demanda, mensajería electrónica, etc.).

Por todo esto, es necesario un lenguaje de modelado específico que tenga en cuenta estas características y suministren a los usuarios un sistema mediante el cual se proporcionen de forma cómoda, eficiente y económica solución a sus necesidades.

Los estándares domóticos más importantes son:

2.1.6.1. KNX\EIB.

Éste es el sistema más conocido y desarrollado en Europa. Consiste en un sistema domótico basado en un bus de datos. A diferencia de X10, que utiliza la red eléctrica, y otros sistemas actuales que se comunican por radio frecuencia (RF), el EIB utiliza su propio cableado, con lo cual se ha de proceder a instalar las conducciones adecuadas en el hogar y para el sistema. Es un completo sistema integrado de automatización y control de edificios y viviendas, destinado a la aplicación de soluciones gradualmente compatibles, flexibles y rentables. Debido a su versatilidad funcional, su uso no se reduce a las instalaciones simples y limitadas sino que también proporciona soluciones para el sector del edificio completo.

Las siglas EIB representan la tecnología de instalaciones de edificios más innovadora (“sistema bus”), promovida desde 1990 por el grupo de fabricantes que engloban la EIBA (Asociación EIB), con sede en Bruselas. EIBA está envuelta en la emisión de las marcas registradas relacionadas con el sistema,

los estándares de comprobación y calidad de los productos, las actividades de marketing y estandarización... . El EIB también es distribuido bajo varias denominaciones diferentes, por ejemplo: instabus, ABB I-Bus, Tebis…

Así, el EIB nació de las exigencias de mayor flexibilidad y comodidad en las instalaciones eléctricas, unidas al deseo de minimizar las necesidades de energía.

Las empresas participantes en EIBA garantizan que sus productos sean compatibles con el bus. Por ello se pueden emplear en una instalación EIB aparatos de distintos fabricantes con total interoperabilidad.

El bus de control (medio de transmisión por pares trenzados) se tiende paralelo al cableado de 230 V, lo cual implica:

- Una reducción considerable de la cantidad total de cable instalado, en comparación con una instalación convencional (hasta un 60%).
- Un incremento del número de funciones posibles del sistema.
- Una mejora en la claridad de la instalación.

Este cable conecta las cargas y los interruptores que las controlan y suministra alimentación a los componentes bus, en la mayoría de los casos.

Al disponer todos los componentes bus de su propia inteligencia, no resulta necesaria una unidad central de control (p. ej. un ordenador). Por lo tanto, el EIB puede ser utilizado tanto para pequeñas instalaciones (viviendas) cómo en proyectos mucho más grandes (hoteles, edificios administrativos, etc).

Gracias a la flexibilidad de la tecnología EIB, cualquier instalación puede ser fácilmente adaptable a las necesidades cambiantes del usuario.

Las ventajas principales que este protocolo ofrece son las siguientes:

- Proporciona una única línea común para controlar, comunicar, y vigilar todas las funciones de servicio y su desarrollo.

- Por esta misma razón, el uso de una línea común, la instalación de un edificio se puede realizar de un modo más sencillo desde el principio y después se puede ampliar y modificar sin problemas.

2.1.6.2. LonWorks.

LonWorks es una plataforma de control creada por la compañía norteamericana Echelon y consiste en un protocolo que proporciona numerosas ventajas; en primer lugar, es un protocolo estándar con uno de los mayores ancho de banda, admite varios medios físicos de comunicación, el lenguaje de programación de las aplicaciones está basado en el lenguaje ANSI C (ampliamente extendido) y es compatible con infinidad de productos de otros fabricantes, ya que además de tener su propio estándar domótico dispone de su propia asociación (Lonmark Internacional) que verifica, valida y certifica productos que trabajan bajo dicho estándar.

Las redes *LonWorks* describen de una manera efectiva una solución completa a los problemas de sistemas de control, como por ejemplo los problemas que conllevan la creación de una arquitectura centralizada, donde existe un "maestro" o controlador principal, físicamente cableado a cada punto de control particular, como actuadores o sensores, denominados "esclavos". El resultado final es funcional, pero es caro y difícil para mantener, ampliar y gestionar. Igualmente, es menos fiable frente a fallos, ya que la caída del controlador principal provoca la caída de todo el sistema.

Por lo que el comienzo de las redes *Lonworks* se basó en conceptos muy simples:

- Los sistemas de control son fundamentalmente idénticos, independientemente de la aplicación final.

- Un sistema de control distribuido es significativamente más potente, flexible, y ampliable que un sistema de control centralizado.

- las empresas ahorran más dinero a largo plazo instalando redes distribuidas que instalando redes centralizadas.

La tecnología *Lonworks* proporciona una solución a los múltiples problemas de diseño, construcción, instalación, y mantenimiento de redes de control; redes que pueden variar en tamaño desde dos a 32,000 dispositivos y se pueden usar en cualquier aplicación desde supermercados a plantas de petrolíferas, desde aviones hasta ferrocarriles, para viviendas particulares o los esquemas de control basados en sistemas centralizados. Los fabricantes están utilizando sistemas abiertos, chips estándar, sistemas operativos estándar y componentes para construir productos que mejoren la flexibilidad, el costo del sistema y su instalación. La tecnología *Lonworks* está acelerando la tendencia a evitar los sistemas propietarios o los sistemas centralizados, proporcionando interoperabilidad, una tecnología robusta, desarrollos más rápidos y ahorro económico.

2.1.6.3. Tecnología o protocolo X10.

X10 es un protocolo de comunicaciones para el control remoto de dispositivos eléctricos. Utiliza la línea eléctrica (220V o 110V) para transmitir señales de control entre equipos de automatización del hogar en formato digital.

El protocolo X10 consta de bits de "direcciones" y de "órdenes". Por ejemplo, permite decir «lámpara #3», «¡enciéndete!» y el sistema procederá a ejecutar dicho mandato.

Los dispositivos están generalmente enchufados en módulos X10 (receptores). X10 distingue entre *módulos de lámparas* y *módulos de dispositivos*. Los módulos de lámpara proporcionan energía y aceptan órdenes X-10. Los módulos de dispositivos son capaces de gestionar cargas grandes (ej. máquinas de café, calentadores, motores, ...), simplemente encendiéndolos y apagándolos.

Actualmente X10 es un protocolo que está presente en el mercado mundial, sobre todo en Norteamérica y Europa (España y Gran Bretaña fundamentalmente).

Éste es el protocolo en el que se basa este proyecto, por lo que posteriormente se dará más información acerca del mismo.

2.1.6.4. Zigbee.

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo, basada en elestándar IEEE 802.15.4 de redes inalámbricas de área personal (*wireless personal area network*, WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

En principio, el ámbito donde se prevé que esta tecnología cobre más fuerza es en domótica, como puede verse en los documentos de la ZigBee Alliance, en las referencias bibliográficas que se dan más abajo en el documento «ZigBee y Domótica». La razón de ello son diversas características que lo diferencian de otras tecnologías:

- Su bajo consumo.
- Su topología de red en malla.
- Su fácil integración (se pueden fabricar nodos con muy poca electrónica).

ZigBee utiliza la banda ISM para usos industriales, científicos y médicos; en concreto, 868 MHz en Europa, 915 en Estados Unidos y 2,4 GHz en todo el mundo. Sin embargo, a la hora de diseñar dispositivos, las empresas optarán prácticamente siempre por la banda de 2,4 GHz, por ser libre en todo el mundo. El desarrollo de la tecnología se centra en la sencillez y el bajo costo más que otras redes inalámbricas semejantes de la familia WPAN, como por ejemplo Bluetooth. El nodo ZigBee más completo requiere en teoría cerca del 10% del hardware de un nodo Bluetooth o Wi-Fi típico; esta cifra baja al 2% para los nodos más sencillos. No obstante, el tamaño del código en sí es bastante mayor y se acerca al 50% del tamaño del de Bluetooth. Se anuncian dispositivos con hasta 128 kB de almacenamiento.

Los protocolos ZigBee están definidos para su uso en aplicaciones encastradas con requerimientos muy bajos de transmisión de datos y consumo energético. Se pretende su uso en aplicaciones de propósito general con características auto organizativas y bajo costo (redes en malla, en concreto). Puede utilizarse para realizar control industrial, albergar sensores empotrados, recolectar datos médicos, ejercer labores de detección de humo o intrusos o domótica. La red en su conjunto utilizará una cantidad muy pequeña de energía de forma que cada dispositivo individual pueda tener una autonomía de hasta 5 años antes de necesitar un recambio en su sistema de alimentación.

A continuación se muestra una tabla comparativa de las tecnologías aquí explicadas, indicando sus ventajas e inconvientes.

Tecnología	Ventajas.	Inconvenientes.
	- Es el protocolo con mayor implantación en el mercado. - Proporciona la herramienta ETS que permite programar dispositivos complejos. - Puede ser usado en nuevas	- Costoso. - No se garantiza la compatibilidad del 100% con productos futuros.

KNX/EIB.	construcciones, así como en construcciones ya existentes. - Puede ser usado para el control de todas las posibles funciones/aplicaciones en casas y edificios. - Soporta diferentes medios de comunicación, así como diferentes modos de configuración.	
LonWorks.	- Mayores anchos de banda. - Permite varios medios físicos de comunicación. - Compatible con productos de otros fabricantes. - Soluciona el problema de la arquitectura centralizada.	- Poco estandarizado. - Presenta problemas con comunicación Wireless.
X10.	- No require cableado adicional, puesto que hace uso de la instalación eléctrica del hogar. - Ahorra energía eléctrica, protección y monitoreo constante de la vivienda. - Uso universal. - Muy extendido en Europa y Estados Unidos.	- A diferencia de EIB el control si es centralizado, puesto que depende del controlador para la ejecución de las órdenes, lo que puede conllevar problemas.
Zigbee.	- Ideal para conexiones punto a punto y punto multipunto. - Diseñado para el direccionamiento y refrescamiento de la red. - Opera en la banda libre de ISM 2.4 Ghz para conexiones inalámbricas. - Óptimo para redes de baja transferencia de datos.	- La tasa de transferencia es muy baja. - Sólo manipula textos pequeños comparado con otras tecnologías. - Trabaja de manera que no puede ser compatible con bluetooth porque no llega a tener las mismas tasas de transferencia, ni la misma capacidad de soporte para nodos. - Tiene menos cobertura porque pertenece a redes inalámbricas de tipo WPAN.

Figura 2.3. Comparación estándares domóticos.

A pesar de que la tabla comparativa demuestra que la tecnología con mayores ventajas y con mayor implantación en el mercado europeo es KNX, para este proyecto se ha seleccionado la plataforma domótica **X10**, ya que permite el montaje de un pequeño caso de estudio para la validación de la propuesta presentada en este PFC, a un coste más reducido y reutilizando material existente en el laboratorio de domótica de la UPCT.

En el siguiente punto se detallan todos los aspectos más relevantes de esta tecnología que serán necesarios conocer para entender el funcionamiento de la aplicación creada.

2.2. Tecnología x10.

2.2.1. Introducción.

Esta tecnología fue desarrollada en 1975 en Glenrothes, Escocia, siendo ésta la primera tecnología domótica de la historia sigue siendo en la actualidad la más extendida y utilizada. Su funcionamiento se basa en la comunicación entre los dispositivos dedicados a la automatización del hogar a través de la red eléctrica existente en cualquier tipo de edificio. Se emplean también para este fin, aunque con menos frecuencia, las ondas de radio.

X10 es el lenguaje de comunicación que utilizan los productos compatibles X10 para hablarse entre ellos y que le permiten controlar las luces y los electrodomésticos de su hogar, aprovechando para ello la instalación eléctrica existente de 220V de su casa, y evitando tener que instalar cables. Los productos de automatización del hogar X10 están diseñados para que puedan ser instalados fácilmente por cualquier persona sin necesidad de conocimientos especiales. Es muy frecuente la utilización de macros para la ejecución de diversas órdenes de alguna manera relacionadas como puede ser la creación de diversos ambientes, por ejemplo el salón, como encender la televisión y apagar las luces en un supuesto “modo cine”, atenuarlas y encender el equipo de música cuando queramos relajarnos o simular presencia a determinadas horas para evitar robos. También es usual la toma de decisiones ante la recepción de determinados valores por parte de sensores, por ejemplo llamar a los bomberos en caso de detección de fuego o encender el aire acondicionado si se supera un determinado umbral de temperatura.

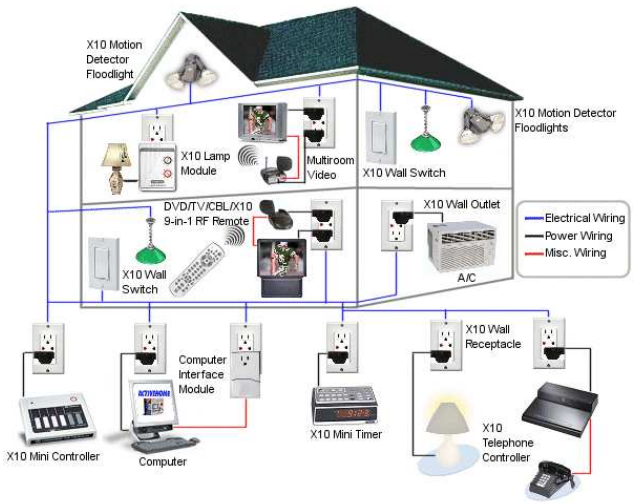


Figura 2.4. Ilustración vivienda domotizada [2].

2.2.2 Dispositivos Hardware.

Existen diferentes dispositivos de los que puede estar formada una red x10:

- Controladores.
- Actuadores.
- Emisores.
- Filtros.
- Sensores.
- Otros dispositivos compatibles.

A continuación se van a detallar los aspectos más relevantes de los dispositivos que se han utilizado en el proyecto.

2.2.2.1. Controladores.

El controlador es el dispositivo esencial en cualquier sistema x10, ya que es el encargado de dar órdenes al resto de dispositivos.

Existen diversos tipos de controladores:

- **Miniprogramador:** dispositivo a través del cual se pueden enviar directamente señales a dispositivos X10 o crear programas o macros.
- **Mandos a distancia RF:** Mandos a distancia universales para el control de electrodomésticos como la televisión además de dispositivos X10 a través de radiofrecuencia.
- **Mandos RF:** Similares a los anteriores pero sólo para el control de dispositivos x10.
- **Programador para PC:** El más completo ya que utilizando software comercial específico o implementando librerías al crear los nuestros propios, se pueden enviar y recoger señales X10 para un control total y sin límites de nuestro hogar. Cuentan con un conector serie o USB para su conexión al PC.

Tanto en el caso del miniprogramador como el de programador para PC suelen estar conectados a la red eléctrica y opcionalmente pueden enviar y recibir señales por radiofrecuencia a dispositivos X10 inalámbricos.

Figura 2.5. Controlador CM11A.

2.2.2.2. Actuadores.

Estos dispositivos son los encargados de escuchar las señales que llegan del controlador, y en caso de estar destinados a ellos, ejecutar la acción correspondiente. Generalmente se conectan a la red eléctrica y a ellos se conectan el dispositivo que controlan (luces, aparatos eléctricos, calefactores, etc.).

A continuación se describen algunos de los módulos que se pueden encontrar en el mercado atendiendo a una clasificación según el tipo de dispositivo que controlan:

- **Módulos de potencia:** se conectan entre un enchufe o al cableado del hogar directamente y al dispositivo a controlar. Cuando reciben la señal de encendido dejan pasar corriente al dispositivo y cortan el suministro si reciben la señal de apagado.

- **Módulos de iluminación o de lámpara:** son receptores de señales X10 que van enchufados a la red y permiten el control de encendido y apagado así como atenuar la luminosidad de una lámpara. Estos dispositivos se conectan al enchufe actual de la lámpara y luego se conectan al módulo.

- **Módulos de persiana:** permiten controlar dispositivos con movimientos en dos direcciones como por ejemplo toldos o persianas.

Figura 2.6. Actuador.

2.2.2.3. Emisores.

A continuación se muestran algunos ejemplos de dispositivos X10 que generan, de alguna forma, señales X10. En concreto se describen los siguientes dispositivos:

- **Receptor de RF.**

  El receptor recibe las señales de los mandos a distancia y envía las órdenes correspondientes para encender y apagar luces y aparatos. Cada receptor es capaz de controlar hasta 16 direcciones X10.

  Además el propio receptor funciona como un modulo de aparato capaz de controlar una potencia de 1000 W, que es cómo se usa en el desarrollo de este proyecto.

Figura 2.7. Receptor de RF.

2.2.2.4. Otros dispositivos domóticos.

A continuación se muestra una tabla resumen dónde se detallan los módulos X10 que pueden ser importante en la creación de una red X10 pero que en el proyecto que aquí se ha desarrollado no se han utilizado.

Emisores de RF (Emisores).		**- ofrece la posibilidad de controlar hasta 16 módulos diferentes de X10.** **- Envía señales vía radio al Receptor de RF o cualquier Consola de Seguridad X10. Compatible con toda la gama de productos X10.**
Micromódulos (Emisores).		**- Permiten enviar y controlar 2 direcciones X10.** **- Además incorpora un módulo de aparato normal, por lo que es capaz de conectar y desconectar localmente cualquier aparato.** **- Incorpora un relé capaz de conectar y desconectar hasta 2200 W de cargas resistivas o 600W de cargas inductivas o capacitiva (motores, fluorescentes, etc.), por lo que se puede emplear con la mayoría de los aparatos y electrodomésticos del hogar.**
Filtros.		**- Se usan para filtrar las señales de las distintas viviendas, debido a que la red eléctrica es compartida con todas, por lo que podría darse el caso de que las señales generadas por los emisores de una de las viviendas actuaran sobre módulos de la otra, y viceversa.** **- También se usan para aislar de la red X10 los aparatos que pudieran crear perturbaciones en la misma, como podrían ser algunos ordenadores, frigoríficos, etc.** **- Y para aislar los módulos X10 de las perturbaciones de los emisores, por si no se recibe correctamente la señal.**

Sensores.		**- Envían mediciones por la línea de comunicación para que sean recibidos y procesados por el controlador.** **- Normalmente utilizan comunicación inalámbrica para enviar datos al controlador.** **- Tipos: sensores no X10, de presencia, termostatos X10.**
Sistemas de seguridad.		**- Disponen de múltiples accesorios, como sensores de presencia, sensores para alarmas técnicas (humo, inundación…), sensores para alarmas medicas, etc...** **- En general las centralitas tienen las siguientes características: funcionan como receptores de RF y como controladores telefónicos, entre otras funciones.**

Figura 2.8. Tabla resumen de dispositivos X10.

2.2.3. Funcionamiento de la tecnología x10.

Este protocolo se basa en el envío de paquetes, bien por el cableado eléctrico del hogar o bien por señales de radiofrecuencia, aunque lo más usado es el cableado eléctrico, puesto que no requiere una instalación adicional. La red de instalación es la base de todo sistema de corrientes portadoras. El elemento básico y fundamental de la técnica de corrientes portadoras es el aprovechamiento doble de la instalación eléctrica ya existente, como conductor de energía y de información. Con los componentes X10 la red, además de suministro de corriente se encarga también de la transmisión de señales de mando (en definitiva paquetes) para los diversos aparatos eléctricos. Con ello se pueden enviar señales de corrientes portadoras a cualquier punto de la instalación que se desee, y a su vez puede solicitarse la información necesaria.

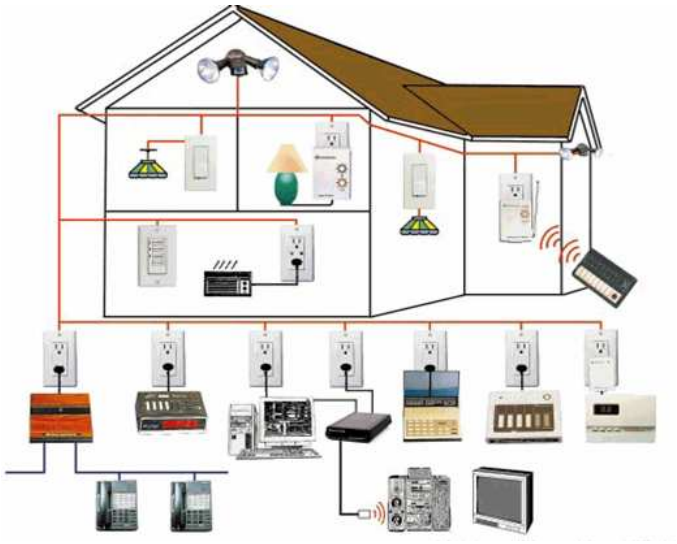


Figura 2.9. Ilustración vivienda domotizada [3].

El sistema permite el accionamiento a distancia y control remoto de diversos receptores eléctricos, desde uno o desde varios puntos.

Para poder referirnos a un determinado dispositivo, el protocolo define un sistema de direccionamiento basado en un código de hogar (16 códigos que van de la A a la P) más un código de dispositivo (16 códigos numerados del 1 al 16). El código de hogar es único para la misma vivienda, aunque se pueden usar varios códigos de hogar en caso de necesitar más de 16 direcciones por vivienda. Este protocolo permite asignar la misma dirección a más de un dispositivo, por ejemplo para encender varias luces a la vez con un solo comando.

Un problema típico del protocolo X10 es el de recibir señales de otro vecino si el cableado de ambos no está suficientemente aislado. Para solucionarlo se acuerda el uso de distintos códigos de hogar o se instala un capacitador que no deje propagar las señales a otra vivienda.

Un paquete o trama de datos en el protocolo X10 está formado por:

Código de inicio de trama. Es un código de 4 bits fijos (1110) que sirven para indicar el comienzo de cada trama de datos.

Código del hogar. Son 4 bits que identifican la vivienda.

Código de función. Éste puede ser un código de dispositivo (0 en el último bit) o bien un comando (indicado con un 1 en el último bit).

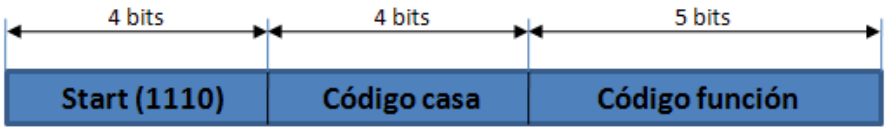


Figura 2.10. Trama de datos del protocol X10.

Cuando se desea enviar una trama a un dispositivo, se envía primero una trama que lo identifique y luego el comando correspondiente.

El protocolo require enviar cada trama dos veces seguidas, para evitar errores. Además requiere enviar una secuencia de 6 ceros seguidos “000000” cuando los datos cambien de una dirección a otra, de una dirección a un comando o de comando a comando.

A continuación se muestra la tabla de los códigos de hogar, dispositivo y funciones permitidas.

HOUSE CODES					KEY CODES					
	H1	H2	H4	H8		D1	D2	D4	D8	D16
A	0	1	1	0	1	0	1	1	0	0
B	1	1	1	0	2	1	1	1	0	0
C	0	0	1	0	3	0	0	1	0	0
D	1	0	1	0	4	1	0	1	0	0
E	0	0	0	1	5	0	0	0	1	0
F	1	0	0	1	6	1	0	0	1	0
G	0	1	0	1	7	0	1	0	1	0
H	1	1	0	1	8	1	1	0	1	0
I	0	1	1	1	9	0	1	1	1	0
J	1	1	1	1	10	1	1	1	1	0
K	0	0	1	1	11	0	0	1	1	0
L	1	0	1	1	12	1	0	1	1	0
M	0	0	0	0	13	0	0	0	0	0
N	1	0	0	0	14	1	0	0	0	0
O	0	1	0	0	15	0	1	0	0	0
P	1	1	0	0	16	1	1	0	0	0
					All Units Off	0	0	0	0	1
					All Lights On	0	0	0	1	1
					On	0	0	1	0	1
					Off	0	0	1	1	1
					Dim	0	1	0	0	1
					Bright	0	1	0	1	1
					All Lights Off	0	1	1	0	1
					Extended Code	0	1	1	1	1
					Hail Request	1	0	0	0	1
					Hail Acknowledge	1	0	0	1	1
					Pre-Set Dim	1	0	1	X	1
					Extended Data (analog)	1	1	0	0	1
					Status=on	1	1	0	1	1
					Status=off	1	1	1	0	1
					Status Request	1	1	1	1	1

Figura 2.11. Tabla de códigos de hogar, dispositivo y función.

En cuanto al aspecto **físico** se refiere, las transmisiones en X10 están sincronizadas con el paso por cero de la tensión de red, esta característica es común a todos los dispositivos X10 y tiene una doble finalidad: la primera es sincronizar a los transmisores y receptores, ya que la única conexión física que existe entre ellos es la línea de red, la segunda es debida a que el nivel mínimo de interferencias producidas por otros equipos eléctricos se produce cuando la señal de red pasa por cero. Los dispositivos X10 no distinguen entre el paso por cero cuando la señal va de positivo a negativo o de negativo a positivo; ambos pasos por cero son interpretados de igual modo por el dispositivo.

Un “1” binario del mensaje se representa por un pulso de 120 Khz durante 1 ms, en el paso por cero de la señal de red, y el “0” binario del mensaje se representa por la ausencia de ese pulso de 120 Khz.

El principio de codificación X10 permite una activación y respuesta definidas de hasta 256 receptores, puestos de control de aparatos o de grupos de consumidores. Con ello resulta posible el montaje de amplias redes.

Cuando se transmite el código de la Figura 11, se utilizan **dos pasos por cero para transmitir cada bit como una pareja de bits complementarios** (un **cero se representa por 0-1** y un **uno es representado por 1-0** según se muestra en la siguiente figura).

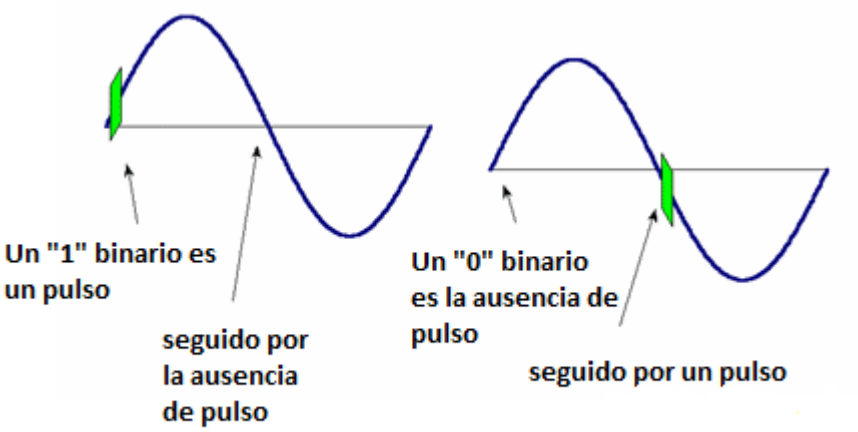


Figura 2.12. Transmisión de cada bit como una pareja de bits complementarios (Paso por uno y por cero).

Haciendo cálculos, el protocolo da una tasa de transferencia aproximada de 20 bits/s, hoy en día excesivamente baja, aunque perfectamente válida para señales sencillas como son las de X10. Es por ello que al dar una orden se nota un cierto retardo en observar los resultados.

2.3. *WiiMote.*

2.3.1. Introducción al *WiiMote.*

La empresa desarrolladora de consolas y videojuegos *Nintendo,* lanzó la consola de videojuegos *Wii* en noviembre de 2006, esta consola tuvo una gran acogida en los hogares de todo el mundo debido en su mayor parte a la novedad de que ahora los jugadores interactúan con los juegos realizando movimientos, lo cual se debe al mando de la misma conocido como ***WiiRemote (WiiMote)***. Este mando es un controlador inalámbrico que permite a los jugadores realizar movimientos naturales, los cuales se usan como entrada de datos, gracias a sus sensores de movimiento y su sistema de posicionamiento espacial, lo cual mejora la interacción con el juego haciéndolo más dinámico, natural y divertido.

A pesar de su gran éxito existen pocos desarrollos para PC que aprovechan las grandes posibilidades que ofrece, es por eso que en el presente proyecto se va a crear una nueva aplicación hasta ahora poco desarrolladas aprovechando todo lo que nos ofrece el dispositivo.

Como ya se comentó en la introducción del proyecto, las grandes ventajas del *WiiMote* son su sencillez y su bajo precio teniendo en cuenta que une varias tecnologías en un mismo dispositivo, cosa que no se ha logrado con otros dispositivos. Además debido a su carácter autónomo no es necesario adquirir una consola *Wii* para su utilización y desarrollo de aplicaciones para el PC.

2.3.2. Controlador *WiiMote.*

Como ya hemos mencionado este dispositivo será la base del proyecto y destaca sobre todo por la capacidad de detección de movimiento en el espacio y la habilidad de apuntar hacia objetos en la pantalla, además de la pulsación de botones. Aunque en este proyecto se tratará en detalle la detección de movimiento y la pulsación de botones y se dejará la capacidad de apuntar objetos en la pantalla como objetivo de los estudios futuros.

Figura 2.13. Controlador Wiimote.

El controlador ***WiiRemote (WiiMote)*** está formado por los siguientes elementos:

- Botones (POWER, A, -, HOME, +, 1, 2, B como gatillo inferior).
- Cruz direccional.
- 4 leds: indican el número de jugador o el nivel de batería.
- 2 pilas AA: con una duración de entre 30 y 60 horas según el uso o no de la función de puntero.
- Acelerómetro ADXL330 de 3 ejes: para la detección del movimiento del usuario.
- Sensor óptico de infrarrojos: para detección de movimientos del usuario.
- Puerto de expansión: para la conexión de otros elementos como el controlador *Nunchuk*.
- Conexión bluetooth: para la comunicación entre la consola y el propio *WiiMote.*
- Altavoz.
- Motor de vibración.
- Chip de memoria EEPROM de 16 KB: destinado a guardar información del juego en ejecución y hasta 10 perfiles de usuario (denominados *Mii*).

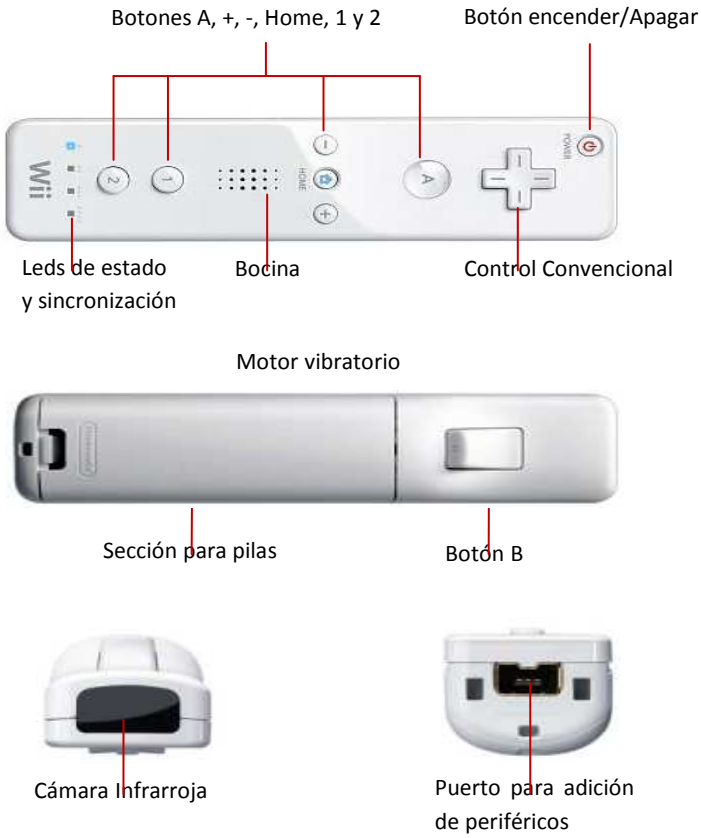


Figura 2.14. Elementos del controlador *WiiMote*.

2.3.3. Funcionamiento.

Como ya se ha mencionado, el control del ***WiiMote*** se basa tanto en la pulsación de botones como en el movimiento del usuario para el control tanto de las aplicaciones como los juegos de la videoconsola *Wii.* Dichos movimientos se detectan utilizando dos elementos del controlador:

- **Acelerómetros de 3 ejes**. El cual mide la inclinación del mando en 3 ejes y se transmite a la consola a través del enlace bluetooth. Este acelerómetro mide las fuerzas de gravedad en los tres ejes principales.

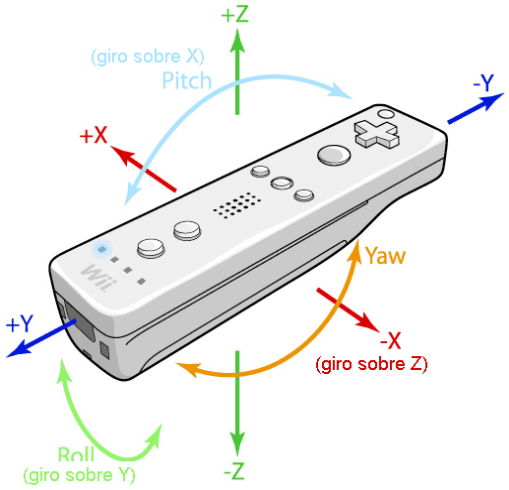


Figura 2.15. Acelerómetros *WiiMote*.

- **Sensor de infrarrojos**. Este elemento es similar al de una cámara de visión infrarroja y se utiliza para indicar a dónde apunta el usuario con el mando, por lo que el mando se usa como puntero. Para que esto funcione se requiere una fuente de luz infrarroja que pueda detectar el sensor para transmitir a la consola su posición. Para ello la videoconsola *Wii* suministra una barra con leds infrarrojos llamada "*Wii Sensor Bar*" que se consola encima o debajo de la pantalla de TV conectada a la consola. En su interior existen 10 leds infrarrojos, 5 en cada extremo.

Figura 2.16. Barra infrarroja.

Los movimientos del mando son los siguientes:

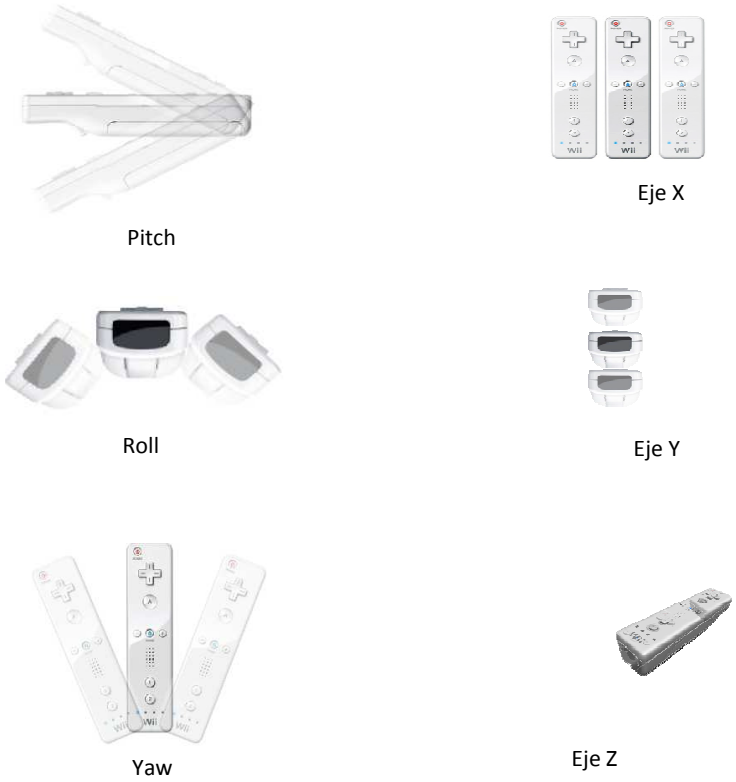


Figura 2.17. Movimientos *WiiMote*.

2.3.4. Conectividad. Bluetooth.

La tecnología inalámbrica *bluetooth* fue diseñada con el fin de eliminar el uso de cables, principalmente en cuanto a dispositivos se refiere. *Bluetooth* es una tecnología de corto alcance, de bajo consumo y de bajo coste. El uso de *bluetooth* se ha extendido a una gran cantidad de dispositivos con diferentes funcionalidades, desde el manos libres para el envío de datos de voz, como el ratón o el teclado inalámbrico, impresoras, PDA's, portátiles y lo que a nosotros más nos interesa ***WiiMote***.

La tecnología *bluetooth* opera en una banda de frecuencia industrial, científica y médica (ISM) que no requiere licencia y que se encuadra, concretamente, entre 2.4 y 2.485 GHz. Utiliza una señal bidireccional en un espectro ensanchado por salto de frecuencia a una velocidad nominal de 1600 saltos/segundo. La banda ISM de 2.4 GHz está disponible en casi todos los países y no suele requerir licencia.

Una de las características principales de esta tecnología es el salto adaptable de frecuencia, lo que significa que el dispositivo al momento de enviar datos verifica qué otros dispositivos están conectados y qué frecuencia están utilizando, posteriormente ajusta una nueva frecuencia para transmitir y recibir datos. El proceso mencionado anteriormente se usa con el fin de evitar interferencias con las transmisiones de otros

dispositivos que comparten el mismo espectro y así garantizar una transferencia limpia y segura. La señal salta entre 79 frecuencias en intervalos diferentes de 1 MHz para tener un alto grado de tolerancia a las interferencias.

Existen tres clases de dispositivos de tecnología Bluetooth:

- **Clase 3:** Su radio varía entre uno y tres metros.
- **Clase 2:** Su radio es promedio de 10 metros y es usado principalmente por portátiles (este también es el caso del ***WiiMote***).
- **Clase 1:** Su radio es promedio de 100 metros y es usado en entornos industriales. Las velocidades que se pueden alcanzar son de 1 Mbps para la versión 1.2 y hasta 3 Mbps para la versión 2.0.

Por tanto y como ya hemos mencionado, para la comunicación entre el ***WiiMote*** y la videoconsola se emplea *bluetooth* y emplearemos dicha tecnología para la comunicación del ***WiiMote*** con el PC o con cualquier otro dispositivo. En el **anexo 1** se explica cómo llevar a cabo esta conexión.

Gracias a la posibilidad de conectar el ***WiiMote*** con el PC a través de la tecnología *bluetooth* han surgido numerosas librerías y aplicaciones para su utilización como dispositivo de entrada sustituyendo un ratón, teclado o joystick. Se considera como objetivo del proyecto el estudio de las diferentes librerías y aplicaciones más importantes así como el desarrollo de nuevas aplicaciones basadas en estas librerías.

2.3.5. Accesorios Hardware *WiiMote.*

Se va a colocar aquí una tabla de los diferentes accesorios que se pueden incorporar al ***WiiMote***.

Nunchuck		**-Este accesorio se conecta al puerto para adición de periféricos del *WiiMote.*** **-Dispone de acelerómetros.**
Classic Controller		**-Mando tradicional, no cuenta con acelerómetros (detección de movimiento).** **- Consta de dos sticks analógicos y diferentes botones.**

Wii Classic Controller Pro		**-Igual que Classic Controller a excepción de los botones traseros → gatillos verticales.** **-Añade agarres por debajo del regulador.**
Guitarra.		**-Alberga en su interior al *WiiMote*, se comunica con él a través del puerto para adición de periféricos.** **-Dota a la guitarra de sensor y movimiento y vibración**
Wii Balance Board		**-Dotada de cuatro sensores de presión en su interior.** **-Pensada para el control de software a través de los movimientos de nuestro cuerpo.** **-Dotada de conexión bluetooth.**
Wii Zapper		**-Combina precisión del *WiiMote* y Nunchuk en una carcasa.** **-Juegos de acción y puntería.**
Wii Wheel		**-Carcasa de plástico con forma de volante donde se añade el *WiiMote*.**
Wii Motion Plus		**-Accesorio que incorpora tres sensores de movimiento correspondientes a los ejes vertical, longitudinal y lateral**
Wii Vitality Sensor		**-Capaz de leer nuestro flujo sanguíneo con ayuda de unos sensores** **-Proporciona al equipo datos sobre el funcionamiento del cuerpo en su interior.**

Wii Remote Plus.		**-Accesorio incorporado al propio *WiiMote*.** **-Combina WiiRemote y Wii Motion Plus en un solo control.**
UDraw Game Tablet		**-Consiste en una tabla de juego con un hueco para colocar el *WiiMote*.**

Figura 2.18. Tabla de accesorio *WiiMote*.

2.3.6. Aplicaciones para el manejo del *WiiMote* desde el PC.

Una de las partes desarrolladas en este proyecto corresponde como ya se ha mencionado a establecer la conexión del ***WiiMote*** con el PC, así como usar algunas de las funcionalidades que ofrece (botones y movimientos) para el manejo de nuestros dispositivos domóticos.

Sin embargo existen aplicaciones ya desarrolladas para tal fin, algunas de las cuales se van a explicar a continuación.

2.3.6.1. *Glovepie.*

GlovePIE (Glove Programmable Input Emulator) fue inicialmente creado para la emulación de teclados, ratones y joysticks utilizando guantes de realidad virtual (o VR) modelo P5. Se trata de un intérprete de *scripts* (con extensión .PIE) con un lenguaje específico parecido a *Basic*, para la asignación de acciones del guante a pulsaciones del teclado o movimientos de un suspuesto joystick o ratón. Actualmente este programa no sólo sirve para la emulación de guantes VR P5 si no que es compatible con una gran cantidad de dispositivos entre los que se incluyen otros modelos de guantes VR, micrófonos, joysticks o gamepads, dispositivos de juego que funcionen por puerto paralelo, máquinas de remo, dispositivos MIDI, y lo que es más interesante, compatibilidad total con ***WiiMote***.

Este intérprete tiene dos cualidades que le han hecho cultivar un gran éxito en el mundo del ***WiiMote*** en PC. Por una parte realizar *scripts* para él es muy sencillo a la vez que completo. Cualquier persona que haya programado en cualquier lenguaje no tardará en adaptarse, e incluso los no programadores encuentran bastante fácil su utilización.

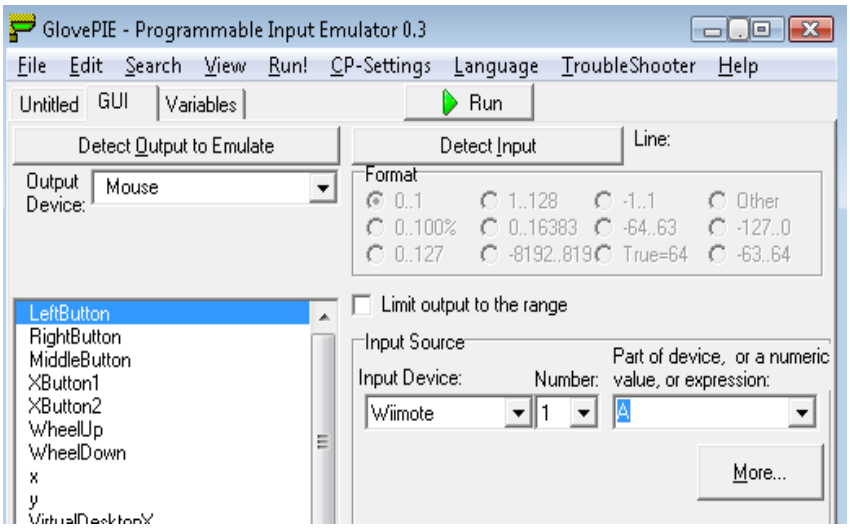


Figura 2.19. Ventana ejemplo de GlovePie.

Los comentarios se ponen con // delante, así no se interpreta como una acción.

La mayor parte de las líneas son simples asignaciones del tipo:

Mouse.RightButton = Wiimote.A //Botón A del Wiimote equivale a click derecho
Mouse.LeftButton = Wiimote.B //Botón B del Wiimote equivale a click izquierdo.

Aunque también permite la creación de estructuras de control básicas como la clásica IF/ELSE y el uso de variables.

La otra ventaja es que no requiere realizar modificaciones en las aplicaciones o juegos que se deseen utilizar, ya que simplemente *mapea* acciones del ***WiiMote*** a acciones de teclado, ratón o joystick, por lo tanto es compatible con prácticamente todos los programas y juegos del mercado.

Por último, el éxito que ha cosechado ha hecho que existan cientos de *scripts* en Internet disponibles para su descarga y utilización gratuita, por lo que utilizar el ***WiiMote*** como dispositivo de juego para PC es muy sencillo.

A continuación se detallan varios Scripts de ejemplos utilizando Glovepie.

// Script para un juego de carreras

//Controles de dirección

Left = Wiimote.Up //Control para girar a la izquierda.
Right = Wiimote.Down //Control para girar a la derecha.

//Controles de aceleración

Up = Wiimote.One //Control de Aceleración.
Down = Wiimote.Two //Control de Frenada.

Con este *script* conseguiremos que, girando el ***WiiMote*** dejando la cruceta a la izquierda, al apretar el botón 1 aceleraremos y con el botón 2 frenaremos, y pulsando arriba o abajo en el ***WiiMote*** (que girando sería izquierda y derecha) giraremos el coche.

//Script para controlar el reproductor Media Player Classic

//Controles de archivo

Ctrl+o = Wiimote.One //Abrir Archivo.
Ctrl+l = Wiimote.Two //Cargar Subtítulos.
//Controles de reproducción

Space = Wiimote.A //Play-Pause.
Period = Wiimote.B //Stop.
Ctrl+right = Wiimote.Right //Avanzar 5 segundos.
Ctrl+Left = Wiimote.LEft //Retroceder 5 segundos.
Add = Wiimote.up //Retraso de audio +10 ms.
Subtract = Wiimote.Down //Retraso de audio -10 ms.

//Controles del reproductor

Ctrl+Enter = Wiimote.Home //Pantalla Completa.
Atl+X = Wiimote.B + Wiimote.A //Salir del Media Player.
Up = Wiimote.Plus //Subir Volumen.
Down = Wiimote.Minus //Bajar Volumen.

Este *script* hace uso de todos los botones del ***WiiMote*** e incluso de una función usando 2 botones a la vez.

2.3.6.2. *Win Remote PC.*

Win Remote PC Control es un software gratuito para plataformas Windows (para uso no comercial) utilizado para administrar y manejar un PC de forma remota, es decir para poder controlar una computadora como si estuvieses en tu propia máquina, ya sea desde internet o una red local.

Como la mayoría de los programas de este estilo, el software incluye dos programas: un cliente y un servidor. El primero es el que permite acceder a la máquina que se quiere controlar o administrar, y el segundo es el programa que debe estar corriendo o instalado en la misma para que el cliente pueda acceder.

Entre las características principales de Win Remote PC Control podemos:

- Enviar combinaciones de teclas al PC, por ejemplo las teclas Ctrl+Alt+Supr para poder acceder rápidamente al administrador de tareas del sistema por ejemplo.
- Se pueden administrar múltiples escritorios mediante pestañas, para que sea más cómodo acceder a las distintas conexiones establecidas.
- Utilizar para el acceso, la seguridad integrada de Windows o la del propio programa.

- Tomar el control del mouse, del teclado y del escritorio.

- Incluye su propio controlador de vídeo para una rápida visualización que proporciona el control en tiempo real con mayor rendimiento y velocidad.

- Ejecutar a pantalla completa.

- Envío de datos en forma encriptada utilizando los algoritmos BlowFish, TEA y otros.

- Poder indicar el modo de color entre los siguientes valores: 16 y 24 Bits para óptima visualización, de 4, 8 y 12 bits en color, y también en blanco y negro.

- Estadísticas de las conexiones: ver las activas, las velocidades, los bytes enviados y recibidos y otros datos.

- Realizar transferencias de archivos entre las máquinas conectadas.

- Arquitectura de plugins, para poder extender las funciones y opciones del programa.

- Soporte para clipboard.

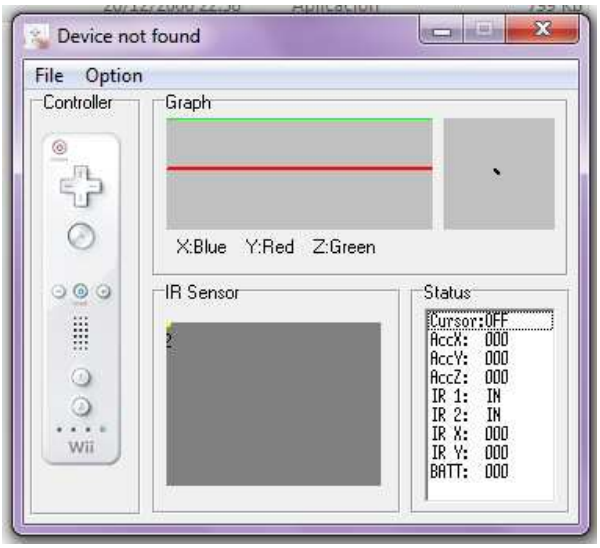


Figura 2.20. Win Remote PC.

2.3.6.3. *Darwiin Remote.*

La aplicación Darwiin Remote nos permite controlar el ordenador de Apple Macintosh (MAC) con el ***WiiMote***. Posee dos modos de control:

- **Modo básico**: nos permite enlazar el mando con el MAC.

- **Modo avanzado**: nos permite monitorizar el acelerómetro del mando y controlar el puntero del MAC con el mismo.

Con Darwiin Remote podemos usar el ***WiiMote*** para controlar FrontRow en su totalidad.

FrontRow es una aplicación para los ordenadores de Apple Macintosh, que actúa como máscara para QuickTime, DVD Player y para las librerías de iTunes e iPhoto que permite a los usuarios navegar por los contenidos multimedia del ordenador.

Figura 2.21. Darwin Remote.

CAPÍTULO 3. Descripción del software utilizado.

Una vez presentado el marco teórico del proyecto dentro del cual también se describe el hardware utilizado en el mismo, vamos a describir más en profundidad el software utilizado, mencionado en el primer capítulo (presentación del proyecto).

3.1. Eclipse y java.

Como ya hemos mencionado anteriormente empleamos como entorno de desarrollo ***Eclipse,*** por su sencillez de utilización e instalación y por las facilidades que ofrece a la hora de programar. Y como lenguaje de programación ***java*** debido al previo conocimiento que se tenía de este lenguaje y a la innovación que supondría puesto que no existen demasiadas aplicaciones sobre ***WiiHome*** desarrolladas en este lenguaje.

3.1.1. Eclipse.

Eclipse es una plataforma de desarrollo software genérica desarrollada por una comunidad "open source" que permite crear entornos de desarrollo integrados para distintos lenguajes programación.

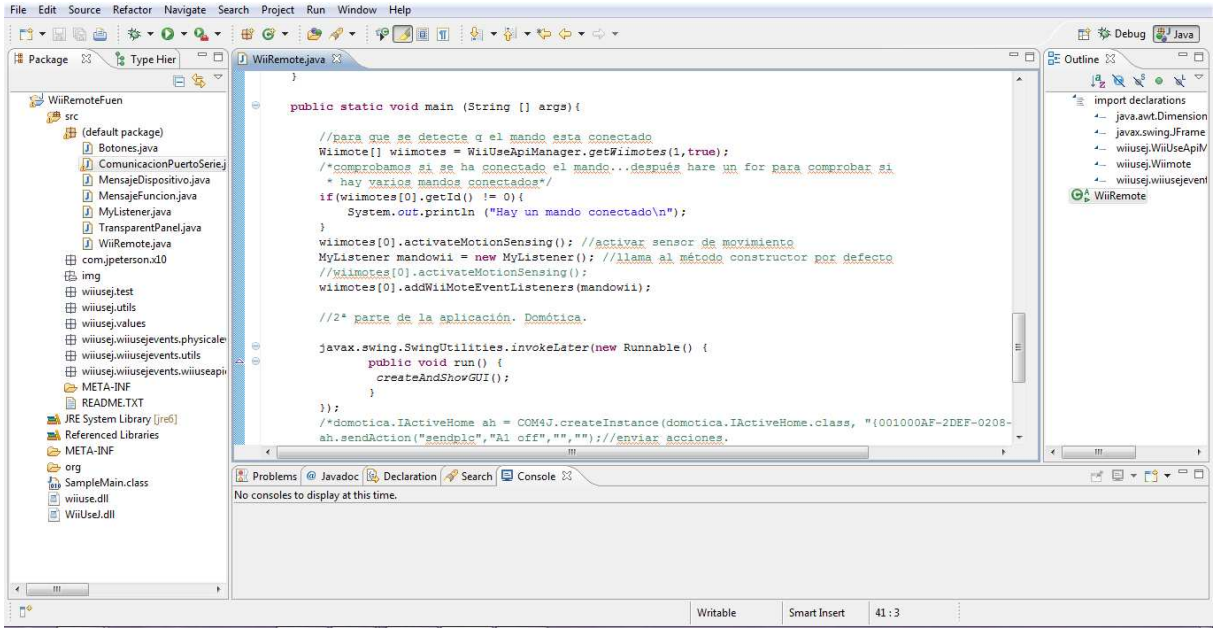


Figura 3.1. Entorno de programación Eclipse.

Aquí se muestra una imagen del entorno de programación. Como se puede observar, en la parte central es dónde aparecen las clases e interfaces creadas con sus respectivos códigos. En la parte de la izquierda de la imagen es donde nos aparece un esquema del proyecto creado incluidas las librerías importadas necesarias a lo largo del desarrollo del proyecto. Y en la parte inferior del entorno nos aparecen los errores que se producen tanto en compilación como durante la ejecución del código creado, además de los resultados obtenidos en un proceso de búsqueda, entre otras cosas.

3.1.2. Java.

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. Con respecto a la memoria, su gestión no es un problema ya que ésta es gestionada por el propio lenguaje y no por el programador.

Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.

Como he comentado anteriormente he considerado que este es el lenguaje más idóneo para el desarrollo de este proyecto por las razones previamente explicadas.

3.2. Librería para el manejo del *WiiMote* → API Wiiuse J.

3.2.1. Descripción de la API.

WiiUseJ es una **API Java** para el acceso al periférico ***WiiMote*** de la consola **Wii** de **Nintendo®**. Se basa en una API C llamada WiiUse que accede directamente al *bluetooth* stack de nuestro sistema (como BlueSoleil, Widcomm o Bluez). La diferencia de **WiiUseJ** con otras APIs Java reside en que no se basa en el paquete javax.bluetooth (implementación del estándar JSR-82), con lo cual resulta ser un mecanismo más eficiente para acceder al mando al estar implementada sobre un lenguaje de bajo nivel como es C.

A continuación se va a detallar de manera general cómo es la API y cómo se hace uso de la misma en el presente proyecto pero será en el **anexo 4** dónde se detallan cada una de las clases e interfaces de las que está formada esta API.

La librería WiiUse en C está disponible para plataformas Windows y Linux (formatos .dll y .so).

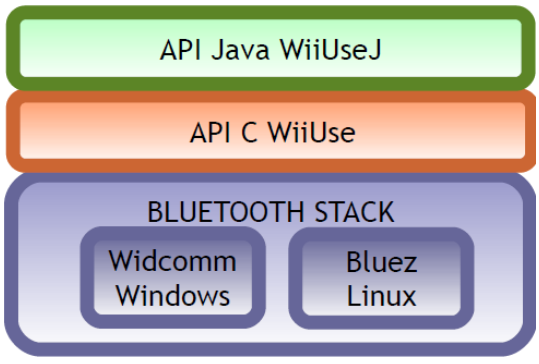


Figura 3.2. Jerarquía de la API WiiUseJ.

Para el correcto funcionamiento de la API esta librería se debe colocar en la carpeta principal de nuestro proyecto.

3.2.2. Funcionamiento WiiUseJ.

Una de las funcionalidades más importantes que ofrece el mando es la recepción de fuentes de luz infrarroja. El funcionamiento de **WiiUseJ** en este aspecto se detalla a continuación.

A un nivel bajo, podemos considerar las clases **IRSource** e **IREvent** (fuente infrarroja y evento infrarrojo, respectivamente).

- **IRSource** representa una luz infrarroja detectada por el mando, vista como un punto bidimensional con un tamaño asociado.

- **IREvent** representa un evento infrarrojo formado por todos los **IRSources** (puntos de luz infrarroja) que estén dentro del radio de acción del ***WiiMote***. Estos puntos se almacenan en un array. En los objetos de la clase **IREvent** se almacena también la posición absoluta del ***WiiMote*** (coordenadas x, y, z) y otra información relevante, como por ejemplo, si el mando se encuentra por encima o por debajo de los puntos de luz.

Figura 3.3. Representación gráfica de IREvent y IRSource.

La librería **WiiUse C** recoge los eventos **IREvent** a bajo nivel y los comunica a la **API Java**. En esta última, la librería **WiiUse** se representa mediante la **clase WiiUseApi**, un patrón de diseño usado para llamar directamente a los archivos .dll o .so desde Java.

La **clase** que maneja a la librería **WiiUse** se llama **WiiUseApiManager**, que hereda de la clase **Thread** y se encarga de analizar quién ha mandado los eventos (técnica de **polling**), en su método run().

En la **API WiiUseJ** existen dos tipos de **oyentes** o **"listeners"**, representados por las dos interfaces siguientes:

- **WiiUseApiListener:** se encarga de escuchar los eventos que recoge **WiiUseApiManager**. La clase que implemente esta interfaz se encargará de llamar a los métodos adecuados dependiendo del tipo de evento escuchado. Si detecta que el evento escuchado es infrarrojo, **llamará** al método **onIrEvent(IREvent arg)** definido en la siguiente interfaz.

• **WiimoteListener:** la clase principal del proyecto tiene que implementar esta **interfaz**, que es la que define las acciones a llevar a cabo en la aplicación concreta cuando se producen los eventos recogidos por **WiiUseApiListener**. **Define** el método **onIrEvent(IREvent arg)**, el cual accede a la información del evento infrarrojo.

Todo esto queda reflejado en el siguiente diagrama:

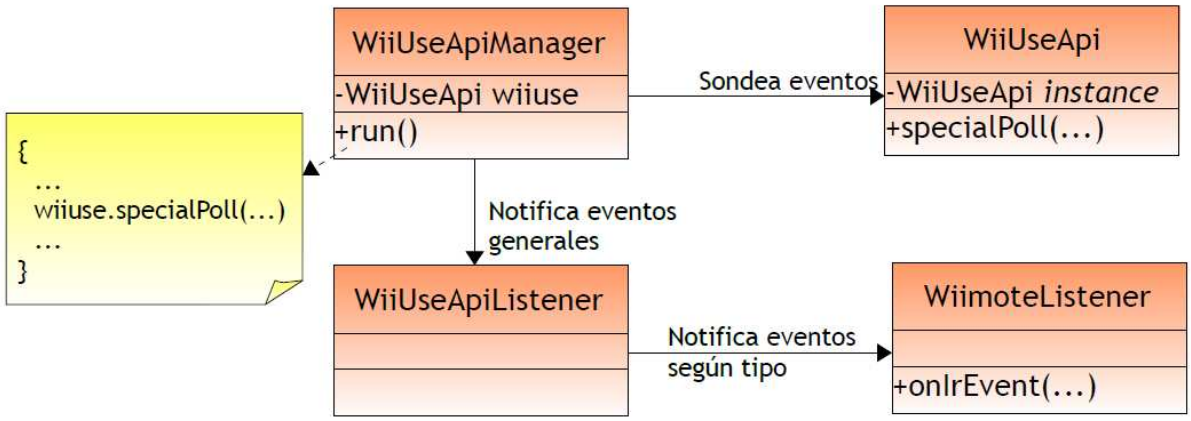


Figura 3.4. Diagrama UML de la librería WiiUseJ.

A parte de esta funcionalidad de control de eventos infrarrojos, esta librería también ofrece otras funciones como:

- Soporta múltiples conexiones con ***WiiMotes***.
- Soporta la conexión de diferentes extensiones del mando.
- Como ya se ha mencionado, funcionamiento por **polling** o **eventos.**
- Con lo que hemos explicado anteriormente podremos obtener soporte para el acelerómetro (en diferentes formas: orientación y GForce) y para la cámara de infrarrojos.
- Lectura de la pulsación de botones.
- Lectura del estado de la batería.
- Encendido/apagado de los LEDs.

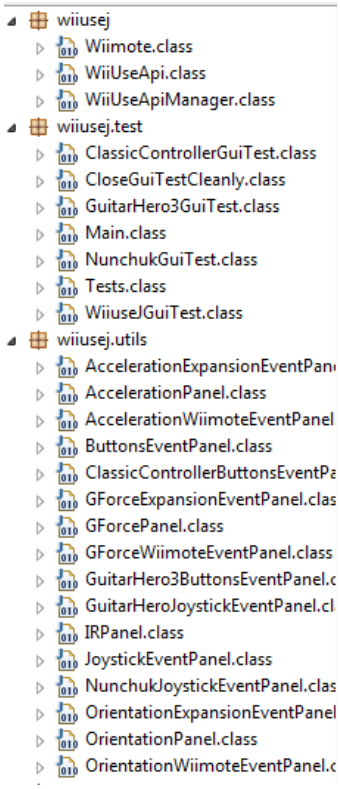


Figura 3.5. Clases de la librería WiiUseJ en el entorno de programación Eclipse.

3.3. Otras librerías para el *WiiMote*.

Además de la librería usada en el proyecto existen otras muchas para distintas plataformas y lenguajes de programación:

- *Motej* (java).
- *WiiRemotej* (java).
- *WiimoteLib* (C# y Visual Basic).
- *WiiYourself!* (C++, Windows).
- *Wiim* (C++,Windows).
- *Wiiuse* (C, Windows/Linux).
- *CWiid* (C, Linux).

3.3.1. *Motej* (java).

Es una librería Java para la comunicación con ***Wiimote***, bajo licencia ASL 2.0. Está basada en dos paquetes: librería y extras. La librería ofrece acceso básico al ***Wiimote***. Los extras por su parte extienden la funcionalidad básica ofrecida por la librería añadiendo funciones más complejas para facilitar el trabajo al programador.

La arquitectura de esta librería puede comprenderse mediante el siguiente esquema. Como se puede observar, la librería depende de una implementación del estándar de Java *JSR82*, por lo que se requiere una API adicional para el acceso al dispositivo. El módulo de extras accede a las funciones básicas de la librería para extender la funcionalidad de la misma.

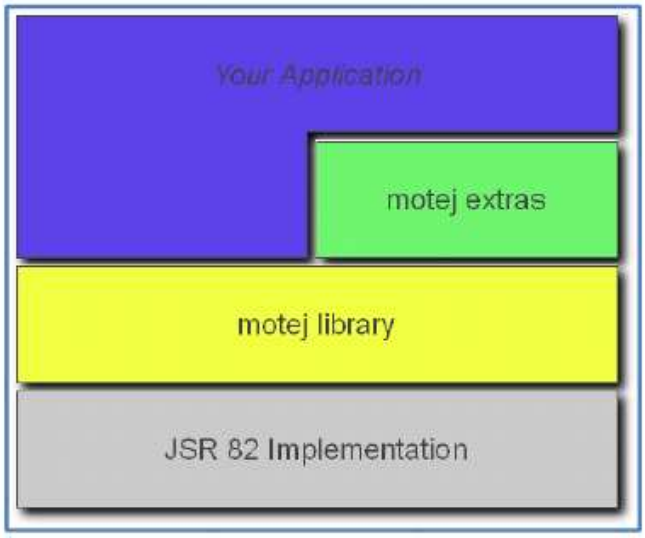


Figura 3.6. Jerarquía que rodea la librería Motej.

Motej ofrece las siguientes funcionalidades, tanto para comprobar el estado del ***WiiMote*** como para ejecutar acciones sobre el mismo:

- Descubrimiento y conexión a dispositivos.
- Cámara de infrarrojos.
- Acelerómetro.
- Botones.
- Activación/desactivación de la función de vibración.
- Encendido/apagado de LEDs.
- Acceso a la memoria EEPROM del dispositivo.
- Acceso a los registros del ***WiiMote***.
- Acceso al estado del ***WiiMote.***
- Acceso a los datos de calibración.

3.3.2. *WiiRemotej* (java).

Esta librería es muy similar a *motej*, también requiere una implementación del estándar JSR82 y las funcionalidades que ofrece son las mismas que las de la librería antes descrita, añadiendo algunas como soporte para extensiones del mando y del dispositivo *Wii Balance Board.*

3.3.3. *WiimoteLib* (C# y Visual Basic).

Esta librería fue creada por *Brian Peek y* es válida para su utilización tanto en lenguaje C# como Visual Basic y permite tanto leer los valores capturados por los acelerómetros, la cámara infrarroja de ***WiiMote*** y los botones como del dispositivo conectado al mismo a través del puerto de expansión. Se han añadido nuevas funcionalidades como la posibilidad de leer los valores capturados por otros accesorios como la *Wii Balance Board,* suministrada con el juego *Wii Fit*, que utilizada como tabla para la realización de distintos ejercicios físicos situando pies o manos sobre ella, es capaz de detectar la presión ejercida por el usuario. Además, permite enviar algunas órdenes al ***WiiMote*** como encender las distintas luces LED de las que dispone, y se está desarrollando la posibilidad de enviar sonidos para reproducirlos a través del altavoz del ***WiiMote***.

3.3.4. *WiiYourself!* (C++, Windows).

WiiYourself! es una librería basada en la creada por *Brian Peek*, pero orientada a C++ y extendida en funcionalidad. Además de las funciones que aporta la librería original, añade algunas novedades como:

- **Estimación de la orientación:** Mediante los métodos añadidos es posible detectar el ángulo del ***WiiMote*** de una manera más sencilla.
- **Mayor soporte de gestores Bluetooth:** Es capaz de utilizar cualquier programa gestor de conexiones Bluetooth.

3.3.5. *Wiim* (C++, Windows).

Esta librería también está desarrollada para ser usada en C++, aunque es mucho menos completa que la anterior.

Simplemente se define como un conjunto de clases C++ que permiten conectarse al ***WiiMote*** a través de la interfaz HID de Windows para enviar y recibir comandos.

3.3.6. *Wiiuse* (C, Windows/Linux).

Aunque no tiene tantas funciones como otras librerías, la ventaja de *wiiuse*, creada para funcionar con lenguaje C, es la posibilidad de utilizarla tanto para programas *Windows* como *Linux*, por lo que resulta más fácil portar programas o juegos de una plataforma a otra. Sus funciones son las siguientes:

- Soporte para acelerómetros.
- Infrarrojos.
- Extensiones (*nunchuck, classic controller*).
- Soporte para el controlador/guitarra de *Guitar Hero*.

3.3.7. *CWiid* (C, Linux).

CWiid, como cuyo nombre indica, está realizada en lenguaje C y orientada a sistemas *Linux*. Tiene una API muy reducida y más complicada de usar que otras. Soporta funciones muy básicas del ***WiiMote***.

3.4. Sistema de automatización para el hogar Active Home Pro.

La mayoría del software comercial usado para tratar con los dispositivos X10, tiene características muy limitadas, permitiendo únicamente la programación y activación de algunas funciones a una hora prefijada. En ningún caso existe la posibilidad de interacción entre elementos de la red.

Por otro lado, existen diversas aplicaciones de particulares y soluciones a medida, que normalmente adaptan el software existente a una necesidad concreta, pero que no pueden ser consideradas como sistemas completos o arquitecturas orientadas a dar soluciones globales. El sistema no necesita ningún software adicional para su administración, pero en el mercado existen programas que proporcionan la posibilidad de manejar y programar los dispositivos X10 desde el PC.

Estos programas necesitan de un módulo especial X10 que haga de intermediario entre el sistema y el computador, y desde este punto se pueden activar, desactivar y hacer temporizaciones y regulaciones con un simple movimiento del ratón del PC o en nuestro caso con el envío de eventos del mando ***WiiMote***.

Para su acceso remoto, con la aplicación de telnet adecuada o mediante un navegador web, se podría tener control de la vivienda a través con un acceso a la Internet.

También existe la posibilidad de administrar la vivienda a través de un teléfono fijo o móvil ya que hay en el mercado diferentes módulos de módem que facilitan este tipo de operaciones.

Entre los software más conocidos, completos y sencillos de uso, se encuentran los que pone a disposición de sus clientes, la empresa X10 Ltd., llamados **ActiveHome** y **ActiveHome Pro**. Ambos se distribuyen junto con interfaces X10 para permitir la comunicación del ordenador con la red X10, incluyendo el envío y monitorización de los comandos que atraviesan la red. En este proyecto se hace uso de la versión más desarrollada **Active Home Pro** el cual proporciona compatibilidad con **Windows 7** y aunque suele usarse con el controlador **CM15** también proporciona la posibilidad de ser usado con el controlador **CM11A**, que es por otro lado el usado en el desarrollo de este proyecto.

3.4.1. Funcionamiento general.

El software **Active Home Pro** es un sistema de automatización de fácil conexión, ya que no necesita ningún cableado especial simplemente las propias tomas de corrientes del hogar, además de tener un precio bastante asequible. Este software además facilitará el control de todo el sistema domótico instalado.

Como sabemos existen dos clases de equipos: controladores y módulos. Cualquier luz o aparato independiente que se quiere controlar se conectará a un módulo, que a su vez, se conectará a una toma de corriente de la vivienda o edificio.

Los módulos reciben comandos de los controladores. La interfaz proporcionada por **Active Home Pro** hace referencia a la información de cada módulo que está guardada en el controlador y por tanto queda reflejado en esta interfaz cada cambio que se produce en el módulo, siendo el controlador el que envía esa orden a los distintos módulos. Estos controladores, al igual que los módulos, también se conectan a una toma de corriente y envían órdenes a través del cableado eléctrico de la vivienda. El controlador es también un sistema receptor: cuando recibe un comando desde algún control remoto inalámbrico, ésta envía señales digitales hacia los módulos que reciben la señal y se ejecuta la orden.

El controlador se conecta al puerto USB del PC y el software **Active Home Pro** le dice a éste qué hacer para manejar los módulos. Una vez conectado el controlador al PC éste se convierte en la parte central y más importante del sistema creado.

La aplicación **Active Home Pro** permite realizar, por tanto, entre otras las siguientes acciones:

- Asignación de una dirección a cada dispositivo conectado, con la utilización de los códigos de hogar y dispositivo.
- Conectar el PC a la red domótica a través de un interfaz (controlador **CM11A**).
- Configurar un elemento por cada dispositivo.
- Programar en la aplicación los eventos y macros.

Este software sirve por tanto de interfaz de control sobre el hardware X10. Los dispositivos a ser controlados deberán estar conectados, como ya hemos mencionado a módulos X10 lo cual nos permite:

- Crear una representación gráfica de los módulos y controlar las luces y aparatos desde el ordenador.
- Crear calendarios de eventos que se ejecutan automáticamente.
- Definir calendarios de viajes que hacen que la casa parezca habitada cuando el usuario está fuera, mediante el encendido de luces y aparatos.
- Definir macros que controlan grupos de módulos, estas macros ayudarán a ejecutar una tarea específica definida a un grupo de modos, dependiendo la manera de configuración y funcionamiento de los equipos X10.
- Crear informes impresos que muestran los diferentes aspectos del sistema domótico como módulos instalados, tiempos de eventos definidos, etc.

Es, como vemos, un software no demasiado caro, de fácil manejo y que proporciona gran funcionalidad respecto a la automatización del hogar.

Para hacer uso de la aplicación que aquí se ha desarrollado es necesario conocer bien el funcionamiento de este software, el cual queda con más detalle explicado en el **anexo 5**.

CAPÍTULO 4. Descripción del software desarrollado.

Una vez que se conoce con detalle todos los dispositivos y todo el software usado, en este capítulo se va a proceder a explicar de manera detallada cómo se ha creado y cómo funciona el software desarrollado en el proyecto. Para ello este capítulo se va a dividir en tres subapartados, uno para cada una de las partes del proyecto puesto que el programa en sí está, como ya se ha comentado, dividido en dos partes (***WiiMote*** y domótica), y un tercero donde se explicará cómo se unen ambas partes. El hecho de dividir este apartado de esta manera, simplemente es para explicar de forma más sencilla cada una de las dos partes por separado, para poder entender más fácilmente de forma global la aplicación desarrollada, ya que finalmente ambos códigos se mezclarán[1].

Toda la aplicación está realizada en lenguaje JAVA empleando el patrón observador, cuya diagrama UML es el siguiente.

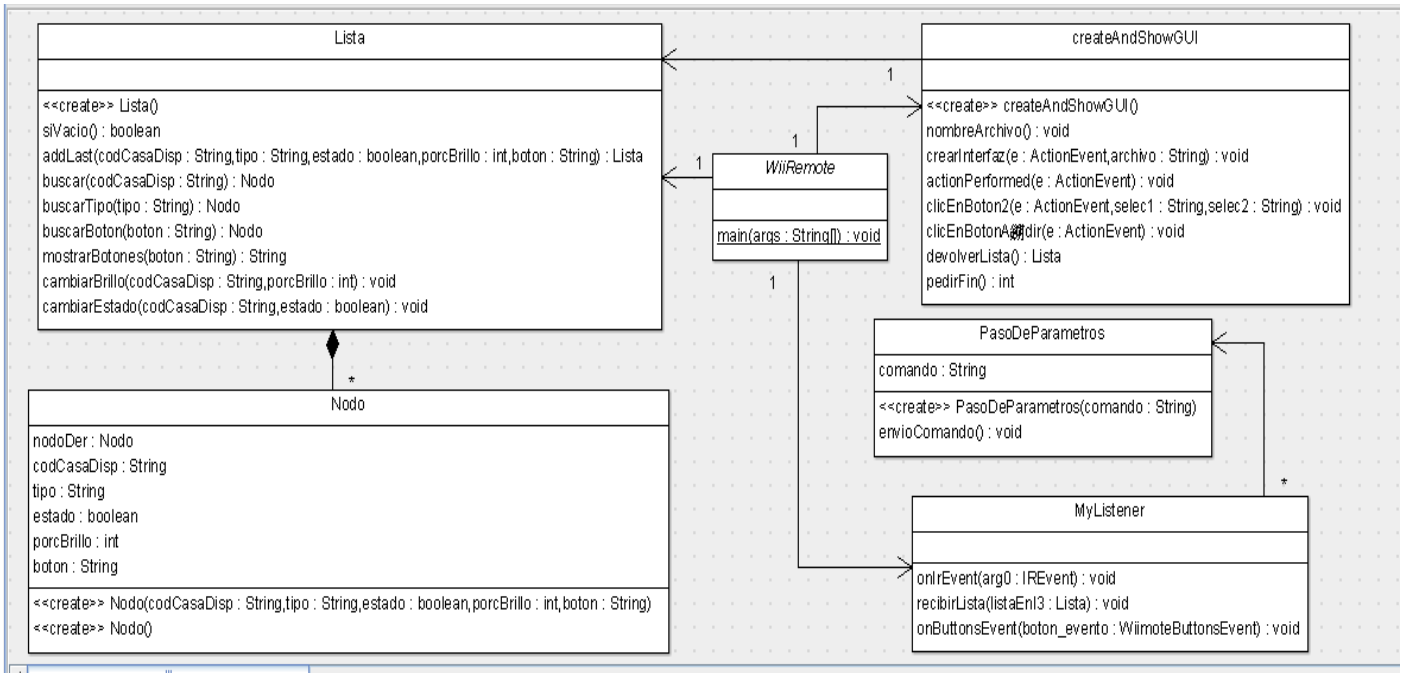


Figura 4.1. Diagrama UML del proyecto creado.

4.1. Desarrollo de software para *WiiMote.*

El objetivo principal del código JAVA creado para esta parte del proyecto es conseguir detectar la conexión del ***WiiMote*** al PC y poder así emplear la pulsación de los botones del mismo y sus movimientos como mecanismo para accionar y trabajar sobre los dispositivos domóticos aquí empleados.

Como ya hemos mencionado, se emplea para esta aplicación el patrón observador, por lo que el diagrama UML correspondiente a esta parte será el siguiente.

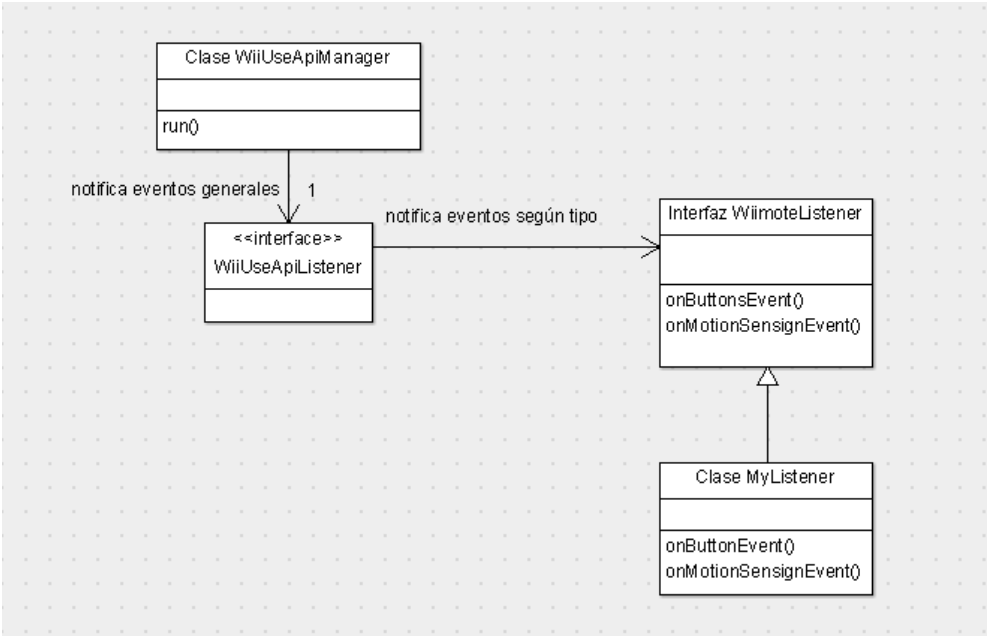


Figura 4.2. Diagrama UML de la parte referente al *WiiMote*.

Posteriormente (y esto es lo que trataremos en la siguiente parte), se transmitirá la información del evento lanzado por el ***WiiMote*** al dispositivo al que le corresponde dicho evento.

A continuación se explicará lo que se ha usado y los pasos seguidos para llevar a cabo lo anteriormente descrito.

La clase **WiiUseApiMAnager** se encarga de lanzar el "polling" de eventos, en su método run().

WiiUseApiListener se encarga de escuchar los eventos que recoge WiiUseApiManager. La clase que implemente esta interfaz se encargará de llamar a los métodos adecuados dependiendo del tipo de evento escuchado.

WiimoteListener. La clase encargada de escuchar los eventos del mando debe implementar esta interfaz, que es la que define las acciones a llevar a cabo en la aplicación concreta cuando se producen los eventos recogidos por WiiUseApiListener (la cual debe ser implementada por la clase principal).

WiiRemote. Es la clase principal del proyecto la cual implementa la interfaz WiiUseApiListener.

MyListener. Es la clase oyente del proyecto (escucha los eventos).

[1] En el **Anexo 2** se detallan los pasos a seguir para la creación del código desarrollado.

Esta aplicación, en primer lugar, intenta detectar que el mando ***WiiMote*** está activado y conectado al PC, para poder capturar sus movimientos y la pulsación de sus botones. Esta parte de la aplicación se muestra a continuación y es llevada a cabo en la clase principal del proyecto:

```
Wiimote[] wiimotes = WiiUseApiManager.getWiimotes(1,true);

if(wiimotes.length != 0){
      wiimotes[0].activateMotionSensing();
}

else{
      System.out.println("MANDO WIIMOTE NO DETECTADO");
      WiiUseApiManager.definitiveShutdown();
      System.exit(1);
}
```

Como vemos se declara un array en el que se almacenarán las conexiones con ***WiiMotes*** detectadas, por tanto para ver si se ha establecido alguna conexión se comprueba que la longitud de dicho array sea distinta de 0, en tal caso se hace vibrar el mando con el método *activateMotionSensing()* y será dentro de ese *if* (donde se detecta la conexión del mando) dónde se llevarán a cabo todas las acciones para la ejecución de la aplicación, que posteriormente se explicarán con más detalle. En caso de que no se detecte ninguna conexión, saltará un mensaje por pantalla, se desactiva la clase *WiiUseApiManager*, de manera que ya no espera recibir eventos y se cierra la aplicación.

Una vez que nuestra aplicación ha detectado el mando ***WiiMote*** y para poder detectar si se ha realizado un movimiento o se ha pulsado algún botón hacemos uso de dos métodos definidos en la interfaz WiimoteListener, los cuales capturan la acción realizada por el usuario. Para ello se crea un objeto de la clase oyente *MyListener* y se añade dicho objeto al array anteriormente creado, de la siguiente manera:

```
MyListener mandowii = new MyListener();
wiimotes[0].addWiiMoteEventListeners(mandowii);
```

Cuando se ha capturado la acción realizada por el usuario se llevarán a cabo las sentencias correspondientes para asociar dicha acción a la función que le corresponde en el sistema domótico creado, lo cual se explicará en el apartado siguiente. Esta captura de acciones se lleva a cabo en la clase oyente **MyListener**, y los métodos empleados para dicha función se muestran a continuación.

➔ Método **OnButtonsEvent**: detecta la pulsación de botones. En este caso he usado un método que detecta que se ha pulsado un botón determinado del mando y en caso de que esto sea así se pone la variable booleana asociada a dicho botón a true.

```
public void onButtonsEvent (WiimoteButtonsEvent boton_evento){

      /*BOTON A*/
      if(boton_evento.isButtonAJustPressed()){
            A = true;
      }
      /*BOTON B*/
      if(boton_evento.isButtonBJustPressed()){
            B = true;
      }
      /*BOTON ABAJO*/
      if(boton_evento.isButtonDownJustPressed()){
            Abajo = true;
      }
```

```
        /*BOTON ARRIBA*/
        if(boton_evento.isButtonUpJustPressed()){
              Arriba = true;
        }
        /*BOTON DERECHA*/
        if(boton_evento.isButtonRightJustPressed()){
              Dcha = true;
        }

        /*BOTON IZQUIERDA*/
        if(boton_evento.isButtonLeftJustPressed()){
              Izq = true;
        }
        /*BOTON MAS*/
        if(boton_evento.isButtonPlusJustPressed()){
              Mas = true;
        }
        /*BOTON MENOS*/
        if(boton_evento.isButtonMinusJustPressed()){
              Menos = true;
        }
        /*BOTON HOME*/
        if(boton_evento.isButtonHomeJustPressed()){
              Home = true;
        }
        /*BOTON UNO*/
        if(boton_evento.isButtonOneJustPressed()){
              Uno = true;
        }
        /*BOTON DOS*/
        if(boton_evento.isButtonTwoJustPressed()){
              Dos = true;
        }
    }
```

Como vemos se usa el método **isButtonXJustPressed()**, que detecta si se acaba de pulsar el botón A, sin embargo existen otros métodos, como por ejemplo **isButtonXHeld()** que detecta si el botón A se está manteniendo pulsado.

Después, en el apartado **4.3** de este mismo capítulo se explicará de forma detallada cómo se hace uso de las combinaciones de pulsación de botones para asociarlas a las acciones que se han querido llevar a cabo en el desarrollo de la aplicación.

➔ Método **onMotionSensingEvent**: detecta el movimiento del mando ***WiiMote***. Al igual que en el caso anterior se asocia una acción a una función domótica concreta.

```
    public void onMotionSensingEvent (MotionSensingEvent movimiento){

        Orientation orientacion = movimiento.getOrientation();

        mov_Z = (int)orientacion.getPitch();
        mov_Y = (int)orientacion.getRoll();

        if(mov_Z <= -55 && B==true){
            System.out.println("Estoy GIRANDO el mando hacia ARRIBA");
            try{Thread.sleep(2000);}
            catch (InterruptedException e){}
        }
```

```
        else if(mov_Z > 60 && B==true){
            System.out.println("Estoy GIRANDO el mando hacia ABAJO");
            try{Thread.sleep(2000);}
            catch (InterruptedException e){}
        }
        else if(mov_Y <= -50 && B==true){
            System.out.println("Estoy ENRROLLANDO hacia la IZQUIERDA");
            try{Thread.sleep(2000);}
            catch (InterruptedException e){}
        }
        else if(mov_Y > 50 && B==true){
            System.out.println("Estoy ENRROLLANDO hacia la DERECHA");
            try{Thread.sleep(2000);}
            catch (InterruptedException e){}
        }
    }
```

En este método se obtiene la orientación del mando en dos ejes (Z e Y) de manera que se puedan obtener cuatro movimientos del mando (arriba y abajo, derecha e izquierda) aunque en este código únicamente se hace uso de los movimientos en el eje Z (arriba y abajo).

Existen otros métodos que se declaran en la clase y que se explicarán en el **Anexo 6,** pero que no son usados para este proyecto.

4.2. Desarrollo del software en el ámbito domótico.

Como ya hemos dicho esta parte del proyecto también se desarrolla en el lenguaje JAVA. Esta parte del proyecto es algo más compleja puesto que aparte de que se necesitan crear varias clases JAVA, se emplea también el software **Active Home Pro** (cuya funcionalidad se ha explicado en el capítulo anterior), el cual nos proporciona una interfaz para poder visualizar los elementos físicos, así como su estado, entre otras cosas, además de proporcionarnos la herramienta para el envío de órdenes por línea de comandos al dispositivos en cuestión. Dicha herramienta es el ejecutable **ahcmd.exe**, el cual es usado para mandar las órdenes a la red domótica creada a través de la línea de comandos. Su funcionalidad y su sintaxis concreta se explican de forma más detallada en el **Anexo 3**.

En este subapartado se va a explicar con detalle el código JAVA que ha permitido la creación de la interfaz a través de la cual nos comunicaremos con el usuario, así como el código que permite el envío de órdenes a los dispositivos domóticos. Será en el **capítulo 5** donde se explique de manera global cómo el usuario debe arrancar y hacer uso de la aplicación creada, detallando el caso concreto de estudio en este proyecto.

En primer lugar se ha creado la clase **createAndDhowGUI** la cual crea una interfaz gráfica donde el usuario va introduciendo los datos pedidos. Esta clase implementa la interfaz **ActionListener**, por tanto es una clase manejadora de eventos, en concreto eventos de pulsación de botón, con lo cual casi todos sus métodos están enlazados a través de este tipo de eventos. A continuación se muestran todas las variables de la clase necesarias y el método constructor.

```
public class createAndShowGUI implements ActionListener{

	JFrame frame;
	JFrame frame2;
	JFrame frame3;
	JFrame frame4;
	JPanel panel;
	JPanel panel2;
	JPanel panel3;
	JPanel panel4;
	JPanel panel5;
	JPanel panel6;
	JPanel panel7;
	JPanel panel8;
	JPanel panel9;
	JPanel panel10;
	JPanel panel11;
	JLabel labelPpal;
	JLabel label2;
	JLabel label3;
	JLabel label4;
	JTextField campoTexto;
	JTextField campoTexto2;
	JTextField campoTexto3;
	JButton boton;
	JButton boton2;
	JButton boton3;
	JButton boton4;
	JButton boton5;
	JScrollPane scrollBar;
	Lista listaEnl = new Lista();
	int fin = 0;
	String archivo = null;
	double log2;
	double logNumDisp;
	double botPulsados;
	int botPulsadosEntero;
	int botonesTotales;

	public createAndShowGUI() {}
}
```

Una vez visto esto, se van a ir mostrando los diferentes métodos de esta clase y explicando su funcionalidad.

En primer lugar el método nombreArchivo() es por así decirlo el método principal de la clase, puesto que es el método al que llama la clase principal **WiiRemote** y a partir del cual van apareciendo las diferentes ventanas creadas (las cuales se crean empleando la biblioteca gráfica **Java Swing**). Las siguientes líneas son las empleadas en la clase **WiiRemote** para la creación de la interfaz:

```
int fin;
createAndShowGUI interfaz = new createAndShowGUI();
interfaz.nombreArchivo();
```

Como vemos se crea un objeto de la clase **createAndShowGUI** y a través de dicho objeto se llama al método que me arrancará la interfaz de usuario en sí.

El método *nombreArchivo()* creará un *JFrame* donde se agregará un JPanel, un JLabel, un *JTextFiel* y un *JButton*, de manera que al arrancar la aplicación aparecerá una ventana donde el usuario deberá introducir el nombre del archivo de **Active Home Pro** creado, para que de esta manera al pulsar el botón se salte al método crearInterfaz() pasándole como parámetros el propio evento de pulsación de botón y el nombre del archivo introducido por el usuario, para que en el siguiente método se ejecute por consola el archivo **Active Home Pro** creado por el usuario. Dicho archivo deberá estar guardado en la misma carpeta de la aplicación.

```
public void nombreArchivo(){

	frame3 = new JFrame("Nombre del archivo");
	panel4 = new JPanel();
	label3 = new JLabel ("Introduzca el nombre del archivo de Active
	Home Pro (sin la extensión)");
	campoTexto3 = new JTextField(10);
	boton4 = new JButton("Siguiente");

	panel4.add(label3);
	panel4.add(campoTexto3);
	panel4.add(boton4);
	frame3.add(panel4);

	boton4.addActionListener(new ActionListener(){
		public void actionPerformed(ActionEvent e){
			archivo = campoTexto3.getText();
			if(archivo.equals("")){
				String str = "Debe introducir el nombre del
				archivo ActiveHome Pro" + "\n";
				JOptionPane.showMessageDialog(null, str);
			}
			else{
				crearInterfaz(e,archivo);
			}
		}
	});
	frame3.addWindowListener(new java.awt.event.WindowAdapter(){
	    public void windowClosing(WindowEvent e){
	       System.exit(0);
	    }
	});
	frame3.pack();
	frame3.setLocationRelativeTo(null);
	frame3.setVisible(true);

}
```

Como vemos una vez que se pulsa el botón “Siguiente” en primer lugar se comprueba que se ha introducido el nombre del archivo, si no se ha introducido salta un mensaje de error y se vuelve a la ventana donde se debe introducir el nombre. Si se ha introducido el nombre correcto se enlaza con el siguiente método, el método *crearInterfaz()*. Este método lo primero que hará será llevar a cabo la ejecución por línea de comandos (empleando el código java creado para ello en la clase *PasoDeParametros* que explicaremos después) del archivo **Active Home Pro** creado por el usuario para que de esta manera pueda tener una visión en conjunto del escenario que él mismo ha creado. Simultáneamente se crea otro *JFrame* al que se le añaden los mismos elementos que en el caso anterior pero para que ahora el usuario añada el número de dispositivos que ha agregado al archivo de **Active Home Pro**. Una vez que el usuario ha agregado esto, ocurre

lo mismo que en el caso anterior, al pulsar el botón "Continuar" se enlaza con el método *actionPerformed()* pasándole como parámetro el propio evento de pulsación de botón.

```
    public void crearInterfaz(ActionEvent e,String archivo){

          String comando;
          comando = "cmd /c " + archivo + ".ahx";
          PasoDeParametros param = new PasoDeParametros(comando);
          param.envioComando();

          frame = new JFrame("Numero de dispositivos");
          panel = new JPanel();
          labelPpal = new JLabel("Introduzca el número de dispositivos que
          tiene (deben ser mínimo 2)");
          campoTexto = new JTextField(10);
          boton = new JButton("Continuar");

          panel.add(labelPpal);
          panel.add(campoTexto);
          panel.add(boton);
          frame.add(panel);

          boton.addActionListener(this);

          frame.addWindowListener(new java.awt.event.WindowAdapter(){
              public void windowClosing(WindowEvent e){
                 System.exit(0);
              }
          });

          frame.pack();
          frame.setLocationRelativeTo(null);
          frame.setVisible(true);

    }
```

Como ya hemos dicho, una vez que se ha pulsado el botón "Continuar" enlazamos con el método *actionPerformed()*, al cual le pasamos como parámetro el propio evento de pulsación de botón, este será por tanto, el método manejador de eventos. Este método se encarga de capturar el texto que el usuario introducido por teclado, empleando para ello el método *getText()* proporcionado por la clase *JTextField*. Este texto es el número de dispositivos que el usuario ha instalado. En primer lugar se comprueba si dicha cadena está vacía, en tal caso se lanza un mensaje de error. Si por el contrario la cadena contiene el número de dispositivos que el usuario ha agregado se calculará el valor entero de dicha cadena y el número de botones que se tienen que pulsar (a través del método *calcularBotonesPulsados()*, explicado posteriormente). Estos valores, junto con la cadena anteriormente capturada, se pasarán como parámetros al método *crearVentana()*, cuya funcionalidad se explicará a continuación.

```
    public void actionPerformed(ActionEvent e) {

      String cad;
      cad = campoTexto.getText();


      if(cad.equals("")){
          String str = "Debe introducir el número de dispositivos que
          tiene" + "\n";
          JOptionPane.showMessageDialog(null, str);
```

```
    }
    else{
        final int valorEnteroDeCad = Integer.parseInt(cad);
        botonesTotales = calcularBotonesPulsados (valorEnteroDeCad);

        if(boton == e.getSource()){
              crearVentana(cad,botonesTotales,valorEnteroDeCad);
        }
    }
}
```

A continuación se muestra el método *crearVentana()*, el cual, en base al parámetro *cad* (cadena que contiene el número de dispositivos), se encargará de agregar filas con tres columnas (empleando para ello un bucle for), una para introducir la dirección del dispositivo, otra para introducir el tipo de dispositivo que es (si permite atenuación o no) y una última que será el botón asociado a cada dispositivo, de forma que pulsándolo se agreguen a una lista enlazada todas las características asociadas a cada dispositivo agregado. Por lo que al hacer clic en el botón asociado a cada dispositivo se enlazará con el método *clicEnBoton2()*, al cual le pasaremos como parámetros el evento y las opciones seleccionadas por el usuario.

Una vez que se han agregado todos los dispositivos se pulsa el botón "Añadir" que enlazará con el método *clicEnBotonAñadir()*, del cual explicaremos su funcionalidad a continuación.

```
private void crearVentana(String cad, final int botonesTotales,final
int valorEnteroDeCad) {

   int i;
   int a;
   panel2 = new JPanel(new FlowLayout());
   panel2.setLayout(new java.awt.GridLayout((2*valorEnteroDeCad), 120 /
   120));
   for(i=0;!cad.equals(String.valueOf(i));i++){
       a = i + 1;
       String nom1 = "dispositivo "+a;
       JPanel panel6 = new JPanel();
       JPanel panel7 = new JPanel();
       JPanel panel8 = new JPanel();
       JPanel panel9 = new JPanel();
       JPanel panel10 = new JPanel();
       JPanel panel11 = new JPanel();
       JLabel label3 = new JLabel();
       JLabel label4 = new JLabel();
       JButton boton3 = new JButton();
       final Choice seleccion1 = new Choice();
       final Choice seleccion2 = new Choice();
       label3.setText("Introduzca la direccion del" + nom1);
       label4.setText("Introduzca el tipo de dispositivo: 'On y Off' o
       'Dim y Bright'");
       boton3.setText(nom1);
       seleccion1.add("A1");seleccion1.add("A2");seleccion1.add("A3");s
       eleccion1.add("A4");seleccion1.add("A5");seleccion1.add("A6");se
       leccion1.add("A7");seleccion1.add("A8");seleccion1.add("A9");sel
       eccion1.add("A10");seleccion1.add("A11");seleccion1.add("A12");s
       eleccion1.add("A13");seleccion1.add("A14");seleccion1.add("A15")
       ;seleccion1.add("A16");
       seleccion1.add("B1");seleccion1.add("B2");seleccion1.add("B3");s
       eleccion1.add("B4");seleccion1.add("B5");seleccion1.add("B6");se
       leccion1.add("B7");seleccion1.add("B8");seleccion1.add("B9");sel
       eccion1.add("B10");seleccion1.add("B11");seleccion1.add("B12");s
```

```
    eleccion1.add("B13");seleccion1.add("B14");seleccion1.add("B15")
    ;seleccion1.add("B16");
    seleccion2.add("On y Off");
    seleccion2.add("Dim y Bright");

    panel7.add(label3);
    panel8.add(seleccion1);
    panel9.add(label4);
    panel10.add(seleccion2);
    panel11.add(boton3);

    panel6.add(panel7);
    panel6.add(panel8);
    panel6.add(panel9);
    panel6.add(panel10);
    panel6.add(panel11);

    panel2.add(panel6);


    boton3.addActionListener(new ActionListener(){
         public void actionPerformed(ActionEvent e){

          clicEnBoton2(e,seleccion1.getSelectedItem(),seleccion2.getS
         electedItem());
         }
    });
}

 frame2 = new JFrame("Direcciones y tipos");
 boton2 = new JButton("Añadir");
 panel2.add(boton2);

 scrollBar = new JScrollPane(panel2);

  scrollBar.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConsta
  nts.HORIZONTAL_SCROLLBAR_ALWAYS);

  scrollBar.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstant
  s.VERTICAL_SCROLLBAR_ALWAYS);
 scrollBar.setViewportView(panel2);
  frame2.add(scrollBar,java.awt.BorderLayout.CENTER);

  boton2.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent e) {

          clicEnBotonAñadir(e,botonesTotales,valorEnteroDeCad);
       }
  });



  frame2.addWindowListener(new java.awt.event.WindowAdapter(){
       public void windowClosing(WindowEvent e){
                System.exit(0);
       }
  }
         );
```

```
        frame2.pack();
        frame2.setLocationRelativeTo(null);
        frame2.setVisible(true);

    }
```

Como vemos, al método *calcularBotonesPulsados()* se le pasa como parámetro el número de dispositivos que el usuario ha introducido. Este método, a través de unos cálculos matemáticos que consisten en el cálculo del logaritmo del número de dispositivos, dividido del logaritmo en base 10 de 2 proporcionará el número de botones que el usuario debe pulsar para seleccionar el dispositivo con el que quiere trabajar (posteriormente se explicará con detalle la forma que se ha usado para la selección de los dispositivos). Este método devuelve un valor entero que como digo corresponde al número de botones que se tienen que pulsar para la selección del dispositivo deseado, este valor se pasará después a la clase *MyListener* y a la clase *Lista*, puesto que tienen que trabajar con él, después se explicará cómo.

```
    private int calcularBotonesPulsados(int valorEnteroDeCad) {
          log2 = Math.log10(2);
          logNumDisp = Math.log10(valorEnteroDeCad);
          if(logNumDisp == 0){
                String str = "El número de dispositivos debe ser mínimo 2"
                + "\n";
                JOptionPane.showMessageDialog(null, str);
          }
          else{
                botPulsados = (logNumDisp)/(log2);
                if((logNumDisp)%(log2) != 0){
                      botPulsadosEntero = (int)(botPulsados) + 1;
                }
                else{
                      botPulsadosEntero = (int)(botPulsados);
                }
          }
          return botPulsadosEntero;
    }
```

Vemos que antes de realizar los cálculos se comprueba si el logaritmo del número de dispositivos es igual a 0 (cosa que ocurrirá cuando se tiene un único dispositivo), si esto ocurre salta de nuevo un mensaje de error indicando que se deben introducir mínimo dos dispositivos.

Como hemos dicho al pulsar el botón asociado a cada dispositivo, éste junto con sus características es agregado en una lista enlazada, empleando para ello un método declarado en la clase **Lista** que explicaremos a continuación. El método *clicEnBoton2()* es el siguiente.

```
    private void clicEnBoton2(ActionEvent e,String selec1,String selec2)
    {
          String cad2 = null;
          String cad3 = null;
          cad2 = selec1;
          cad3 = selec2;
          boolean crear = listaEnl.buscarRepe(cad2);
          if(crear){
                String str = "El dispositivo ya está creado" + "\n";
                JOptionPane.showMessageDialog(null, str);
          }
          else{
                listaEnl = listaEnl.addLast(cad2,cad3,false,0,null,false);
          }
```

```
}
```

Vemos que se comprueba si el dispositivo que se quiere agregar está guardado ya en la lista enlazada, a través del método *buscarRepe()* de la clase **Lista**, dicho método devolverá true en caso de que ese dispositivo ya se encuentre en la lista y por tanto se lanzará un mensaje de error. Si el dispositivo no está en la lista se añade a la misma a través del método *addLast()* de la clase **Lista**, el cual se explicará posteriormente.

Y finalmente al pulsar el botón "Añadir" se enlazará con el método clicEnBotonAñadir(), el cual cerrará las ventanas creadas por la interfaz, pondrá la variable fin a 1, lo que nos servirá para poder hacer uso de la lista enlazada creada, a continuación explicaremos cómo, y llama al método *asignarBoton()* de la clase **Lista** pasándole como parámetros el valor entero de los botones que se tienen que pulsar y el número de dispositivos que el usuario ha agregado, para que este método cree una matriz con las combinaciones de botones necesarias, después se explicará cómo.

```
private void clicEnBotonAñadir(ActionEvent e,int botonesTotales,int
valorEnteroDeCad){

      fin = 1;
      frame.dispose();
      frame2.dispose();
      frame3.dispose();
      listaEnl.asignarBoton(botonesTotales,valorEnteroDeCad);
      return;

}
```

Como vemos con el método anterior devolvemos el control a la clase principal, la cual pedirá el valor de la variable fin para así recuperar la lista creada y poder enviársela a la clase oyente (**MyListener**) que será la que realmente haga uso de la misma, ya que es esta clase la que relaciona las órdenes que recibe del ***WiiMote*** con las órdenes que se envían a los dispositivos domóticos. La clase principal también pedirá el valor del número de botones que se tienen que pulsar, el cual le hará falta para trabajar a la clase *MyListener*.

Los métodos que devuelven la lista creada, el número de botones y el valor de la variable fin están definidos en la clase que estamos describiendo y son los siguientes.

```
public Lista devolverLista(){

      return listaEnl;

}

public int devolverNumBot(){

      return botonesTotales;
}

public int pedirFin(){

      return fin;

}
```

Estos métodos como hemos dicho son llamados por la clase principal a través de las siguientes líneas de código.

```
Lista listaEnl2 = new Lista();
```

```
boolean bucle = true;
int botones = 0;
while(bucle){
	fin = interfaz.pedirFin();
	if(fin == 1){
		listaEnl2 = interfaz.devolverLista();
		botones = interfaz.devolverNumBot();
		bucle = false;
	}
}
```

Como vemos se crea una lista vacía donde se guardará la lista que se ha ido creando conforme el usuario añadía los valores requeridos. Después se crea un bucle "infinito" en el que se va pidiendo el valor de la variable fin, mientras esta no sea uno se va repitiendo el bucle y una vez que se recibe uno como valor de la variable fin, se pide a la interfaz que devuelva la lista creada y el número de botones, estos dos datos se mandan a la clase **MyListener** a través del método *recibirLista()* mostrado a continuación.

```
public void recibirLista(Lista listaEnl3, int numBotones){
	this.listaEnl3 = listaEnl3;
	this.numBotones = numBotones;
	boton = new int[numBotones];
}
```

Como vemos, el método cuando recibe el número de botones crea un array del tamaño de ese número que será donde se guarde, en cada posición del mismo el botón que se ha pulsado, de manera que para seleccionar el dispositivo con el que se quiere trabajar se buscará el dispositivo que tenga asociada dicha pulsación de botones, que como se ha comentado siempre serán combinaciones de los botones 1 y 2 del ***WiiMote*** esta asociación y búsqueda se realiza en la clase **Lista**, después se explicará cómo.

Por tanto, es necesaria la definición de una clase que se encargue de la creación de una lista enlazada (clase **Lista**) cuyos nodos van a tener diferentes campos y van a estar definidos en la clase **Nodo**. En primer lugar vamos por tanto a definir la clase **Nodo** para poder entender así los diferentes métodos de la clase **Lista**.

En la clase **Nodo** únicamente se definen los campos de los que va a constar un nodo de la lista enlazada, los cuales se explican a continuación:

- Campo **codCasaDisp**, el cual será un String donde se guardará la dirección del dispositivo en cuestión.

- Campo **tipo**, donde se guardará el tipo de dispositivo que es ("Dim y Bright" o "On y Off").

- Campo **estado**, que será un booleano que contendrá un valor true si el dispositivo está ON y un valor false si el dispositivo está OFF.

- Campo **porcBrillo**, en este campo se guardará un valor entero que será el porcentaje de brillo del dispositivo. Este campo siempre tendrá un valor entre 0% y 100% con aumentos de 10% para los dispositivos que permitan tal atenuación, mientras que siempre valdrá 0 para los dispositivos que únicamente permiten encendido y apagado.

- Campo **botón**. Este campo se define como un array de enteros dónde se guarda la combinación binaria entre los botones 1 y 2 que le corresponde a este nodo para poder así seleccionarlo (a través del campo **seleccionado** explicado a continuación) cuando se desee trabajar con él.

- Campo **nodoDer**, está declarado como tipo **Nodo** y es usado como puntero para enlazar con el siguiente elemento de la lista enlazada.

- Campo **seleccionado**, será un booleano el cual se pondrá a true cuando el usuario pulse los botones asociados a dicho dispositivo para seleccionarlo y poder así trabajar con él. Cuando el usuario termine de trabajar con él pulsará otro botón del mando para que este campo se ponga a false.

Y dos métodos constructores, uno que se le pasarán como parámetros los valores requeridos y otro que creará un nodo vacío, el cual se empleará como auxiliar en los casos que haya que recorrer la lista enlazada.

```
public Nodo(String codCasaDisp,String tipo, boolean estado,int
porcBrillo,int []boton,boolean seleccionado) {
        this.codCasaDisp =  codCasaDisp;
        this.tipo = tipo;
        this.estado = estado;
        this.porcBrillo = porcBrillo;
        this.boton = boton;
        this.nodoDer = null;
        this.seleccionado = seleccionado;
    }


public Nodo(){}
```

Una vez que conocemos cómo se han distribuido los nodos de la lista y para lo que sirve cada campo vamos a explicar a continuación la funcionalidad de cada uno de los métodos creados en la lista enlazada.

En primer lugar en la lista se declaran dos variables tipo **Nodo** denominadas primero y último para tener siempre una referencia al primer y último elemento de la lista. También se declaran tres variables enteras, de las cuales dos se utilizarán en bucles y en la otra se guardará el tamaño de la lista.

La clase **Lista** tendrá en primer lugar un método constructor en el que simplemente se inicializarán las variables primero y último a null y la variable entera *tamaño* se inicializará a cero, ya que la lista estará vacía al crearla.

```
public Lista() {
     this.primero = null;
     this.ultimo = null;
     this.tamano = 0;
}
```

Después simplemente se crea un método que comprobará si la lista está vacía, ya que esta comprobación es usada a lo largo de la clase en varias ocasiones. Para ello únicamente se comprueba si el nodo primero está apuntando a null, devolviendo true en tal caso, significando esto que la lista está vacía.

```
public boolean siVacio() {
    return (this.primero == null);
}
```

A continuación he creado un método para agregar los elementos en la lista enlazada, este método va agregando elementos por el final de la lista, de manera que el elemento más nuevo será siempre el último.

```
public Lista addLast(String codCasaDisp,String tipo, boolean
estado,int porcBrillo,int []boton,boolean seleccionado) {
      if(siVacio()) {
```

```
            Nodo nuevo = new
            Nodo(codCasaDisp,tipo,estado,porcBrillo,boton,seleccionado);
            primero = nuevo;
            ultimo = nuevo;
            nuevo.nodoDer = null;
        }

        else {
            Nodo nuevo = new
            Nodo(codCasaDisp,tipo,estado,porcBrillo,boton,seleccionado);
            nuevo.nodoDer = null;
            ultimo.nodoDer = nuevo;
            ultimo = nuevo;
        }
        this.tamano++;
        return this;

    }
```

Como vemos, en este método se pasan como parámetros el valor de los campos de los que debe estar formado cada nodo. El campo botón también se pasa como parámetro pero con un valor null (esto se puede observar en el método *clicEnBoton2()* de la clase **createAndShowGUI**), puesto que después en otro método de la lista, que se explicará a continuación, se le va a dar el valor que le va a corresponder.

Como hemos dicho los elementos se agregan por el final de la lista, por lo que el mecanismo para agregarlos será el siguiente: en primer lugar tendremos que comprobar si la lista está vacía o no. En caso de que esté vacía, se crea el nuevo nodo con los parámetros correspondientes y el primero y el último nodo (puesto que son el mismo al estar la lista vacía) pasan a ser el nuevo nodo y el puntero del nuevo nodo que apunta a siguiente (nodoDer) pasa a apuntar a null. Si la lista no está vacía del mismo modo se crea un nuevo nodo con los parámetros correspondientes y se pone, al igual que en el caso anterior, que el puntero que apunta a siguiente del nuevo nodo, apunte a null; que el nodoDer del último apunte a nuevo y que el último nodo pase a ser nuevo.

Al final del método se incrementa en uno el tamaño de la lista y se devuelve la lista que está siendo creada.

A continuación, en esta misma clase se define el método *asignarBoton()*, el cual es llamado en la clase **createAndShowGUI**, como ya se ha visto anteriormente y se le pasan como parámetros el número de dispositivos de los que se dispone y el número de botones que se tienen que pulsar (*numDisposisitivos* y *botonesTotales*), este método es el siguiente.

```
    public void asignarBoton(int botonesTotales,int numDispositivos){

        int matriz[][] = new int[numDispositivos][botonesTotales];
        boolean cambio = true;
        int i = 0;
        int j;
        for(j = matriz[i].length - 1; j>= 0; j--){
              for(i = 0; i < matriz.length; i++){

                  if(cambio == true){
                      matriz[i][j] = 1;
                  }
                  else{

                      matriz[i][j] = 2;
                  }
```

```
            if(((i+1) % (Math.pow(2,matriz[i].length-1-j))) == 0){

                if(cambio == true){
                    cambio = false;
                }
                else{
                    cambio = true;
                }
            }
          }
          cambio=true;
        }

      Nodo aux = null;
      if(tamano != 0){
            aux = primero;

            for (int k = 0; k < this.tamano; k++){
                  aux.boton = matriz[k];
                  aux = aux.nodoDer;
            }

      }
}
```

Como se ha mencionado varias veces, lo que se va a emplear para seleccionar el dispositivo con el que se desea trabajar en cada momento, es la combinación de la pulsación de los botones 1 y 2. Las posibles combinaciones que me van a determinar el dispositivo seleccionado se crean, como ya se ha comentado, como combinaciones binarias de ambos botones, esto es, si se tienen cuatro dispositivos se deberán pulsar dos botones para la selección de cada uno (el número de botones se determina con los cálculos del método correspondiente en la clase **createAndShowGUI**), de manera que con dos botones se podrán conseguir hasta cuatro combinaciones, las cuales son 11, 12, 21 y 22. Estas combinaciones se podrán obtener desde un botón hasta ocho, puesto que el número máximo de dispositivos es 256 (2^8).

Entendiendo esto es más fácil comprender el funcionamiento del método anteriormente declarado, que será el siguiente: con los valores que se han pasado al método se ha creado una matriz de enteros de tales dimensiones, de manera que las filas de la matriz corresponderán al número de dispositivos de los que se dispone y las columnas al número de botones que se tienen que pulsar para seleccionar dichos dispositivos. Dicha matriz se irá rellenando por columnas desde la última hasta la primera, alternando los valores de cada posición de la matriz según el siguiente criterio `((i+1) % (Math.pow(2,matriz[i].length-1-j))) == 0`, esto es, en la última columna los valores se van alternando (1 y 2), en la penúltima los valores cambian de dos en dos (1, 1 y 2, 2), en la anterior de cuatro en cuatro y así sucesivamente hasta terminar con la primera columna. Como se ha comentado, las dimensiones de la matriz están adaptadas a los dispositivos que se tienen, por lo que se creará una nueva matriz cada vez que se arranque la aplicación.

Una vez que se ha creado la matriz se recorre la lista enlazada para asignarle a cada nodo la fila de la matriz que le corresponde, de manera que, si por ejemplo se tienen que pulsar dos botones, el primer dispositivo que se ha añadido tendrá en su campo **botón** asociado el array 11, el segundo el array 12, y así sucesivamente.

A continuación se definen en la clase **Lista** una serie de métodos para buscar dispositivos en función de uno de sus parámetros, los cuales devuelven el nodo buscado. Únicamente se va a poner aquí uno de esos métodos puesto que los otros actúan del mismo modo aunque buscando un parámetro diferente.

```
public Nodo buscar (String codCasaDisp) {

      Nodo aux = null;

      if(siVacio()){
         return null;
      }
      else{
         aux = primero;
         while((aux != null) &&
         (!((aux.codCasaDisp).equals(codCasaDisp)))){
                  aux = aux.nodoDer;
         }
         if((aux != null) && ((aux.codCasaDisp).equals(codCasaDisp))){
            return aux;
         }

         else{
            return null;
         }
      }
 }
```

En este método se busca el dispositivo por su dirección del hogar. Para ello se pasa como parámetro un String indicando la dirección que se busca y se crea un nodo auxiliar vacío que se usará para la búsqueda. Comprobamos si la lista está vacía, en caso de ser así se devuelve un null puesto que no existe el nodo buscado. Si la lista no está vacía, se pone el nodo auxiliar apuntando a primero para comenzar a recorrer la lista, por lo que se crea un bucle while en el que se va comprobando en primer lugar, que el nodo no sea null y después si el campo “codCasaDisp” del nodo aux coincide con el código de hogar buscado y en caso de que no sea así se va pasando al siguiente nodo. Cuando se encuentra la dirección pedida se devuelve el nodo en concreto y si no se encuentra, es decir, el dispositivo no existe, se devuelve null.

Como he mencionado existen otros métodos para buscar dispositivos en función de otros parámetros. Se crea uno para buscar también por la dirección pero que en este caso devuelve un valor booleano a true si se encuentra en la lista el dispositivo buscado, este método es usado cuando se están agregando dispositivos en la lista para comprobar que los códigos de hogar y dispositivo no se repiten. Después se crean otros para buscar por tipo, por botón y por campo seleccionado, el cual devolverá el nodo que tenga el campo seleccionado a true. Estos tres últimos actúan igual que el explicado anteriormente y son llamados en la clase **MyListener** (después se explicará su funcionalidad concreta).

Por otro lado se han creado dos métodos para cambiar el porcentaje de brillo y el estado del dispositivo, en primer lugar explicaremos el método que cambia el parámetro brillo.

```
public void cambiarBrillo(String codCasaDisp, int porcBrillo){

      Nodo aux = null;
      if(siVacio()){
         System.out.println("La lista está vacía");
      }
      else{
         aux = primero;
```

```
            while((aux != null) &&
            (!((aux.codCasaDisp).equals(codCasaDisp)))){
               aux = aux.nodoDer;
            }
            if((aux != null) && ((aux.codCasaDisp).equals(codCasaDisp))){
               aux.porcBrillo = porcBrillo;
               if(aux.porcBrillo > 0){
                     aux.estado = true;
               }
               else if(aux.porcBrillo == 0){
                     aux.estado = false;
               }
               System.out.println("Se ha cambiado el brillo a " +
               aux.porcBrillo);
            }

            else{
               System.out.println("No existe el dispositivo pedido");
            }
         }
      }
```

Como vemos se pasan como parámetros la dirección del dispositivo del cual queremos cambiar el brillo y el valor que queremos darle al mismo. Al igual que en el método anterior se crea un nuevo nodo que servirá como auxiliar para recorrer la lista en busca del dispositivo requerido, por lo que esta función se hará igual que en los métodos de búsqueda. Lo único que cambia aquí es que una vez que se encuentra el dispositivo se le asigna el valor del brillo que se ha pasado y se comprueba si el valor del mismo es mayor que 0 o igual a cero, para cambiar el estado a true o false según corresponda.

El método creado para cambiar el estado es el siguiente:

```
   public void cambiarEstado(String codCasaDisp, boolean estado){

         Nodo aux = null;
         if(siVacio()){
            System.out.println("La lista está vacía");
         }
         else{
            aux = primero;
            while((aux != null) &&
            (!((aux.codCasaDisp).equals(codCasaDisp)))){
               aux = aux.nodoDer;
            }
            if((aux != null) && ((aux.codCasaDisp).equals(codCasaDisp))){
               if(estado == false){
                     aux.estado = estado;
                     aux.porcBrillo = 0;
               }
               else if(estado == true) {
                     aux.estado = estado;
                     aux.porcBrillo = 100;
               }

               System.out.println("Se ha cambiado el estado a " +
               aux.estado);
            }
```

```
            else{
              System.out.println("No existe el dispositivo pedido")
            }
        }
}
```

Como vemos se pasan como parámetros la dirección del hogar del dispositivo del que se quiere modificar el estado y el nuevo estado que se le va a dar. Como en los métodos anteriores se comprueba si la lista está vacía y en caso contrario se recorre la lista empleando un método while. Una vez que se ha encontrado la dirección buscada se comprueba cual es el estado al que se quiere modificar, si es false se cambia el estado del dispositivo a false y el porcentaje de brillo a cero, independientemente del tipo de dispositivo que sea, y si el estado es true cambiamos el estado a true y el porcentaje al 100%.

También se ha creado un método para cambiar el campo seleccionado del dispositivo concreto el cual actúa del mismo modo que los anteriores.

Finalmente se ha creado en esta clase un método que imprime en una ventana la dirección de cada dispositivo, el tipo que es y la combinación de botones que se tienen que pulsar para seleccionarlo, dicho método es el siguiente:

```
    public void imprimirBotones() {

            if(tamano != 0) {
                Nodo temp = primero;
                String str = "";

                for(int i = 0; i < this.tamano; i++) {
                      int []boton = temp.boton;
                      str = str + temp.codCasaDisp + " "+ temp.tipo + " ";
                      for(int j = 0; j<boton.length;j++){
                            str = str + boton[j];
                      }
                      str = str + "\n";
                      temp = temp.nodoDer;
                }
                JOptionPane.showMessageDialog(null, str);
            }
```

Como vemos en este método se comprueba en primer lugar si el tamaño de la lista es distinto de cero, en tal caso se crea un nodo que apunta a primero para ir recorriendo la lista (con un bucle for) e ir almacenando para cada nodo, en una variable tipo String el código del hogar, el tipo y sus botones asociados (los cuales se obtienen recorriendo con un bucle for el array donde se almacena dicha combinación). Finalmente se imprime todo en filas en una ventana.

Por último la clase que permite el envío de órdenes a los dispositivos es la clase **PasoDeParametros**, la cual posibilita el envío de comandos por consola, puesto que este será el mecanismo a través del cual se envíen las órdenes, empleando para ello un fichero llamado **ahcmd.exe** el cual llamándolo por consola con una sintaxis adecuada ejecuta la orden pedida. El funcionamiento de este fichero, como ya se ha mencionado, se explicará más adelante en el **Anexo 3**.

La clase **PasoDeParametros** tiene únicamente dos métodos, el método constructor y el propio método que se asocia con la consola. Solamente se declara una variable tipo String llamada "comando" donde se guardará el comando que se va a ejecutar. El método constructor es el siguiente:

```
public PasoDeParametros (String comando){
      this.comando=comando;
}
```

Como vemos únicamente se pasa el comando (que dependerá de la acción que se quiera ejecutar).

El método encargado de enviar la orden es el siguiente:

```
public void envioComando(){

      try{
          Runtime.getRuntime().exec(comando);
      } catch (IOException e) {
          System.out.println("Excepción: ");
          e.printStackTrace();
          System.exit(-1);
      }
}
```

Estas líneas pueden lanzar una Runtime Exception y es por eso que se escriben dentro de un try-catch para capturar la excepción en caso de que ocurra. La línea `Runtime.getRuntime().exec(comando);` se encarga de ejecutar el comando que se ha pasado por consola, en caso de que esto falle se captura la excepción y se cierra el programa.

Esta clase es llamada en la clase **MyListener** cada vez que el mando envía un evento, puesto que es en tal caso cuando se manda una orden. Y esto es lo que nos lleva finalmente al siguiente y último apartado del capítulo.

4.3. *WiiMote* + domótica.

Aquí se va a detallar en que parte del código entran en contacto ambas partes del proyecto, que como he dicho anteriormente es en la clase **MyListener** cuando el ***WiiMote*** envía un evento, ya sea la pulsación de un botón o un movimiento del mismo.

Al principio de este capítulo se ha descrito cómo funciona el método que detecta los eventos de pulsación de botones del ***WiiMote***, como se ha comentado, en dicho método se definen una serie de sentencias condicionales en las que se comprueba en cada una de ellas que botón se ha pulsado y en función de esto se pone una variable booleana asociada a cada botón a true. Esto se hace así porque después va a ser necesario evaluar dichas variables booleanas para comprobar si se ha pulsado un botón concreto.

El funcionamiento de esta clase es el siguiente: como se ha dicho antes esta clase recibe la lista y el número de botones que se tienen que pulsar para la selección del dispositivo a través del método *recibirLista()* puesto que estos serán los dos parámetros principales con los que tendrá que trabajar esta clase, ya que para ir creando el array que contendrá los botones pulsados será necesario comparar el valor de un contador (creado para hacer referencia a cada una de las posiciones del array) con el número de botones que se tienen que pulsar. A continuación se muestra esta funcionalidad.

```
while(((Uno) || (Dos)) && (contador < numBotones)){
   if(Uno){
      Uno = false;
      boton[contador] = 1;
      contador++;
   }
   else if(Dos){
      Dos = false;
      boton[contador] = 2;
      contador++;
   }
}
if(contador == numBotones){
   nodo1 = listaEnl3.buscarBoton(boton,numBotones);
   if(nodo1 == null){
      String cadena2 = "No existe el dispositivo que tiene asociados
      esos botones" + "\n";
      JOptionPane.showMessageDialog(null, cadena2);
   }
  else{
      String direccion = nodo1.codCasaDisp;
      boolean estado = nodo1.estado;
      String tipo = nodo1.tipo;
      int brillo = nodo1.porcBrillo;
      listaEnl3.cambiarSeleccion(direccion, true);
      String cadena = "Su dispositivo tiene las siguientes
      características: ";
      if(tipo.equals("On y Off")){
            cadena = cadena + " direccion: "+ direccion + " estado: "
            + estado + " brillo: " + brillo + " tipo: " + tipo + "\n" +
            "¿Qué desea hacer?" + "\n" + "encender o apagar: A " + "\n"
            + "Pulse Home cuando termine con el dispositivo" + "\n";
            JOptionPane.showMessageDialog(null, cadena);
      }
      else{
            cadena = cadena + " direccion: "+ direccion + " estado: "
            + estado + " brillo: " + brillo + " y es del tipo: " + tipo
            + "\n" + "¿Qué desea hacer?" + "\n" + "encender o apagar: A
            " + "\n" + "aumentar brillo: + " + "\n" + "disminuir
            brillo: - " + "\n" + "Pulse Home cuando termine con el
            dispositivo" + "\n";
            JOptionPane.showMessageDialog(null, cadena);
      }
   }
   contador = 0;
}
```

Como vemos, se comprueba en un bucle while si se ha pulsado el botón 1 o el 2 y si la variable *contador* (que se ha inicializado a cero) vale menos de *numBotones* (número de botones que se han pulsado, dicho valor se pasa a esta clase a través del método *recibirLista()*), puesto que, se deberá rellenar el array creado en función de este número. Si por ejemplo se tienen que pulsar dos botones, se rellenarán las posiciones 0 y 1 del array. Una vez que dicho array se ha creado, ése es el que se tendrá que buscar en la lista, a través del método *buscarBoton()* de la clase **Lista** descrito anteriormente.

Una vez que se ha terminado de rellenar el array (lo cual se comprueba comprobando en una sentencia condicional si el valor de *contador* coincide con *numBotones*), se llama al método *buscarBoton()* pasándole como parámetros el array que se tiene que buscar y el número de botones que se tienen que pulsar, este método devuelve el nodo cuyo campo **botón** corresponde al array que se está buscando o null si no se ha

encontrado el dispositivo, es por esta razón que se comprueba si el nodo que devuelve el método es null y en tal caso salta un mensaje de error. En caso contrario, se recuperan los datos de ese dispositivo, la dirección, el estado, el tipo y el brillo y se cambia su campo **seleccionado** a true. Una vez que se tienen guardados en variables todos los datos del dispositivo, se comprueba de que tipo es y se lanza un mensaje en el que se describen las características del dispositivo y las acciones que se pueden llevar a cabo sobre el mismo, que únicamente serán de encendido y apagado si el dispositivo es del tipo *On y Off* y de atenuación e incremento de brillo (además de encendido y apagado) si el dispositivo es del tipo *Dim y Bright*.

Una vez que sabemos cómo se selecciona el dispositivo con el que se desea trabajar debemos conocer que eventos debe enviar el ***WiiMote*** para que se envíe la orden adecuada al dispositivo concreto. Únicamente se va a emplear la pulsación de tres botones para el envío de órdenes a los dispositivos domóticos puesto que solamente se pueden producir tres acciones: encendido/apagado, atenuación e incremento del brillo.

- Para la orden de **encendido y apagado** se empleará la pulsación del botón ***A***.

```
if(A){
   nodo2 = listaEnl3.buscarSeleccion();
   if(nodo2 == null){
         String cadena2 = "No se ha seleccionado ningún dispositivo"
         + "\n";
         JOptionPane.showMessageDialog(null, cadena2);
   }
   else{
         if(nodo2.seleccionado){
               String direccion = nodo2.codCasaDisp;
               boolean estado = nodo2.estado;
               if(!estado){
                     String comando;
                     comando = "cmd /c ahcmd sendplc " + direccion +
                     " On";
                     PasoDeParametros param = new
                     PasoDeParametros(comando);
                     param.envioComando();
                     listaEnl3.cambiarEstado(direccion,true);
               }

               else{
                     String comando;
                     comando = "cmd /c ahcmd sendplc " + direccion +
                     " Off";
                     PasoDeParametros param = new
                     PasoDeParametros(comando);
                     param.envioComando();
                     listaEnl3.cambiarEstado(direccion,false);
               }
         }
   }
   A = false;

}
```

Vemos que en primer lugar se busca, a través del método *buscarSeleccion()* el nodo que tiene el campo **seleccionado** a true, es decir, que el usuario lo ha seleccionado previamente. Si no se encuentra salta un mensaje de error. En caso contrario, si se encuentra, se obtiene su dirección y su estado y en función de éste se manda orden de encendido o apagado a través de la clase **PasoDeParametros** explicada anteriormente. Para enviar dicha orden simplemente, se crea una variable tipo String donde se guardará el

comando que se quiere enviar con la siguiente sentencia `comando = "cmd /c ahcmd sendplc " + direccion + " On";` como se observa lo único que cambiará en esta sentencia será la dirección del dispositivo, guardada en "dirección", además de la orden en función del estado. En este caso la orden enviada será "ON" puesto que el dispositivo está apagado. Para que el comando se ejecute por línea de comandos es necesario ejecutar la orden "cmd /c" seguida de "ahcmd sendplc" donde con "ahcmd" se llama al ejecutable "ahcmd.exe" que será el que envíe la orden al dispositivo domótico, y "sendplc" corresponde a la orden que se envía por la línea eléctrica, aunque hay otras muchas, como por ejemplo "sendrf" que envía las ordenes empleando la radiofrecuencia, pero esto se explicará de manera más detallada en el **Anexo 3**. Una vez que se ha escrito la sentencia del comando que se quiere enviar se crea una variable del tipo **PasoDeParametros** pasándole como parámetro el propio comando, y después se "envía" para que se ejecute por consola con el método *envioComando()*. Una vez hecho esto habrá que cambiar el estado del dispositivo con el método *cambiarEstado(direcc,estado)* de la clase **Lista**.

- Para la orden de **incremento de intensidad** se empleará la pulsación del botón **+**.

- Para la orden de **atenuación de intensidad** se empleará la pulsación del botón **-**.

Para estas dos órdenes el mecanismo a seguir es el mismo que para la orden de encendido y apagado, únicamente la diferencia será que también se debe recuperar el porcentaje de brillo del dispositivo en cuestión puesto que es necesario conocer este valor para el envío de la orden, ya que a la hora de crear el comando es necesario indicar el porcentaje de brillo que se desea dar al dispositivo en cuestión, como se muestra en la siguiente sentencia `comando = "cmd /c ahcmd sendplc " + direccion + " Bright 010";`. En este caso la orden es de incremento de intensidad, por ello se usa el parámetro *Bright*, y el resto de la sentencia es igual que en el caso anterior.

Como se ha explicado, cuando un dispositivo se selecciona su campo **seleccionado** se pone a true y mientras este valor este a true se pueden enviar al dispositivo las órdenes deseadas, sin embargo se debe permitir deseleccionar dicho dispositivo con el que se trabaja, para poder así seleccionar otro en otro momento. Esta acción se lleva a cabo pulsando el botón ***Home***, cuando se pulsa este botón simplemente se busca el dispositivo que tiene el campo **seleccionado** a true y se pone a false a través del método *cambiarSeleccion()* de la clase **Lista**.

Todo lo que se ha explicado en este capítulo se va a poder visualizar gráficamente en el siguiente capítulo, en el que se va a detallar el **caso de estudio** que ha permitido el desarrollo de este PFC.

CAPÍTULO 5.

Caso de estudio.

En este capítulo se va a proceder a explicar el **caso de estudio** de este proyecto, en el que se va a detallar el material del que se disponía, y por tanto del que se ha hecho uso, para realizar las pruebas que finalmente han sido la clave para el desarrollo de este PFC tal y como ha quedado.

5.1. Material usado.

El material del que se disponía en el laboratorio para realizar las pruebas es el siguiente.

- Panel en el que se recrea una planta de una vivienda que consta de dos dormitorios, un salón, un baño, un aseo, pasillo y cocina. En cada una de estas salas se encuentra enchufado un módulo que soporta tecnología X10 (de los tipos *appliance module*, *lamp module* y *receptor RF*) y a cada uno de ellos se encuentra enchufado el aparato al que se le desea dar control domótico. En este caso sólo se dispone de luces, sin embargo los módulos de tipo *appliance module* son los que permiten únicamente las acciones de encendido y apagado, y en un hogar serán por tanto los que se conecten a aparatos que permitan estas acciones, tales como cafeteras, televisión, lavavajillas, etc…aunque también se pueden conectar, como es el caso, a lámparas que no permitan atenuación ni incremento de brillo.

Figura 5.1. Panel de trabajo.

Como se observa en la foto, en la parte superior, también se dispone del controlador CM11A puesto que es indispensable para el envío de órdenes al resto de módulos X10.

- Controlador de la *Wii* ***WiiMote***. Esta ha sido la herramienta más importante para el desarrollo de este PFC, puesto que una detección de evento del mismo se ha traducido en una orden concreta al dispositivo adecuado, a continuación explicaremos cómo.

Figura 5.2. *WiiMote*.

5.2. Puesta en marcha de la aplicación.

En este apartado se va a proceder a explicar todo los pasos que se han tenido que llevar a cabo en el laboratorio para hacer las pruebas pertinentes del código creado.

Como hemos dicho se dispone de un panel con siete módulos X10, y como sabemos a cada uno deberemos asociarle un código de hogar y dispositivo concreto, de manera que cuando se envíe la orden el controlador CM11A sepa a qué dispositivo debe reenviarla.

Estas direcciones pueden tomar cualquier valor desde la A a la P (código del hogar) y desde el valor 1 hasta el 16 (código del dispositivo). A continuación se va a mostrar una tabla con las diferentes habitaciones, los módulos X10 que hay conectados, la acción que permiten dichos módulos y sus códigos de hogar y dispositivo.

HABITACIÓN.	MÓDULO X10.	ACCIÓN PERMITIDA.	CÓDIGO DE HOGAR Y DISPOSITIVO.
Pasillo	Receptor RF más actuador TM751.	- On/Off. - Dim y Bright.	A1.
Dormitorio 2.	Módulo de aparato AM12.	- On/Off.	A2.
Dormitorio 1.	Módulo de lámpara LM7245.	- On/Off. - Dim y Bright.	A5.

Baño.	Módulo de aparato AM7256.	- On/Off.	A4.
Salón.	Módulo de lámpara LM565.	- On/Off. - Dim y Bright.	B4.
Cocina.	Módulo de casquillo LM15ES.	- On/Off.	B3.
Aseo.	Receptor RF/MA TM13.	- On/Off.	B1.

Figura 5.3. Tabla de direcciones de los módulos de los que se dispone.

Estas direcciones se deben asignar manualmente a cada uno de los módulos, para que, como ya se ha comentado, el controlador CM11A los reconozca y pueda enviar las órdenes requeridas. Una vez que estas direcciones están asignadas a cada módulo se deberá crear el archivo del programa **Active Home Pro** el cual, como sabemos, proporcionará la interfaz gráfica con el usuario, de manera que éste podrá ver en la pantalla del PC cada uno de los módulos que tiene y su estado. Una vez que se han agregado todos los módulos en este archivo (como se explica en el **anexo 5**) queda la siguiente interfaz.

Figura 5.4. Archivo Active Home Pro con módulos de caso de estudio agregados.

Una vez que se ha creado el archivo, dicho fichero, así como el archivo *ahcmd.exe* se deben guardar en la misma carpeta donde se encuentra la aplicación aquí desarrollada, es decir, la carpeta del proyecto, puesto que posteriormente el código creado hará uso de ambos archivos.

Ya tenemos, por tanto, el archivo que va a ser necesario para la ejecución de la aplicación y conocemos todos los datos de la red domótica creada, por lo que ahora se va a proceder a ejecutar la aplicación aquí desarrollada.

1. En primer lugar se deberá establecer la conexión *bluetooth* entre el PC y el ***WiiMote*** tal y como se explica en el **anexo 1**.

2. Una vez que se ha establecido la conexión correctamente se deberá arrancar la aplicación. Cuando dicha aplicación arranca, lo primero que se pedirá es que el usuario introduzca el nombre del archivo **Active Home Pro** para que la aplicación lo ejecute por línea de comandos, en este caso el archivo creado se ha llamado "Casa".

Figura 5.5. Ventana "Nombre del archivo".

De manera que, cuando se pulse el botón "Siguiente" aparecerá el archivo que previamente se ha creado (en la siguiente imagen) y una nueva ventana.

Figura 5.6. Archivo Active Home Pro "Casa".

3. En esta nueva ventana que aparece se deberá introducir el número de dispositivos de los que se dispone, que en este caso será 7.

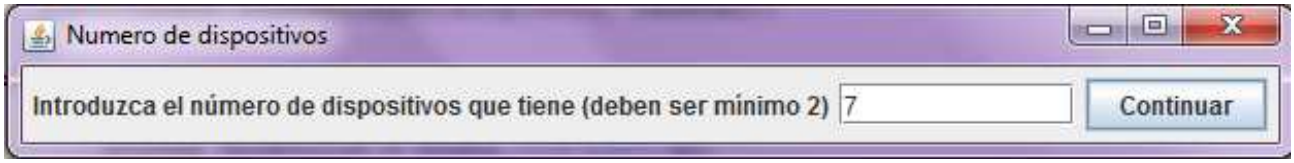


Figura 5.7. Ventana "Número de dispositivos".

4. Cuando se haya pulsado el botón “Continuar” aparecerá una ventana, como la siguiente, en la que habrán tantas filas como dispositivos, 7 en este caso, en las que se deberán introducir los datos pedidos, que simplemente serán los códigos de hogar y dispositivo de cada módulo así como el tipo, que corresponderá a *On y Off* (si únicamente permite encendido y apagado) y *Dim y Bright* (si también permite atenuación e incremento de brillo). Cuando se introduzcan dichos datos se deberá pulsar el botón que hay asociado a cada dispositivo, para que se guarde en la lista enlazada con todas sus características, y una vez que se termine de introducir todos los dispositivos se deberá pulsar el botón “Añadir”.

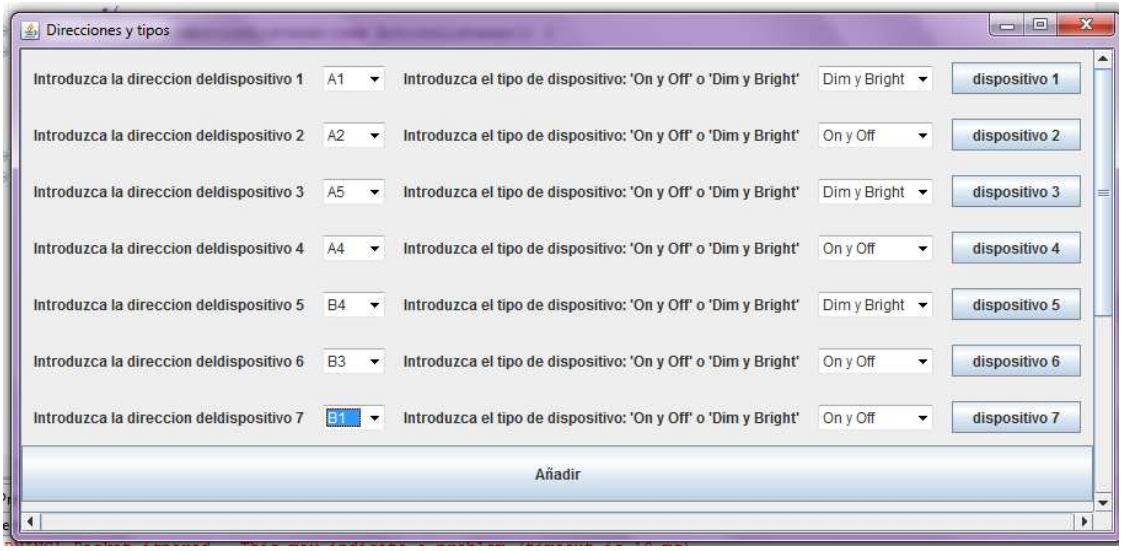


Figura 5.8. Ventana “Direcciones y tipos”.

Como vemos en la ventana anterior ya se han introducido todos los datos de los módulos de los que se han detallado todas sus características en la tabla de la **figura 5.3**.

5. Una vez que todos los dispositivos están en la lista ya se puede hacer uso de la aplicación, de manera que, pulsando el botón “*Abajo*” del ***WiiMote***, nos aparecerá una ventana en la que se informará al usuario de las direcciones, el tipo y la combinación de botones que debe pulsar para seleccionar cada uno de ellos, dicha ventana es la siguiente.

Figura 5.9. Mensaje de información necesario para el manejo de los dispositivos.

6. Una vez que sabemos qué combinación de botones debemos pulsar para seleccionar cada dispositivo y poder así hacer uso del mismo, procedemos a trabajar con la aplicación. De manera que si deseamos trabajar con el dispositivo con dirección A1 (que en este caso es el del pasillo y permite atenuación), pulsaremos la combinación de botones 111 y nos aparecerá la siguiente ventana con la información del dispositivo seleccionado, así como las acciones que se pueden llevar a cabo sobre el mismo.

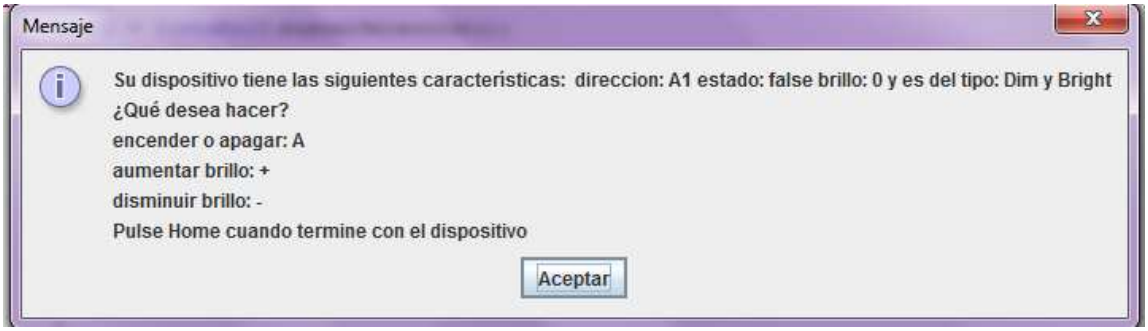


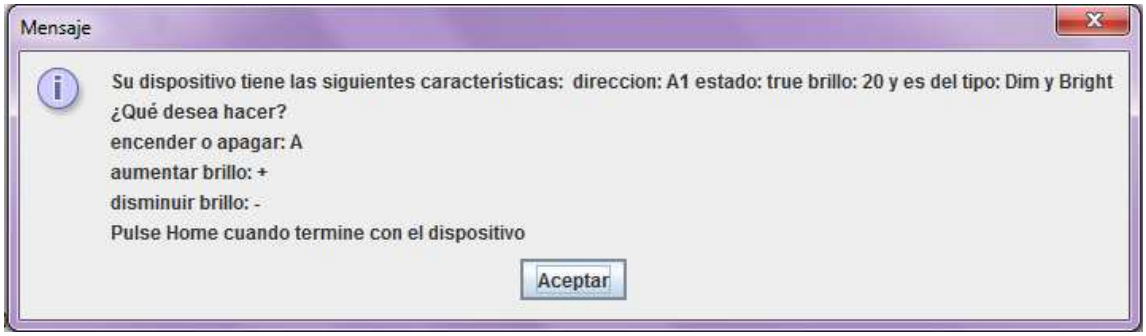
Figura 5.10. Mensaje de información del dispositivo seleccionado.

Como vemos, este dispositivo está apagado, por tanto su porcentaje de brillo es cero y es del tipo *Dim y Bright*, por lo que se podrá encender y apagar (pulsando el botón ***A***), incrementar el brillo (pulsando el botón ***+***) y disminuir el brillo (pulsando el botón **-**).

Si por ejemplo, pulsamos el botón ***+*** dos veces, se incrementará el brillo en un 20 por ciento y se podrá observar el cambio tanto en el dispositivo del panel como en el archivo "Casa" del software **Active Home Pro** como se muestra en la siguiente imagen.

Figura 5.11. Muestra de incremento de brillo del dispositivo con dirección A1.

Si de nuevo volvemos a pulsar los botones de selección del dispositivo (111) nos aparecerá la misma



ventana de antes pero con los nuevos parámetros.

Figura 5.12. Mensaje de información con los nuevos parámetros del dispositivo en cuestión.

Si finalmente pulsamos el botón ***A*** para encender el dispositivo se traducirá de la siguiente manera en el archivo "Casa".

Figura 5.13. Muestra de cambio al estado On del dispositivo con dirección A1.

Y al volver a seleccionarlo (111) aparecerá el siguiente mensaje.

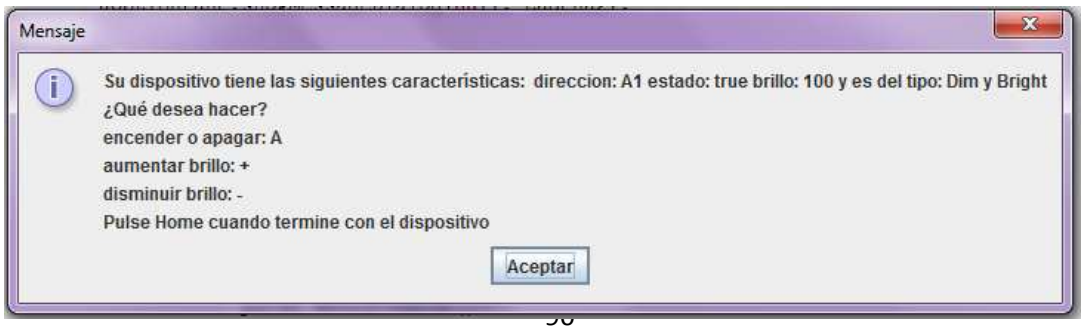


Figura 5.14. Mensaje de información con los nuevos parámetros del dispositivo en cuestión.

Vemos como finalmente el estado del dispositivo cambia a true y por tanto su porcentaje de brillo se incrementa al 100%.

Como se ha explicado en el capítulo anterior, del mismo modo que se selecciona el dispositivo con el que quiere trabajar, se deberá deseleccionar para poder así poder seleccionar otros posteriormente, esta acción se lleva a cabo pulsando el botón ***Home***, que me cambiará el campo **seleccionado** del dispositivo en concreto a false.

Es **caso de estudio** es solamente un ejemplo de las ventajas y facilidades que esta aplicación puede llegar a proporcionar, pero como ya se ha comentado anteriormente, el código aquí desarrollado va a permitir el manejo de hasta 256 dispositivos que por otro lado es el máximo número de dispositivos permitidos en el protocolo X10.

CAPÍTULO 6. Conclusiones y trabajos futuros.

A lo largo de los meses de desarrollo del proyecto se han creado y aprendido numerosas técnicas hardware y de desarrollo de software, así como una forma poco común de comunicarse e interactuar con el ordenador, ya que anteriormente se requería tecnología difícil de obtener y de alto coste. En concreto hemos aprendido:

- ➔ Cómo usar correctamente el lenguaje de programación JAVA.
- ➔ Origen, funcionalidad y uso de la tecnología X10.
- ➔ Funcionalidades ofrecidas pos la consola *Wii* de Nintendo.
- ➔ Conocimiento de las diferentes librerías creadas para el manejo del ***WiiMote*** a través del PC.
- ➔ Uso de la librería *WiiUseJ*.
- ➔ Cómo conectar el uso del ***WiiMote*** con el hardware domótico creado.
- ➔ Cómo elaborar una documentación completa.

Como sabemos la consola *Wii* de Nintendo, así como su controlador ***WiiMote***, está experimentando un gran desarrollo debido fundamentalmente a la alta aceptación que está teniendo por parte del público, es por esto que cada vez se está haciendo menos costoso y más fácil acceder al hardware necesario (en lo que a esta parte se refiere) para hacer uso de la aplicación aquí desarrollada. Sin embargo, en lo que a la domótica se refiere, el coste del hardware necesario se incrementa con respecto al de la *Wii* puesto que es necesario un equipo hardware para que cada aparato (ya sea lámpara, electrodoméstico, persiana…) reciba la orden correspondiente a través de estos dispositivos domóticos.

Asimismo, hace pocas décadas era extraño encontrar un ordenador en un hogar, sin embargo hoy en día por numerosas razones entre las que destacan la distribución digital de la música, las películas y los videojuegos, se está imponiendo la existencia de ordenadores en los hogares y cada vez más también el uso de la tecnología domótica, ya que en muchos hogares se cuenta con un equipo informático en el salón que permite acceder a todo el contenido almacenado en el mismo, en otros equipos del hogar, o bien en Internet, así como el control automatizado de luces, persianas…gracias como digo a la tecnología domótica. Debido a que la utilización de un hardware y un ratón es bastante incómoda, se necesita nuevo hardware y software para el manejo de todas las posibilidades que ofrecen estos equipos. Por estos motivos ha sido desarrollado el software de este proyecto.

Además, debido a la poca movilidad que requiere este software puede ser usado por personas de movilidad reducida (o PMR) para manejar el encendido y apagado de luces, así como la atenuación únicamente pulsando un botón o con un simple movimiento del ***WiiMote***. Debido a su simplicidad también puede ser sencillo de usar para personas que no suelen usar un ordenador.

Como trabajos futuros lo que aquí se propone es hacer uso de la tecnología infrarroja de la que dispone el ***WiiMote***. Para ello se colocaría un led en cada dispositivo, de manera que apuntando con el mando a cada dispositivo, el software que se cree reconocerá el dispositivo y por tanto cada aparato podrá disponer del servicio de todos los eventos del mando, de manera que cada botón, conjunto de botones o movimientos tendrá una funcionalidad determinada de las que se podrá hacer uso cada dispositivo determinado. Del mismo modo, en lugar de colocar leds en los dispositivos también se pueden colocar sensores que detecten la presencia del mando y que del mismo modo el software reconozca el dispositivo en cuestión. De manera aún más avanzada también se pueden desarrollar algunas aplicaciones que empleando el ***WiiMote*** detecten el movimiento de los dedos y los gestos faciales y que actúen de forma diferente según éstos.

CAPÍTULO 7.
Bibliografía.

- Introducción

http://www.estrucplan.com.ar/articulos/verarticulo.asp?idarticulo=858

- Introducción domótica.

  http://www.cedom.es/que-es-domotica.php

- Qué es la domótica.

  http://www.arqhys.com/arquitectura/domotica-historia.html

- Historia de la domótica.

http://www.domoticausuarios.es/historia-de-la-domotica/1123/

- Evolución de la domótica en España.

https://sites.google.com/site/proyectodedomotica/1--troduccion/1-1-la-evolucion-de-la-domotica-en-espana

http://html.rincondelvago.com/domotica_4.html

- Tecnología X10.

http://es.wikipedia.org/wiki/X10

http://www.domotica.es/x10

- KNX/EIB

http://es.wikipedia.org/wiki/KNX

http://www.mundomotica.es/web1/knx1.htm

- Zigbee

http://es.wikipedia.org/wiki/ZigBee

- LonWorks

http://www.domotica.es/lonworks
http://upcommons.upc.edu/pfc/bitstream/2099.1/4300/1/Articulo%20PFC-%20Francesc%20Xavier%20Cano%20Palaz%C3%B3n.pdf

- Hardware domótica.

http://www.iit.upcomillas.es/pfc/resumenes/48c93805cb99e.pdf

http://www.dspace.espol.edu.ec/bitstream/123456789/10595/1/Dise%C3%B1o%20de%20un%20Sistema%20Dom%C3%B3tico%20aplicado%20a%20una%20cl%C3%ADnica%20de%20Hemodi%C3%A1lisis.pdf

- Wiimote.

http://es.wikipedia.org/wiki/Wiimote

- Accesorios Wii.

http://www.nintendo.es/NOE/es_ES/systems/accesorios_1243.html

- Wiimote y Bluetooth.

http://pegasus.javeriana.edu.co/~CIS0810TK02/Documentos/Memoria_Trabajo_de_Grado.pdf

- Eclipse

http://www.ctr.unican.es/asignaturas/programacion2/apuntes/seminarioEclipse.pdf

- Java.

http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

- Motej

http://motej.sourceforge.net/index.html

- WiiRemotej (java)

http://wiibrew.org/wiki/Wiimote/Library

- WiimoteLib

http://wiimotelib.codeplex.com/

- WiiYourself! (c++,Windows).

http://wiiyourself.gl.tter.org/

- Wiim (C++, Windows)

http://digitalretrograde.com/projects/wiim/

- Wiiuse (C, Windows/Linux)

http://wiibrew.org/wiki/Wiiuse

- CWiid (C,Linux)

http://abstrakraft.org/cwiid/wiki/libcwiid

- Active Home pro

http://www.cib.espol.edu.ec/Digipath/D_Tesis_PDF/D-36625.pdf

http://www.dspace.espol.edu.ec/bitstream/123456789/10595/1/Dise%C3%B1o%20de%20un%20Sistema%20Dom%C3%B3tico%20aplicado%20a%20una%20cl%C3%ADnica%20de%20Hemodi%C3%A1lisis.pdf

ANEXO 1. Conexión bluetooth *WiiMote* + PC.

La conexión entre el mando ***WiiMote*** y el PC empleando la tecnología bluetooth se puede realizar de dos maneras:

- **Utilizando *Bluesoleil*:** la cual es una aplicación dedicada a la gestión y conexión de dispositivos bluetooth. Como inconveniente presenta que es de pago y no es compatible 100% con todos los adaptadores *bluetooth* ya sean internos o de tipo *dongle*.

- **Empleando el gestor de *Windows 7*:** para utilizar este gestor no es preciso instalar ningún software adicional, además es compatible con un mayor número de adaptadores para conexiones bluetooth, ya sea el interno de algunos portátiles o bien un *dongle bluetooth* conectado normalmente por USB.

Para el uso de la aplicación aquí creada se pueden emplear ambas formas puesto que no presentan ningún problema, sin embargo, para el desarrollo del proyecto se ha empleado el **gestor de *Windows 7*** ya que, como hemos dicho presenta mayor compatibilidad con los adaptadores tipo *dongle bluetooth* que es lo que se ha usado en el proyecto.

A continuación se van a explicar ambas formas de conexión.

1. Utilizando *Bluesoleil*.

Para utilizar *Bluesolel* se deben seguir los siguientes pasos:

1. Instalar y adaptar la aplicación.

2. Emparejar ***WiiMote*** con *Bluesoleil*.

1.1. Instalar la aplicación.

Instalar *BlueSoleil* no es una tarea más compleja que instalar cualquier otra aplicación destinada a usuarios sin muchos conocimientos. El problema está en que no todos los adaptadores *bluetooth* son compatibles con él, por lo que se explicará una forma de conseguir su compatibilidad en la mayoría de los casos en los que surja dicho problema.

La siguiente figura muestra el comportamiento de *BlueSoleil* cuando éste no es capaz de encontrar nuestro adaptador *bluetooth*:

Figura Anexo 1.1. Problema de incompatibilidad *Bluesoleil*.

Para solucionarlo se deben realizar los siguientes pasos:

1. **Averiguar los valores VID** (Vendor ID o identificador del fabricante) y **PID** (Product ID o identificador del producto) **de nuestro adaptador bluetooth**: Para ello se debe ir al administrador de

dispositivos y buscar dentro de “dispositivos bluetooth” nuestro adaptador, pulsamos botón derecho del ratón sobre él y seleccionamos “propiedades”, luego la pestaña “detalles” y por último en el combobox seleccionamos “identificadores de hardware”. Debería aparecer algo similar a esto:

USB\Vid_050&Pid_0121&Rev_0100
USB\Vid_050&Pid_0121

Por lo que, en este caso, el VID es “050” y el PID es “0121”.

2. Editar el fichero *btcusb.inf*: localizado normalmente en la carpeta “C:\Archivos de programa\IVT Corporation\BlueSoleil\Driver\usb” salvo que se escogiera otra ruta diferente durante la instalación.

Ahora se debe buscar la sección identificada como “[ControlFlags]” y añadir una línea con la forma:

ExcludeFromSelect = USB\VID_050D&PID_0121

Sustituyendo el VID y PID por los que correspondan.

En el mismo fichero, buscar la sección “[IVT]” y encontrar alguna línea con el fabricante de nuestro adaptador. Por ejemplo si fuera “*Belkin*” habría que encontrar una línea así:

%Belkin.DeviceDesc%=BTusb_DDI, USB\VID_050D&PID_0081

Se copia y se pega debajo y se cambian los valores del *VID* y *PID* según corresponda.

3. Editar el fichero *nttl.ini*: localizado habitualmente en “C:\Archivos de programa\IVT Corporation\BlueSoleil” salvo que se cambiase la ruta por defecto de la instalación.

Ahora hay que buscar un bloque de código similar al mostrado a continuación cuyo fabricante indicado en la línea “Manufacture” sea el mismo que el de nuestro adaptador, copiarlo y pegarlo tras el último bloque de este tipo realizando las modificaciones oportunas de *VID* y *PID:*

[HW77]
ID=USB\VID_050d&PID_0084
Type=Bluetooth USB Dongle
DLL=Driver\USB\btcusb.dll
DLLD=Driver\USB\btcusbd.dll
Inffile=Driver\USB\btcusb.inf
Manufacture=BelKin

Por último solo queda modificar la primera línea del bloque “[HWxxx]” cambiando el número por el siguiente al del último bloque. Es decir, si el último bloque está numerado como “[HW110]”, el nuestro copiado justo debajo será “[HW111]”. También hay que buscar la línea del fichero indicada como “NUM=xxx” y aumentar en uno el número, haciéndolo coincidir con el que acabamos de modificar.

4. Reinstalar el *driver* de nuestro dispositivo *bluetooth*: realizar la instalación con normalidad, pero en el punto donde pregunte dónde está el *driver*, especificar manualmente la ruta eligiendo el fichero *btcusb.inf* anteriormente editado.

1.2. Emparejar *WiiMote* con *Bluesoleil*.

Una vez instalado correctamente *BlueSoleil*, se debe emparejar el *Wiimote* con el mismo. Para ello pinchamos con el botón derecho del ratón en el icono azul que representa el logotipo de *bluetooth* de la barra de tareas y elegimos "*Display classic view*". Aparecerá la siguiente ventana:

Figura Anexo 1.2. Ventana de Bluesoleil.

Para emparejar el *Wiimote* con el programa, se deben seguir los siguientes pasos:

1. **Mantener pulsados simultáneamente los botones *1* y *2* del *Wiimote*.**

2. **Hacer doble *click* sobre la bola amarilla de *BlueSoleil*:** Tras ello aparecerá el *Wiimote* representado como un *joystick.*

3. **Hacer doble *click* sobre el *Wiimote*:** aparecerán iluminados arriba los servicios que proporciona el dispositivo. En nuestro caso se iluminará el ratón, que indica que el *Wiimote* puede usarse como dispositivo de entrada.

4. **Hacer doble *click* sobre el icono del ratón:** A partir de ese momento, el *Wiimote* quedará emparejado con nuestro PC, listo para ser usado por nuestras aplicaciones.

El siguiente esquema muestra el proceso completo:

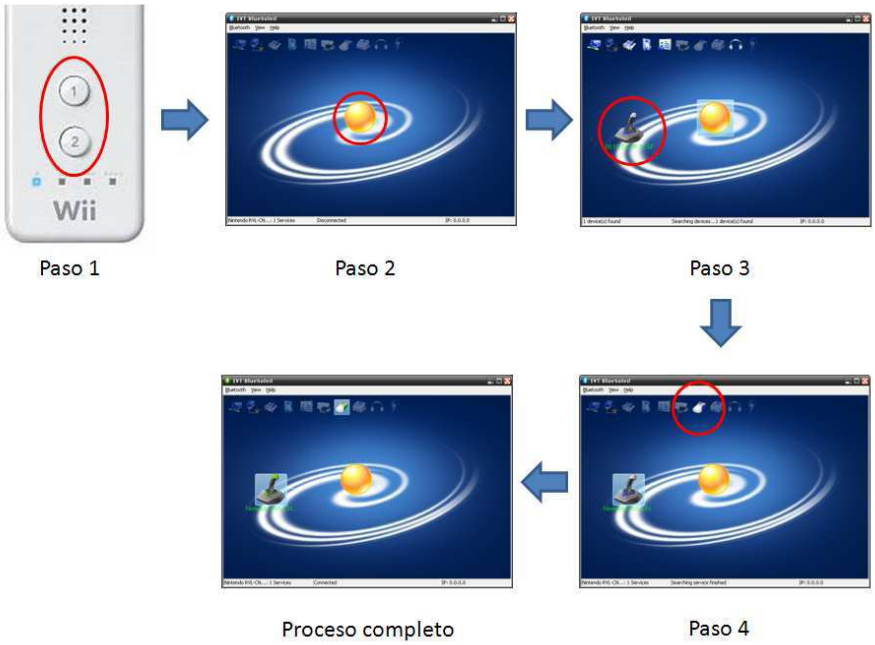


Figura Anexo 1.3. Proceso de emparejado de *WiiMote* con *Bluesoleil.*

2. Empleando el gestor de *Windows 7*.

El proceso de conexión del ***WiiMote*** con el gestor de conexiones *bluetooth* de *Windows 7* es muy sencillo, y a continuación se van a indicar los pasos a seguir:

1. En primer lugar, en este caso he trabajado con un *dongle bluetooth*, por lo que al conectar dicho aparato al PC aparecerá la siguiente ventana de información y en la parte derecha donde se muestran las notificaciones el siguiente icono.

Figura Anexo 1.4. Detección de *dongle bluetooth*.

Figura Anexo 1.5. Icono conexión *bluetooth.*

2. Una vez que se ha detectado el *dongle bluetooth* se pulsará clic derecho sobre el icono de la barra de tareas de *bluetooth* y se seleccionará la opción "*Agregar un dispositivo*". En algunas ocasiones es conveniente que antes de agregar el dispositivo se seleccione la opción "*Mostrar dispositivo bluetooth*" y eliminar en la ventana que nos aparece nuestro dispositivo y después volverlo a agregar, debido a que en

algunas ocasiones si ya se ha agregado con anterioridad el dispositivo, no se agregará correctamente y por tanto no se detectará bien la conexión.

3. Cuando ya se ha seleccionado la opción para agregar un nuevo dispositivo aparecerá una ventana en la cual aparecerán los dispositivos *bluetooth* encontrados, y para que se detecte nuestro ***WiiMote*** deberemos mantener pulsados al mismo tiempo los botones 1 y 2 del mismo, de esta forma aparecerá nuestro dispositivo como *"Nintendo RVL-CNT-01"*.

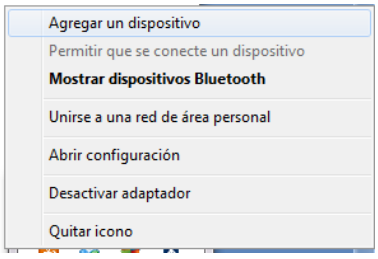
4. Una vez que aparece el dispositivo, lo seleccionamos y pulsamos *Siguiente* (sin dejar de pulsar los botones 1 y 2 del ***WiiMote***) y nos aparecerá otra ventana donde nos salen diferentes formas de emparejar el ***WiiMote*** con el PC, nosotros seleccionaremos *"Aparear sin usar código"* y pulsamos *Siguiente*.

5. A continuación nos aparecerá una ventana indicando que se está realizando la conexión y un globo de información en la barra de tareas donde pondrá *"Instalando controlador de dispositivo"*.

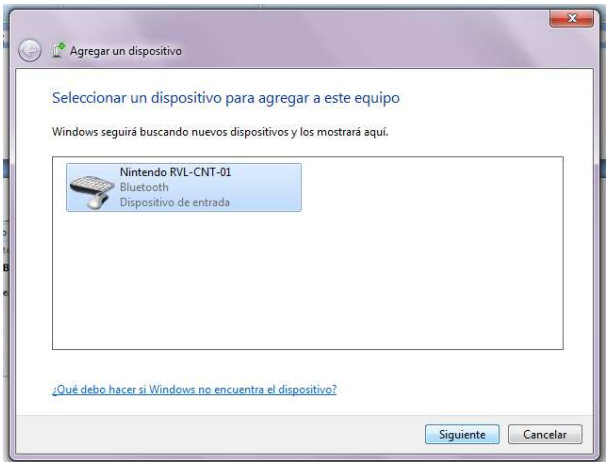
6. Finalmente cuando el dispositivo se haya agregado se indicará en una ventana y en un globo de información donde pondrá *"Dispositivo HID de bluetooth"Controlador de dispositivo instalado correctamente*.

Una vez que se han seguido todos estos pasos el dispositivo ya ha sido agregado y se puede emplear en la aplicación, cuando se ejecute el código de ésta el ***WiiMote*** se conectará sin problema, indicando dicha conexión una breve vibración del mando y la iluminación de uno de sus leds.

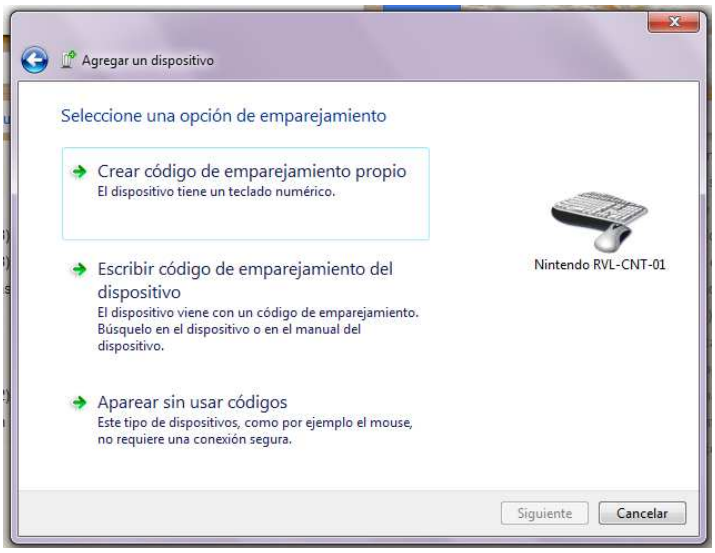
A continuación se muestra en un diagrama los pasos anteriormente explicados.

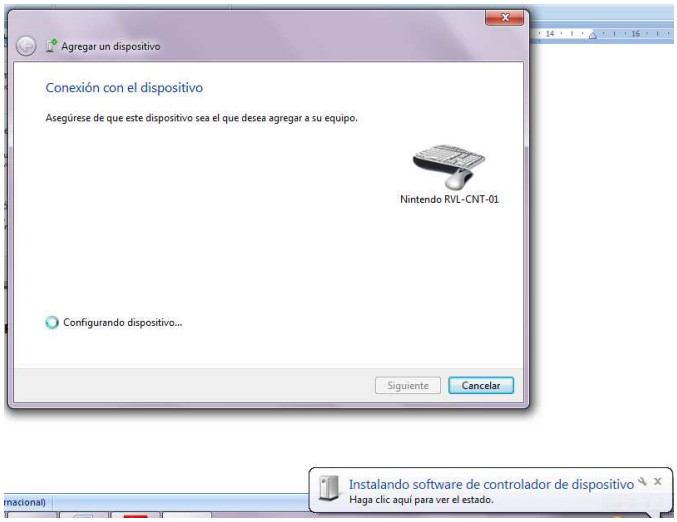


PASO 2



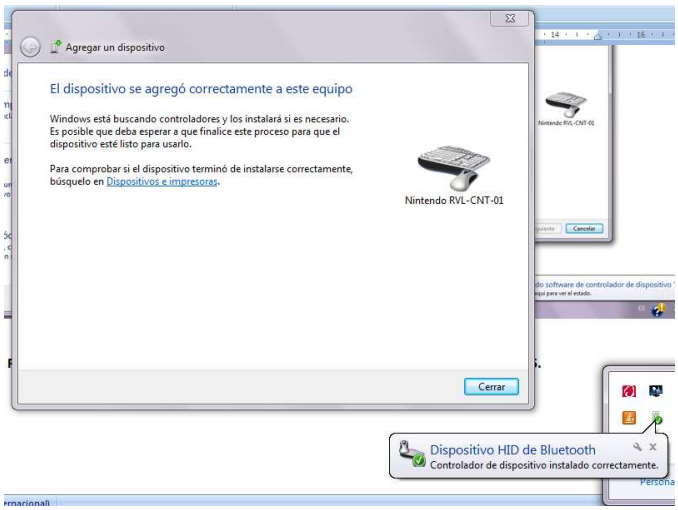
PASO 3.





PASO 4.

PASO 5.



PASO 6.

Figura Anexo 1.6. Diagrama de conexión empleando el gestor de conexiones *bluetooth* de *Windows 7*.

ANEXO 2.
Creación del proyecto y comienzos.

En este anexo se van a explicar los **pasos** que se han **seguido** para la **creación del software aquí desarrollado**, en el entorno de programación **Eclipse**.

1. Lo primero que tenemos que hacer es crear un nuevo proyecto en **Eclipse**. Pulsando File → New → Java Project, como muestra la siguiente figura. Como vemos a continuación nos aparece la ventana New Java Project donde introducimos el nombre del proyecto y pulsamos el botón Finish.

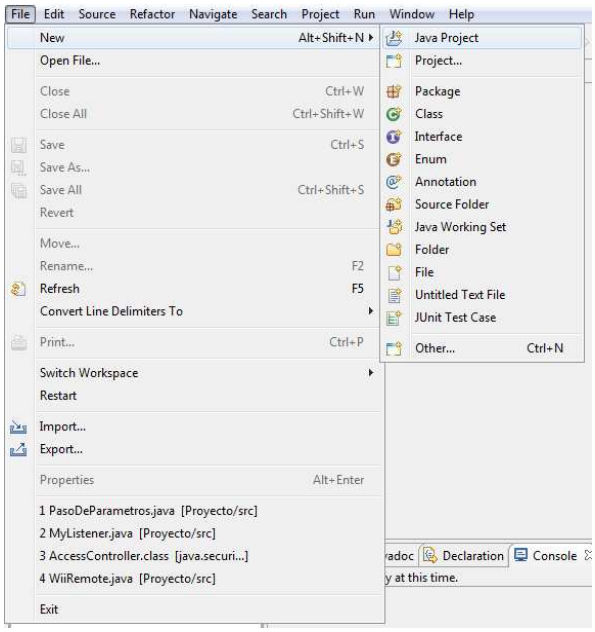


Figura Anexo 2.1. Creación de un proyecto en Eclipse [1].

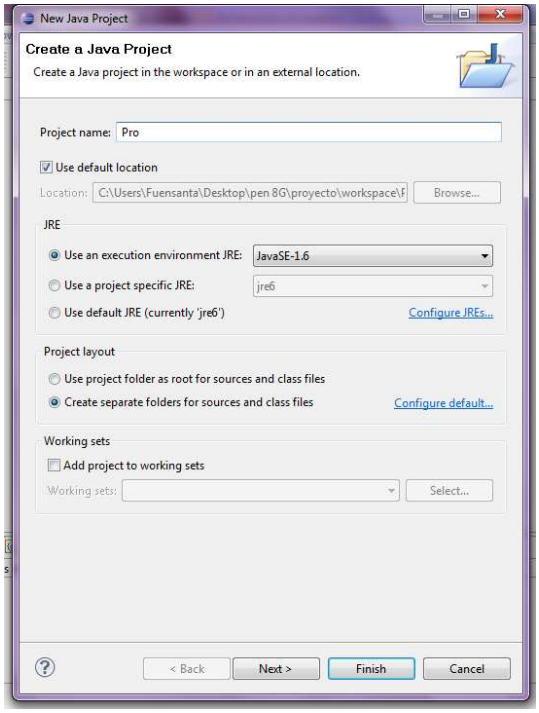


Figura Anexo 2.2. Creación de un proyecto en Eclipse [2].

Finalmente una vez que se crea el proyecto nos aparece a la izquierda una carpeta con el nombre del proyecto creado y en su interior un directorio llamado *src*, el cual es el directorio principal donde se crearán todas las clases. Dentro de la carpeta del proyecto también aparece un paquete llamado *JRE System Library* donde se añaden automáticamente todas las librerías .jar de la máquina virtual de java. Todo esto se muestra en la siguiente figura.

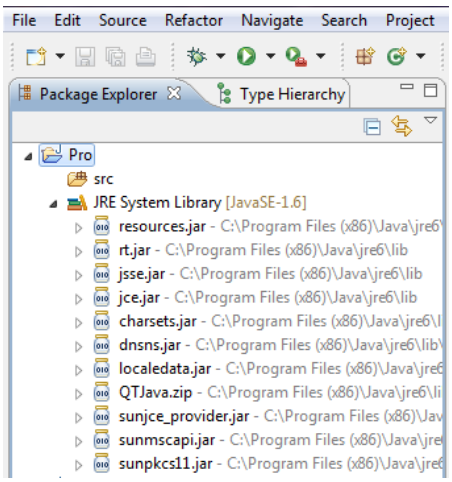


Figura Anexo 2.3. Carpeta del proyecto creada.

2. A continuación se deben copiar las librerías *wiiuse.dll* y *WiiUseJ.dll* en la ruta *Equipo → C: → Windows → System32* y se deben agregar al proyecto creado simplemente arrastrando ambas librerías a la carpeta creada en Eclipse.

3. Para terminar de agregar la librería para el manejo del ***WiiMote*** se deberá agregar el archivo *wiiusej.jar* de la siguiente manera.

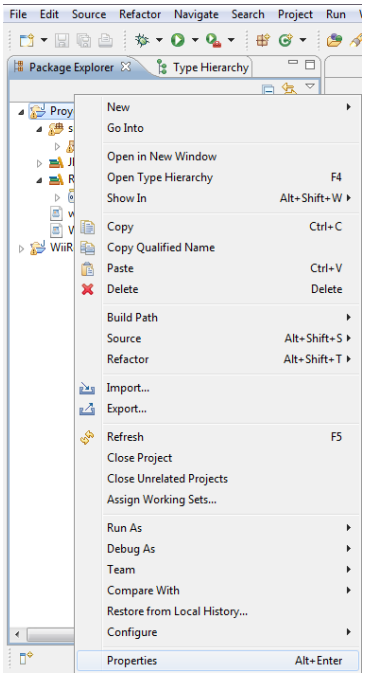


Figura Anexo 2.4. Agregar librerías [1].

Pulsamos clic derecho en la carpeta del proyecto y seleccionamos la opción *Properties*, de manera que nos aparecerá la siguiente ventana.

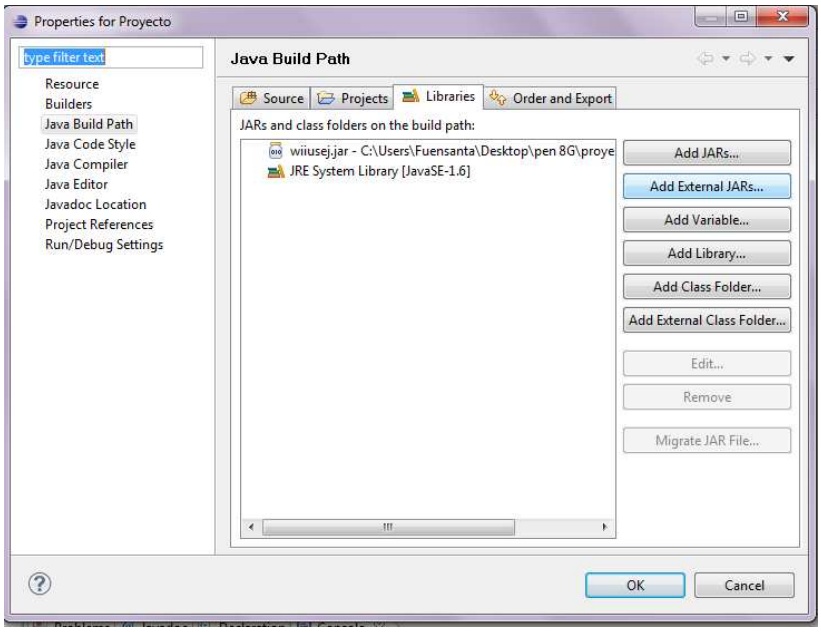


Figura Anexo 2.5. Agregar librerías [2].

Donde como vemos seleccionaremos el botón Add External JARs… y seleccionaremos de la carpeta donde se encuentra la librería *wiiuse.jar*, de manera que se agrega como vemos en la figura 100. Una vez que pulsamos el botón OK la librería queda agregada al proyecto, pudiendo hacer uso de todas sus clases.

4. Una vez que se han agregado al proyecto todas las librerías procederemos a crear las clases, que por defecto se crean en la carpeta *src* en el paquete por defecto. El código de cada una de ellas queda detallado en el **Anexo 6**.

ANEXO 3.

Puente de comunicaciones. *Ahcmd.exe.*

En este anexo se va a proceder a explicar la funcionalidad que proporciona el archivo *ahcmd.exe* el cual ha sido usado para el envío de comandos por consola a los aparatos domóticos. Este *puente de comunicaciones* proporciona el vínculo entre la interfaz y el PC, permitiendo así el envío y recepción de comandos. Permite un seguimiento del estado actual y ajusta los comandos en consecuencia.

El archivo del cual se va a detallar la funcionalidad en este apartado, es un archivo que se crea automáticamente cuando se instala el programa **Active Home Pro** y lo podemos encontrar en la siguiente ruta: *C: → Archivos de programa (x86) → Common Files → X10 → Common.* Aunque para poder usarlo en este proyecto se ha copiado este archivo a la ruta donde se encuentra nuestro proyecto.

Una vez que sabemos dónde podemos encontrar el archivo vamos a proceder a explicar cómo funciona y cómo lo hemos usado.

Existen dos métodos para pasar los datos al *puente de comunicaciones*:

1. Línea de comandos.
2. Envío de mensajes de Windows.

Nosotros emplearemos la línea de comandos, pero se explicará con detalle ambas formas de usar el *puente de comunicaciones.*

1. Línea de comandos.

El comando que se debe enviar siempre va a tomar la forma ***ahcmd <Command> [<delay>]***. A continuación se va a explicar que valores debe tomar cada parte.

Comandos aceptados.

Los comandos aceptados por la interfaz tienen típicamente la siguiente estructura:

<housecode and device code list> <function>, <delay>

Si se quiere enviar un comando a varios dispositivos se pondrá una lista de códigos de hogar + dispositivo delimitada por comas (siendo condición indispensable que los dispositivos de la lista tengan el mismo código de hogar). No debe haber ningún espacio después de la coma ni en el principio de la cadena. Después de la lista de códigos de casa + dispositivo (o sólo un código hogar + dispositivo si se envía la orden a un único dispositivo), debe haber un solo espacio seguido por la función y los datos adicionales asociados con la función. Las **funciones aceptadas** son las siguientes:

- All Units Off.
- All Lights On.
- All Lights Off.
- Hail Request.
- Hail Acknowledge.
- Status On.

- Status Off.

- Status Request.

Las siguientes funciones deben estar precedidas por un código de hogar + dispositivo:

- On.

- Off.

- Dim.

- Bright.

Estas dos últimas funciones son las encargadas de regular la intensidad de los dispositivos. Siempre están seguidas de 3 dígitos y un signo de porcentaje, por ejemplo si se quiere atenuar el nivel de brillo en un 50% del aparato con dirección B10, se tendrá que poner la siguiente sentencia:

B10 Dim 050%

Precedida, claro está, del comando *ahcmd*.

También existe la posibilidad de transmitir **código extendido**. Dicha transmisión se debe seguir de un espacio y luego los bytes de comando y los datos en formato hexadecimal, sin espacios, como por ejemplo:

C7 ExtendedCode 5A12

Con esta sentencia, se envía al dispositivo C7 el comando extendido 0x5A y 0x12 de datos. La lista del código de casa y dispositivo sólo debe abordar un único dispositivo, debido a la aplicación del protocolo.

Existen otros dos comandos adicionales que pueden ser transmitidos a la interfaz:

- Dim To. Es un pseudo comando X10 que depende del seguimiento de los comandos de brillo actuales del *puente de comunicaciones*. El software utilizará el conocimiento de los niveles actuales para atenuar u oscurecer las lámparas que se tratan en la lista de dispositivos al nivel final especificado en el comando, por ejemplo:

A2,A3,A9 Dim To 090%

Como vemos todos tienen la misma dirección del hogar. Si el nivel de iluminación es desconocido entonces se le da el 100% del brillo y luego se atenúa con el nivel requerido.

- Ring. Con este comando se especifica que suene el dispositivo cuando un comando X10 es recibido. No necesita código de hogar ni dispositivo.

Ring 1 → encendido.

Ring 2 → apagado.

Delays (Retardos).

Los comandos se pueden retrasar (utilizando el programador interno) haciendo que tengan efecto unos minutos después de que se haya transmitido la orden. Esto se consigue mediante la introducción de un tiempo de retardo después del comando, de la siguiente forma:

A3,A7 Dim to 080%, Delay 025

Para poner el tiempo de retard siempre si ponen 3 dígitos, en este caso ***025*** indica 25 minutos, por lo que este comando atenuará los dispositivos ***A3 y A7*** en un ***80%*** después de ***25*** minutos.

2. Envío de mensajes de Windows.

El *puente de comunicaciones* aceptará el mensaje **WM_COMMAND** de Windows con los siguientes parámetros y valores correspondientes:

Parámetro de palabra (WParam)	Valor Ordinal
IDC_REGISTER	300
IDC_UNREGISTER	301
IDC_TXDATA	302

A continuación se va a proceder a explicar para que sirve cada uno de estos parámetros de palabra.

IDC_REGISTER.

Este parámetro se envía con el parámetro LONG (LParam) y proporciona un puntero a la estructura de datos **REGMESSAGE**, la cual se define para pasar datos específicos de la aplicación entre las aplicaciones, y está definida por:

```
Typedef struct tagREGMESSAGE {

  HWDD hwnd;  //Identificador de la ventana para notificar.

  UINT uiMessage;  //mensaje a transmitir.

  WPARAM wParam;  //identificador de palabra para identificar de forma única el mensaje.

}REGMESSAGE;
```

Al recibir el mensaje IDC_REGISTER, el *puente de comunicaciones* añade los parámetros a su lista de notificación actual. Después al recibir las funciones X10 el estado actual del dispositivo se publica en las ventanas registradas. El parámetro LONG del mensaje enviado contendrá un puntero a una matriz de bytes que identifican el estado del dispositivo actual. Este puntero se debe copiar a la memoria local de inmediato, ya que es volátil. Esta matriz de bytes se divide en cuatro grupos de 256 bytes y un grupo de 14 bytes que detallan la siguiente información:

Grupo 1.

Nivel de brillo de la lámpara:

- ➔ 0 para apagado.
- ➔ 100 para el brillo máximo.
- ➔ 254 para un nivel de brillo sin definir.

Grupo 2.

El estado del aparato.

- ➔ 0 desactivado.
- ➔ 255 otros.

Grupo 3.

Número de comandos de “encendido” que se han recibido para el dispositivo indexado.

Grupo 4.

Número de comando se “apagado” que se han recibido para en dispositivo indexado.

Grupo 5.

14 bytes que detallan la información de dispositivos específica recibida de la interfaz.

Todos estos datos son expresados en una estructura **REGDATA**, la cual se define para pasar datos específicos del dispositivo entre las aplicaciones:

```
Typedef struct tagREGDATA{
  BYTE bStatus [4][256];
  BYTE bDeviceData [14];
}REGDATA;
```

Los datos del dispositivo se definen de la siguiente forma:

Rango de bits	Descripción.
111 to 96	Temporizador de la batería (se establece 0xfff para reset).
95 to 88	Hora actual en segundos.
87 to 80	Hora actual (minutos de 0 a 119).
79 to 72	Hora actual (horas/2, de 0 a 11).
71 to 63	Día del año.
62 to 56	

55 to 52	
51 to 48	Nivel de revisión del firmware de 0 a 15.
47 to 32	
31 to 16	On/Off de los dispositivos monitoreados.
15 to 0	Estado de atenuación de los dispositivos monitoreados.

Donde el bit 0 se define como el bit 0 del bDeviceData [13] y el bit 11 se define como el bit 7 del bDeviceData [0].

IDC_UNREGISTER.

Este mensaje debe ser enviado antes de que la aplicación termine. Si la aplicación termina antes de que el mensaje se reciba puede causar datos no válidos. El parámetro LONG debe contener un puntero a una estructura REGMESSAGE que define los parámetros de uso.

IDC_TXDATA.

Este mensaje se utiliza para enviar un comando por la línea de alimentación. El parámetro LONG del mensaje puede contener un puntero a un búfer que contiene el comando que va a ser enviado, o si la palabra alta es null, el parámetro puede contener un índice de comando para causar un comando de configuración para transmitir a la interfaz.

Índices de los comandos.

A continuación se muestra una lista de los comandos (y su funcionalidad) que pueden ser tratados a través del mensaje **IDC_TXDATA** para crear un mensaje de configuración que se enviará a la interfaz (en cada caso, los 8 primeros bits del parámetro LONG deben ser cero).

➔ Mostrar la ventana de configuración avanzada COM

X10_DRIVERSETTINGS 0X01.

➔ Mostrar la configuración básica de COM y la ventana de prueba.

X10_BASICCOMSETTINGS 0X02.

➔ Solicitar el estado actual del módulo X10 al driver de comunicación. Solicitar una actualización de estado desde el puente de comunicación.

X10_UPDATESTATUS 0X03.

➔ Provocar una macro para ser descargada a una interfaz CM10.

X10_NEWCM10MACRO 0X04.

➔ Cambiar el puerto COM (el byte alto de la palabra (bits 4 a 7) debe contener el identificador de puerto, el cual puede ser cualquier valor entre 1 y 255).

X10_CHANGECOMPORT 0X05.

➔ Descargar los temporizadores en el registro o archivo *.ini* a un enchufe múltiple de la interfaz CP10.

X10_WAKEUPTIMER 0X06.

➔ Descargar de la interfaz CP10 el código de dispositivo y hogar del registro (Windows 95) o archivo *.ini* (Windows 3.x).

X10_MODULECODE 0X07.

➔ Descargar los nuevos datos de la EEPROM a una interfaz CP10 o CM11.

X10_EEPROMDOWNLOAD 0X08.

➔ Solicitar al driver de comunicación el estado actual del dispositivo de la interfaz (CM11 ó CP10).

X10_REQUESTSTATUS 0X09.

➔ Estudiar la interfaz para obtener los temporizadores de uso corriente de la batería (CM11 ó CP10).

X10_BATTERYLEVEL 0X0a.

➔ Hace que la hora actual del sistema sea descargada a la interfaz y desactiva la máscara de la dirección actual, si el byte alto de la palabra (bits 4 a 7) se establecen.

X10_SETCLOCK 0X0b.

➔ Informar a la interfaz que se deben restablecer los contadores de la batería.

X10_CHANGEBATTERY 0X0c.

➔ Purgar todos los cronómetros de las macros pendientes.

X10_MACROTIMERPURGE 0X0d.

El *puente de comunicaciones* también acepta el mensaje **WM_COPYDATA**, el cual proporciona la misma funcionalidad que el mensaje **WM_COMMAND**. Los parámetros del **WM_COPYDATA** son los siguientes:

- wParam: identificador de la ventana que envía el mensaje.
- lParam: puntero a la estructura **COPYDATASTRUCT**.

Esta estructura se define para pasar datos entre aplicaciones (16 y 32 bits), y se define como:

Typedef struct tagCOPYDATASTRUCT{

DWORD dwData; //define la acción que debe ser tomada por el puente de comunicación.

DWORD cbData; //define cualquier dato adicional que sea requerido.

LPVOID lpData; //define cualquier dato adicional que sea requerido.

}COPYDATASTRUCT;

A continuación se definen los posibles valores que puede tomar el campo *dwData*:

➔ CDM_REGISTER 0x0001.

➔ CDM_UNREGISTER 0X0002.

Estos dos valores se utilizan como mensajes para registrar o borrar la aplicación cliente para recibir actualizaciones de estado cada vez que se detecta un cambio de estado. Los parámetros cbData y lbData suelen ser cero.

➔ CDM_TRANSMIT 0X0003.

Utilizado para iniciar una transmisión X10. El cbData debe indicar la longitud de la cadena de datos que identifica la transmisión y la lpData contiene un puntero a dicha cadena de datos.

➔ CDM_STATUSREQUEST 0X0004.

Enviado al *puente de comunicaciones* para solicitar el estado actual del dispositivo (con cbData y lpData igual a 0). El *puente de comunicaciones* a continuación devolverá los datos de estado a través de un mensaje WM_COPYDATA con los miembros de la estructura se la siguiente manera:

wParam → identificador de ventana del *puente de comunicaciones*.

lParam → puntero a la estructura **WM_COPYDATA**.

La estructura **WM_COPYDATA** contendrá:

- *dwData → COM_STATUS_REQUEST.*
- *cbData → tamaño de la estructura REGDATA.*
- *lpData → puntero a la estructura REGDATA.*

Los siguientes comandos tienen la misma funcionalidad que para el mensaje **IDC_TXDATA**.

➔ CDM_DRIVERSETTINGS 0X0005.

➔ CDM_BASICCOMSETTING 0X0006.

➔ CDM_NEWCM10MACRO 0X0007.

➔ CDM_CHANGECOMPORT 0X0008.

➔ CDM_WAKEUPTIMER 0X0009.

➔ CDM_MODULECODE 0X000a.

➔ CDM_EEPROMDOWNLOAD 0X000b.

➔ CDM_REQUESTSTATUS 0X000c.

➔ CDM_BATTERYLEVEL 0X000d.

➔ CDM_SETCLOCK 0X000e.

➔ CDM_CHANGEBATTERY 0X000f.

➔ CDM_MACROTIMERPURGE 0X0010.

ANEXO 4. Librería *WiiUseJ*.

4.1. Descripción de la librería *WiiUseJ*.

WiiUseJ es una API java para el acceso al periférico ***WiiMote*** de la consola *Wii* de Nintendo. Como ya se ha explicado en el capítulo 3, se basa en una API C llamada *WiiUse* que accede directamente al *bluetooth* stack de nuestro sistema. La diferencia de *WiiUseJ* con otras APIs Java reside en que no se basa en el paquete javax.bluetooth (implementación del estándar JSR-82), con lo cual resulta ser un mecanismo más eficiente para acceder al mando al estar implementada sobre un lenguaje de bajo nivel como es C.

A continuación se detallan las interfaces y clases más significativas de esta API:

4.1.1. Interfaz *WiimoteListener*.

Esta es la interfaz a implementar para escuchar los eventos que ocurren con el ***WiiMote***. Los métodos más significativos se explican a continuación:

onButtonsEvent

void onButtonsEvent (WiimoteButtonsEvent e): método que es llamado cuando se produce un evento de botón.

Parámetros: e - El ButtonEvent con los últimos datos acerca de los botones del ***WiiMote***.

onIrEvent

void onIrEvent (IREvent e): método que es llamado cuando se produce un evento de infrarrojos.

Parámetros: e – El IREvent con los puntos de vista IR.

onMotionSensing Event

void onMotionSensingEvent (MotionSensingEvent e): Método que es llamado cuando se produce un evento de detección de movimiento.

Parámetros: e – El sensor de movimiento con la orientación y la aceleración.

onExpansionEvent

void onExpansionEvent (ExpansionEvent e): Método que es llamado cuando se produce un evento de expansión.

Parámetros: e – El caso de la expansión que se produjo.

onStatusEvent

void onStatusEvent (StatusEvetn e): Método que es llamado cuando se produce un evento de estado. Un evento de estado se produce cuando preguntamos si “un controlador de expansión ha sido conectado” o

"un controlador de expansión ha sido desconectado". Aquí es donde se pueden obtener los diferentes valores de la configuración de los parámetros del ***WiiMote***.

Parámetros: e – El evento de estado.

onDisconnectionEvent

void onDisconnectionEvent (DisconnectionEvent e): Método que es llamado cuando se produce un evento de desconexión. Un evento de desconexión ocurre cuando el ***WiiMote*** ha sido apagado o la conexión se interrumpe.

Parámetros: e – El caso de la desconexión.

onNunchukInsertedEvent

void onNunchukInsertedEvent (NunchukInsertedEvent e): Método que es llamado cuando se produce la conexión de un Nunchuk.

Parámetros: e – El evento de la conexión del Nunchuk.

onNunchukRemovedEvent

void onNunchukRemovedEvent (NunchukRemovedEvent e): Método que es llamado cuando se produce la desconexión de un Nunchuk.

Parámetros: e – El evento de la desconexión del Nunchuk.

onGuitarHeroInsertedEvent

void onGuitarHeroInsertedEvent (GuitarHeroInsertedEvent e): Método que es llamado cuando se produce la conexión del Guitar Hero.

Parámetros: e – El evento de la conexión del Guitar Hero.

onGuitarHeroRemovedEvent

void onGuitarHeroRemovedEvent (GuitarHeroRemovedEvent e): Método que es llamado cuando se produce la desconexión del Guitar Hero.

Parámetros: e – El evento de la desconexión del Guitar Hero.

onClassicControllerInsertedEvent

void onClassicControllerInsertedEvent (ClassicControllerInsertedEvent e): Método que es llamado cuando se produce la conexión del Classic Controller.

Parámetros: e – El evento de la conexión del Classic Controller.

onClassicControllerRemovedEvent

void onClassicRemovedEvent (ClassicControllerRemovedEvent e): Método que es llamado cuando se produce la desconexión del Classic Controller.

Parámetros: e – El evento de la desconexión del Classic Controller.

4.1.2. Clase *WiiuseApi*.

Clase para manipular la API *WiiUseJ*. Entre sus métodos, destacan:

Continuous

void activateContinuous (int id): Hace que el ***WiiMote*** genere un evento cada vez.

void deactivateContinuous (int id): Desactiva el método anterior.

IRTracking

void activateIRTracking (int id): Activa el puerto IR de seguimiento en el ***WiiMote*** pasándole un identificador.

void deactivateIRTracking (int id): Desactiva el puerto IR de seguimiento en el ***WiiMote*** pasándole un identificador.

MotionSensing

void activateMotionSensing (int id): Activa el sensor de movimiento del ***WiiMote*** pasándole un identificador.

Void deactivateMotionSensing (int id): Desactiva el sensor de movimiento del ***WiiMote*** pasándole un identificador.

Rumble

void activateRumble (int id): Activa la vibración sobre el ***WiiMote*** pasándole un identificador.

void deactivateRumble (int id): Desactiva la vibración sobre el ***WiiMote*** pasándole un identificador.

Smoothing

void activateSmoothing (int id): Hace que los acelerómetros den resultados más suaves.

void deactivateSmoothing (int id): Desactiva el método anterior.

setLeds

void setLeds (int id, Boolean led1, Boolean led2, Boolean led3, Boolean led4): Establece el estado de los leds del ***WiiMote***.

claseConnection

void closeConnection (int id): Cierra la conexión del ***WiiMote*** pasándole un identificador.

4.1.3. Interfaz *WiiUseApiListener.*

Esta es la interfaz a implementar para escuchar los eventos de la *WiiUse* API.

4.1.4. Clase *WiiUseApiMAnager*

Clase que maneja el uso de la librería API *WiiUseJ*.

4.2. Documentación API *WiiuseJ*.

Packages	
1	**Wiiusej**
2	**Wiiusej.test**
3	**Wiiusej.utils**
4	**Wiiusej.values**
5	**Wiiusej.wiiusejevents**
6	**Wiiusej.wiiusejevents.physicalevents**

7	Wiiusej.wiiusejevents.utils
8	Wiiusej.wiiusejevents.wiiuseapievents

4.2.1. Package wiiusej.

Class Summary	
Wiimote	Class that represents a wiimote.
WiiUseApi	Songleton used to manipúlate WiiUse Api.
WiiUseApiManager	Class that manages the use of Wiiuse API.

4.2.2. Package wiiusej.test.

Class Summary	
ClassicControllerGuiTest	This frame is used to display events from a classic controller.
CloseGuiTestCleanly	This class is used to close wiiusej cleanly.
GuitarHero3GuiTest	This frame is used to display events from a Guitar Hero 3.
Main	Main Clas to launch WiiuseJ GUI Test.
NunchukGuiTest	This frame is used to display events from a nunchuk.
Tests	This class used to test WiiuseJ in text mode.
WiiuseJGuiTest	Gui class to test WiiuseJ.

4.2.3. Package wiiusej.utils.

Class Summary	
AccelerationExpansionEventPanel	Panel to display Acceleration in a MotionSensingEvent from an expansion.
AccelerationPanel	This panel is used to watch raw acceleration values from a MotionSensingEvent.
AccelerationWiimoteEventPAnel	Panel to display Acceleration in a MotionSensingEvent from

	a wiimote.
ButtonsEventPAnel	This panel is used to see what buttons are pressed on the wiimote.
ClassicControllerButtonsEventPanel	This panel is used to display what happens on the classic controller.
GForceExpansionEventPanel	Panel to display GForce in a MotionSensingEvent from an expansion.
GForcePAnel	This panel is used to watch gravity force values from a MotionSensingEvent.
GForceWiimoteEventPanel	Panel to display GForce in a MotionSensignEvent from a wiimote.
GuitarHero3ButtonsEventPanel	This panel is used to display what happens on the buttons of the Guitar Hero 3 controller.
GuitarHeroJoystickEventPanel	Panel to display Guitar Hero 3 controller joystick events.
IRPanel	This panel is used to see what the IR camera of the wiimote sees.
JoystickEventPanel	Panel to display joystick events.
NunchukJoystickEventPanel	Panel to display nunchuk joystick events.
OrientationExpansionEventPanel	Panel to display Orientation in a MotionSensingEvent from an expansion.
OrientationPanel	This panel is used to watch orientation values from a MotionSensignEvent.
OrientationWiimoteEventPanel	Panel to display Orientation in a MotionSensignEvent from a wiimote.

4.2.4. Package wiiusej.values.

Class Summary	
GForce	Represents gravity force on each axis.
IRSource	Class used for IR spurces.
Orientation	Class that represents the orientation of the wiimote.
RawAcceleration	Represents raw acceleration on each axis.

4.2.5. Package wiiusej.wiiusejevents.

Class Summary	
GenericEvent	Abstract mother class representing an event with a wiimote id.

4.2.6. Package wiiusej.wiiusejevents.physicalevents.

Class Summary	
ButtonsEvent	Class which represents a buttons event.
ClassicControllerButtonsEvent	Class which represents a buttons event from a Classic controller.
ClassicControllerEvent	This class represents the values from the classic controller and its events.
ExpansionEvent	Mother Class of all expansion event.
GuitarHeroButtonsEvent	Class which represents a buttons event from a Guitar Hero controller.
GuitarHeroEvent	This class represents the values from the GuitarHero and its events.
IREvent	Class which represents IR event.
JoystickEvent	Class that stores values on a joystick Event.
MotionSensingEvent	Class which represents a motion sensing event.
NunchukButtonsEvent	Class which represents a buttons event from a Nunchuk.
NunchukEvent	This class represents the values from the joystick and its events.
WiimoteButtonsEvent	Class which represents a buttons event for a generic event.

4.2.7. Package wiiusej.wiiusejevents.utils.

Interface Summary	
WiimoteListener	This is the interface to implement to listen to events from wiimotes.
WiiUseApiListener	This is the interface to implements to listen to events from the wiiuse API.

Class Summary	
EventsGatherer	This class is used to gather events during a call to the Wiiuse API.

4.2.8. Package wiiusej.wiiusejevents.wiiuseapievents.

Class Summary	
ClassicControllerInsertedEvent	Event that represents the connection of a classic controller to a wiimote.
ClassicControllerRemovedEvent	Event that represents the disconnection of a classic controller from a wiimote.
DisconnectionEvent	Class representing a disconnection event.
GuitarHeroInsertedEvent	Event that represents the connection of a Guitar hero controller to a wiimote.
GuitarHeroRemovedEvent	Event that represents the disconnection of a guitar hero from a wiimote.
NunchukInsertedEvent	Event that represents the connection of a nunchuk from a wiimote.
NunchukRemovedEvent	Event that represents the disconnection of a nunchuk from a wiimote.
StatusEvent	Class used to represent a status event.
WiimoteEvent	Class that is a bean to be filled by the wiiuse API on an event that occurs on a wiimote.
WiiUseApiEvent	This class describes the structure of an event from the WiiUse API event.

ANEXO 5.
Configuración y uso de Active Home Pro.

En este anexo se va a dar una explicación de cómo instalar la red domótica y cómo configurar y hacer uso del software **Active Home Pro**, puesto que será necesario conocer su funcionamiento para hacer uso de la aplicación creada.

5.1. Instalación del hardware básico

Antes de la ejecución de la aplicación **Active Home Pro**, los dispositivos X10 deberán ser instalados físicamente en la red eléctrica.

Todos los dispositivos con el estándar X10 son muy fáciles de instalar por lo que son considerados dispositivos plug and play, debido a que su utilización dependerá de la característica del diseño del equipo y del punto de la red eléctrica a donde será conectado.

En el capítulo anterior se han explicado los tipos de dispositivos de los que debe constar una red domótica X10 pero es en este apartado donde se explica más detalladamente cómo funcionan estos dispositivos y cómo llevar a cabo su conexión.

5.1.1. Programador PC. Controlador CM11A.

Como ya he mencionado anteriormente la interfaz **CM11A** es la utilizada en este proyecto, a pesar de usar el software **Active Home Pro**, creado para dar funcionalidad a la interfaz CM15.

Como ya hemos dicho este controlador se conecta a la red eléctrica (de esta forma es como se envían las órdenes a los módulos) y también consta de un cable RJ11-RS232 para la conexión del dispositivo con el PC, aunque en este proyecto se ha empleado un adaptador RS232-USB para esta conexión. La forma de llevar a cabo esta conexión queda detallada a continuación.

Figura Anexo 5.1. Controlador CM11A.

1. En primer lugar se conecta el adaptador RS232-USB al puerto USB del PC.

2. Después se conecta el puerto RS232 del cable que va al CM11A al puerto RS232 del adaptador.

3. Finalmente se conecta el conector RJ11 al puerto RJ11 del dispositivo controlador

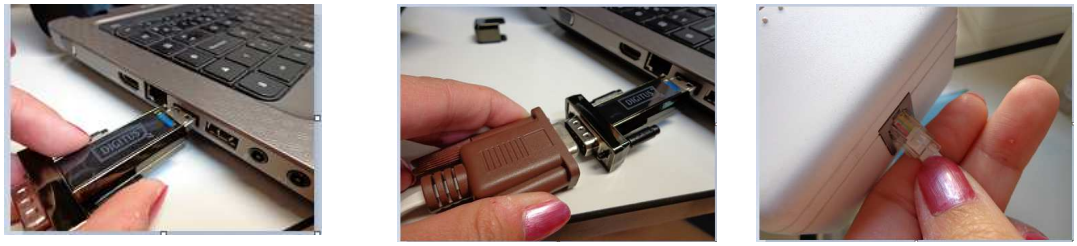


Figura Anexo 5.2. Esquema de conexionado CM11A.

El único problema que puede presentar esta conexión es que el PC no reconozca el adaptador, lo cual se soluciona instalando los drivers del mismo, los cuales vienen al comprar el dispositivo, de manera que cuando esto ocurra automáticamente se le asigna un puerto COM que es el que posteriormente tendremos que seleccionar en el software **Active Home Pro**, en *Herramientas → Configuración CM11A*, para que de este modo el software quede conectado a la red creada.

5.1.2. Módulos X10.

Cualquier módulo X10 se configura asignándole un código de casa y un código de dispositivo.

Figura Anexo 5.3. Módulo X10. Ruletas Cod. Dispositivo y cod. Hogar.

Los equipos X10 poseen dos ruedas las cuales son utilizadas para la configuración del código del módulo correspondiente en la red eléctrica. La segunda, la de color rojo representa el **código de hogar** que es el que asigna el módulo a un grupo de control y está identificada con las letras de la A a la P, y la primera, en color negro representa el **número del módulo** que corresponde a dicho dispositivo. Se pueden realizar todas las combinaciones posibles entre las dos ruedas para identificar los equipos, pudiendo obtenerse así hasta 256 direcciones distintas que es el máximo número de dispositivos diferenciados que compone un sistema domótico X10. Si dos actuadores tienen los mismos códigos de casa y numérico, ejecutarán simultáneamente las órdenes procedentes de la red eléctrica.

En el capítulo anterior se ha dado una clasificación de los actuadores atendiendo a su funcionalidad, sin embargo también podemos clasificar estos actuadores en función del tipo de conexión:

- **De pared.**
- **De casquillo.**
- **De carril DIN.**
- **Pulsadores empotrables.**
- **Módulos de cable.**

A continuación se explicará con detalle la funcionalidad y la forma de conectar cada uno de estos actuadores.

5.1.2.1. De pared.

La ventaja principal de este tipo de dispositivos actuador X-10 es que para su conexión, no es necesario ningún tipo de cableado, instalación u obra adicional, ya que se conectan en cualquier toma de enchufe estándar de pared.

Figura Anexo 5.4. Módulo X10 de pared.

Su misión es detectar una instrucción X10 que circule por la instalación eléctrica, y en caso de que esta vaya dirigida hacia él, actuar en consecuencia conectando o apagando el aparato eléctrico que se encuentra enchufado al mismo, y siempre que este aparato no tenga su propio interruptor desconectado.

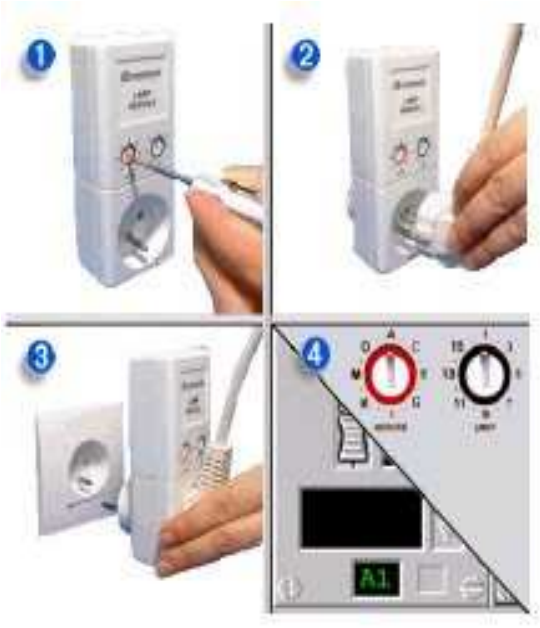


Figura Anexo 5.5. Ilustración conexión módulo X10 de pared.

La forma de conectarlos es la que se muestra en la figura anterior:

1. En primer lugar se asigna su dirección X10 empleando las ruedas de código de hogar (A a la P) y dispositivo (1 a 16).

2. Después hay que conectarle el aparato eléctrico que se quiera controlar con su propio interruptor en posición de encendido.

3. En tercer lugar, se enchufa a la red eléctrica.

4. Hecho esto, el dispositivo ya está listo para responder a cualquier instrucción X10 que llegue al mismo a través de la red eléctrica y que incluya su misma dirección casa y aparato. En la imagen de la derecha se muestra como se ha configurado un dispositivo virtual en la aplicación Active Home para generar una instrucción X10 mediante un ordenador.

Existen diferentes tipos de módulos de pared X10 en función de la carga que puede conectárseles, y no sólo de la potencia que pueden conmutar, sino el tipo de la misma (inductiva, resistiva, halógena,…). Algunos de estos tipos se muestran a continuación:

- **De Lámpara.** Admite funciones de ON y OFF y de atenuación (DIMMER) de lámparas de incandescencia desde 40 W hasta 300 W. Responde también a las instrucciones “ALL LIGHTS ON” y “ALL LIGHTS OFF” que encienden y apagan respectivamente todas las luces.

- **De aparato.** Admiten funciones de ON y OFF de aparato de hasta 3500 W. Admiten un máximo de 500 W para lámparas fluorescentes, y corrientes máximas de 1 A para motores y cargas inductivas y 16 A para el resto de cargas. No admiten funciones de atenuación (DIMMER).

5.1.2.2. De casquillo.

Al igual que los módulos de pared, este tipo de dispositivo actuador X10 tiene una conexión muy simple, sin necesidad de cableado, ni de obra, ni de instalación previa.

Se puede utilizar solo con bombillas incandescentes de hasta 60 W de potencia en lámparas cerradas, y 100 W en lámparas abiertas, admite instrucciones X10 de encendido (ON), apagado (OFF), encendido de todas las luces (ALL LIGHTS ON) y apagado de todas las luces (ALL LIGHTS OFF), no permite funciones de regulación.

Figura Anexo 5.6. Módulo X10 de casquillo.

Su programación es diferente de los anteriores dispositivos, en este caso deberemos proceder de la siguiente forma:

1. Desconectar la corriente. Retirar la bombilla del casquillo e insertar el módulo de casquillo en la lámpara.

2. Insertar la bombilla en el casquillo.

3. Restablecer la corriente. La lámpara no se encenderá.

4. Con cualquier controlador X10 (por ejemplo un mando a distancia) con el código de casa del casquillo, presionar 3 veces seguidas, en intervalos de 1 segundo, el código unidad deseado para la bombilla, antes de que pasen 30 segundos desde que se restablezca la corriente. A la tercera vez que se pulse el código unidad, la lámpara se encenderá y el código quedará almacenado en la misma.

5. Para volver a cambiar el código, apagar la lámpara, desconectarla de la corriente, conectarla de nuevo y volver al punto 4.

5.1.2.3. De carril DIN.

Su programación y funcionamiento son similares a los de los módulos de pared, pero, a diferencia de estos, necesitan cableado adicional desde su ubicación en los carriles DIN hasta las cargas que se desean controlar, aunque en la mayoría de los casos estos cables ya se encuentran disponibles en la propia caja de conexiones donde se encuentra el carril DIN junto con otros dispositivos habituales en estas cajas como los elementos de protección (diferenciales, magnetotérmico, …).

Figura Anexo 5.7. Módulo X10 de carril DIN.

Estos dispositivos se utilizan cuando se quieren controlar todos los dispositivos de una sala, o de una parte de la casa, o de un tipo (luces, enchufes…) ya que en este caso el cableado estaría disponible en la propia caja de conexiones.

La forma de ponerlo en operación se ilustra en las siguientes imágenes, donde se describe la conexión de un modulo de aparato carril DIN:

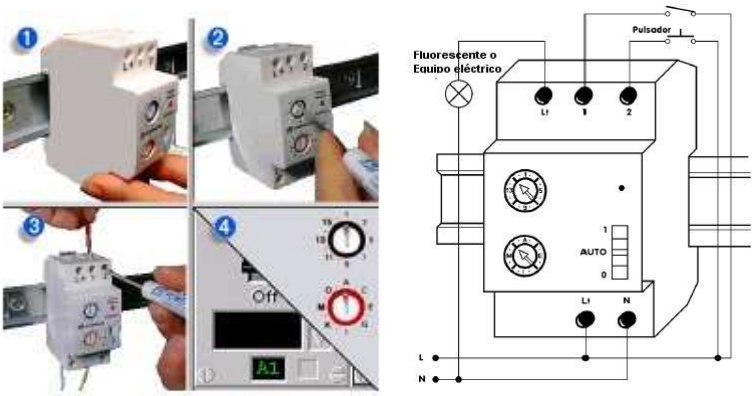


Figura Anexo 5.8. Conexión de un módulo de aparato de carril DIN.

1. Desconectar la corriente. Para ello operar sobre el magnetotérmico correspondiente. Montar el módulo sobre el carril DIN.

2. Mediante un destornillador asignar la dirección casa y aparato al módulo, haciendo girar las dos ruedas hasta la posición deseada.

3. Conectar los cables de fase al terminal L ↑, el neutro a N y la salida de la carga al terminal L ↓. El terminal 1 del módulo está diseñado para utilizar interruptores de pared convencionales, de forma que se activa el relé del modulo de carril DIN en función de que estos interruptores estén o no pulsados. El terminal 2 está diseñado para utilizar pulsadores, de forma que cada vez que pulsan, el relé cambia de estado (al pulsar la primera vez se activa, la segunda se desactiva, la siguiente se activa, etc.).

Una vez conectado se volverá a dar tensión, rearmando el diferencial o le magnetotérmico y se comprobara el correcto funcionamiento del módulo de carril DIN pulsando el botón de test de su parte formal. Si este botón se coloca en posición 1, la carga conectada seguirá encendida y no se podrá apagar ni siquiera con los pulsadores de pared. Si se conectan en posición 0, la carga permanecerá apagada permanentemente. En posición AUTO serán las señales X10 o la de los pulsadores conectados los que activen o desactiven la carga.

4. Como el resto de controladores X10, se puede crear un componente virtual en el ordenador utilizando la aplicación **Active Home** ó **Active Home Pro**, o cualquier otro software compatible. De esta forma, se podrá activar o desactivar el módulo desde el PC.

Existen varios tipos diferentes de módulos de carril X10, en función de la carga que pueden conectárseles, y no solo de la potencia que puede soportar, sino del tipo de la misma (inductiva, resistiva, Halógena,…). La forma de conexión varia ligeramente de uno a otros. La anteriormente descrita corresponde a un modulo de aparato. A continuación se muestran las características de algunos tipos:

- **De lámpara.** Diseñado para lámparas de 220 VAC o luces halógenas de 12 V. Admiten funciones ON/OFF y de atenuación (DIMMER) de luces incandescentes desde 40 W hasta 700 W. Responde

también a las instrucciones “ALL LIGHTS ON” y “ALL LIGHTS OFF” que encienden y apagan respectivamente todas las luces.

- **De aparato.** Admiten función de ON/OFF de cargas con un máximo de 2000 W para lámparas fluorescentes, corrientes máximas de 3 A para motores y de 16 A para cargas resistivas. No admiten funciones de atenuación (DIMMER).

- **De persiana.** Ocupa 4 espacios en el carril DIN. Carga máxima de 500 VAC, motores de 220 V, 10 A.

5.1.2.4. Pulsadores empotrables.

Estos dispositivos reemplazan a los interruptores y pulsadores convencionales para el control de luces y aparatos.

Las cargas conectadas se activan o desactivan por dos razones: porque se accione el pulsador incluido en el dispositivo (también puede añadirse otro pulsador externo) o porque reciban una señal X10 que coincida con la que tiene que configurar. Los dos tipos de pulsadores existentes son:

- **De lámpara.** Admiten funciones de ON/OFF y de atenuación (DIMMER) de luces incandescentes desde 60 W a 500 W. Responden también a las instrucciones “ALL LIGHTS ON” y “ALL LIGHTS OFF” que encienden y apagan, respectivamente, todas las luces.

- **De aparato.** Admiten funciones de ON/OFF de cargas con un máximo de 2000 W para cargas resistivas. No admiten funciones de atenuación (DIMMER).

Para configurarle la dirección, hay que quitar el pulsador, y aparecerán a la vista las dos ruedas con la dirección casa y aparato (1). Hay que conectar las tomas de tensión, y conectar el dispositivo a la carga (luces, aparatos...) (2).
Se vuelve a colocar la tapadera del pulsador (3) y ya está listo para recibir señales X10 desde cualquier fuente, por ejemplo, desde un PC (por ejemplo con **Active Home Pro**, en la figura 4).
Por supuesto, estas acciones se efectuarán sin tensión en la red, que deberá cortarse accionando el magnetotérmico correspondiente.

Figura Anexo 5.9. Conexión de pulsadores empotrables.

5.1.2.5. Módulos de cable.

Los módulos de cable son dispositivos X10, con la misma función que los de pared, que están diseñados para su instalación en falsos techos o en cajas universales. Los módulos deben conectarse con las cargas mediante cables. Así mismo, se alimentan de la red mediante una conexión que debe realizarse también con cables. A través de esta conexión cualquier información codificada en el formato X10 accede al

modulo, accionándolo en el caso de que la misma contenga los códigos de casa y aparatos con los que se ha configurado previamente el modulo.

5.2. Configuración del software básico de la aplicación Active Home Pro.

Después de haber instalado todos los módulos necesarios y la aplicación **Active Home Pro**, se creará un menú dentro de la carpeta programas de Windows, el cual nos servirá de ejecutable para la utilización del programa.

El entorno principal divide la casa en distintas habitaciones, donde se van colocando los componentes X10, como se observa en la siguiente figura.

Figura Anexo 5.10. Entorno de trabajo de Active Home Pro.

5.2.1. Crear y editar habitaciones.

Cuando se está creando el nuevo archivo, de forma automática el programa abrirá la ventana de Creación de una habitación (Add a Room).

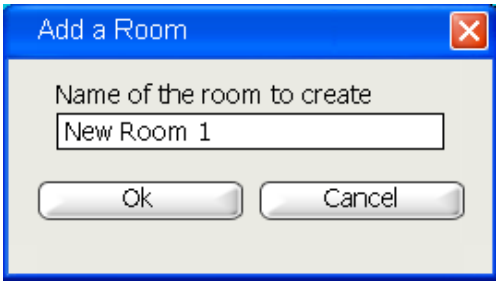


Figura Anexo 5.11. Ventana de creación de una habitación.

Donde introduciremos como podemos ver el nombre de la habitación a crear. Una vez pulsado el botón "Ok" la habitación se añadirá y aparecerá en la vista general de habitaciones (Rooms) a la izquierda del panel.

Como vemos nos aparecen las habitaciones creadas, pudiendo obtener en nuestro panel tanto una vista general de los módulos como una vista específica de los módulos de cada habitación.

Figura Anexo 5.12. Vista general de habitaciones creadas.

Para realizar cambios en una habitación ya creada, simplemente se selecciona la habitación deseada, se da click derecho en el ratón o en la barra de menú ***Edit*** y se selecciona la acción deseada: cambiar el nombre de la habitación (Rename Room o Edit Room), borrar la habitación (Delete Room) o bien añadir módulos o macros (Add Module o Add Macro).

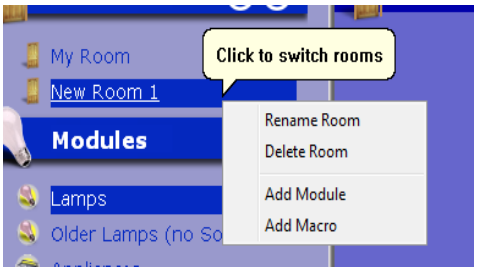


Figura Anexo 5.13. Opciones de configuración de una habitación.

5.2.2. Creación y configuración de módulos.

Una vez que hemos instalado todos los dispositivos hardware necesarios, que están conectados a la red eléctrica y se ha asignado el código de casa y dispositivo físicamente a cada dispositivo, se colocarán en los módulos virtuales que representan a los verdaderos en el entorno **Active Home Pro** y se le asociará el mismo código que a estos.

Para seleccionar los dispositivos hardware instalado existe un menú de módulos en el que vamos seleccionando la categoría que queremos y dentro de la misma el dispositivo que se corresponde con el módulo instalado en nuestra red eléctrica.

Figura Anexo 5.14. Panel principal señalando las diferentes categorías y tipos de módulos.

Estas categorías son:

1. **Para lámparas.**

   Módulos de luminosidad variable y aquellos que están diseñados sólo para el control de luces incandescentes. Los más usados son:

   - Módulo para control de luminosidad LM465.

Figura Anexo 5.15.Módulo LM465.

   - Base enroscable para lámpara LM15A.

Figura Anexo 5.16. Base para lámpara LM15A.

   - Switch para pared Leviton LEV6381.

Figura Anexo 5.17. Switch Leviton LEV6381.

2. Para módulos de aparato de encendido/apagado.

Módulos solo con control encendido/apagado que están diseñados para usarse en cargas de alto voltaje, motores pequeños, etc… Los más usados son:

- Módulo de 2 pines AM486.

Figura Anexo 5.18. Módulo AM486.

- Módulo de 3 pines AM466.

Figura Anexo 5.19. Módulo AM466.

- Módulo para carga pesada 220 V, 20 Amp HD245.

Figura Anexo 5.20. Módulo HD245.

3. Sensores.

Inalámbricos, alimentados por una batería. No se pueden controlar los sensores desde la computadora pero se puede observar cuándo han sido activados.

- Sensor de movimiento HawKEye MS14.

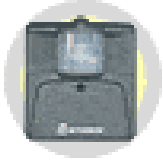


Figura Anexo 5.21. HawKEye MS14.

- Sensor puerta/ventana.

Figura Anexo 5.22. Sensor puerta/ventana.

4. Otros.

Módulos con funciones especiales y de otro tipo de uso que todos los anteriores.

- Llavero remoto KR21A.

Figura Anexo 5.23. Llavero remoto KR21A.

- Adaptador para termostato TH2807.

Figura Anexo 5.24. Adaptador para termostato TH2807.

- Cámara de vigilancia XX11A.

Figura Anexo 5.25. Cámara de vigilancia XX11A.

Una vez explicados los tipos de módulos que este software nos ofrece, vamos a explicar de forma detallada cómo agregar estos módulos en nuestro panel creado. Existen dos formas de añadir módulos al procesamiento de datos del programa.

A. Ventana de adición de módulo (Add a Module).

Para añadir un nuevo módulo se seleccionará la opción ***Add Module*** del menú ***Edit*** de la barra de herramientas, de esta forma nos aparecerá la ventana "Add a Module".

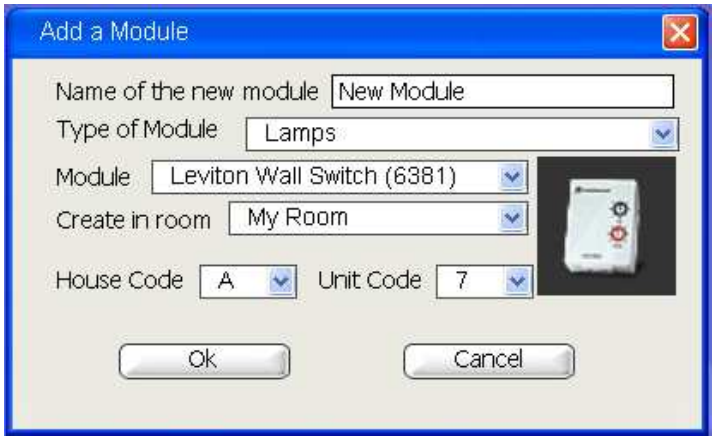


Figura Anexo 5.26. Ventana para añadir y configurar un módulo.

Donde como vemos se puede añadir el nombre que se desea dar al módulo, la categoría del módulo (Lamps, Appliance, etc…), el modelo y la habitación en la que se creará. El código de casa y dispositivo se añadirá automáticamente por el programa, sin embargo se puede modificar en la ventana mostrado.

B. Usando recuadros de módulos.

En este caso basta simplemente con arrastrar el módulo deseado del menú inferior al panel de la habitación en la que queremos añadirlo.

Figura Anexo 5.27. Añadir módulos usando recuadros de módulos.

Se agregará así el módulo y haciendo click derecho sobre el mismo nos aparecerá el menú mostrado en la figura 64 para poder modificar las características del mismo.

Finalmente sea cual sea la forma de añadir el módulo nos quedará la siguiente ventana sobre nuestro panel creado.

Figura Anexo 5.28. Módulo que queda añadido a nuestro panel.

Para actuar sobre los módulos X10 instalados simplemente se deberá colocar el ratón sobre el botón ON/OFF y pulsarlo (después se explicará cómo llevar a cabo esta acción empleando el ***WiiMote***). Así el interruptor cambiará de estado y la acción ejecutada se transmitirá al aparato conectado. También observamos en esta figura que usando la barra de desplazamiento (a la derecha del interruptor),

arrastrando arriba o abajo, se puede regular la intensidad de la luz (en este módulo en concreto), lo que recibe el nombre de función *Dimmer.*

5.2.3. Ventana principal Active Home Pro.

En este apartado se da una explicación de la información y los menús mostrados en la ventana principal del software.

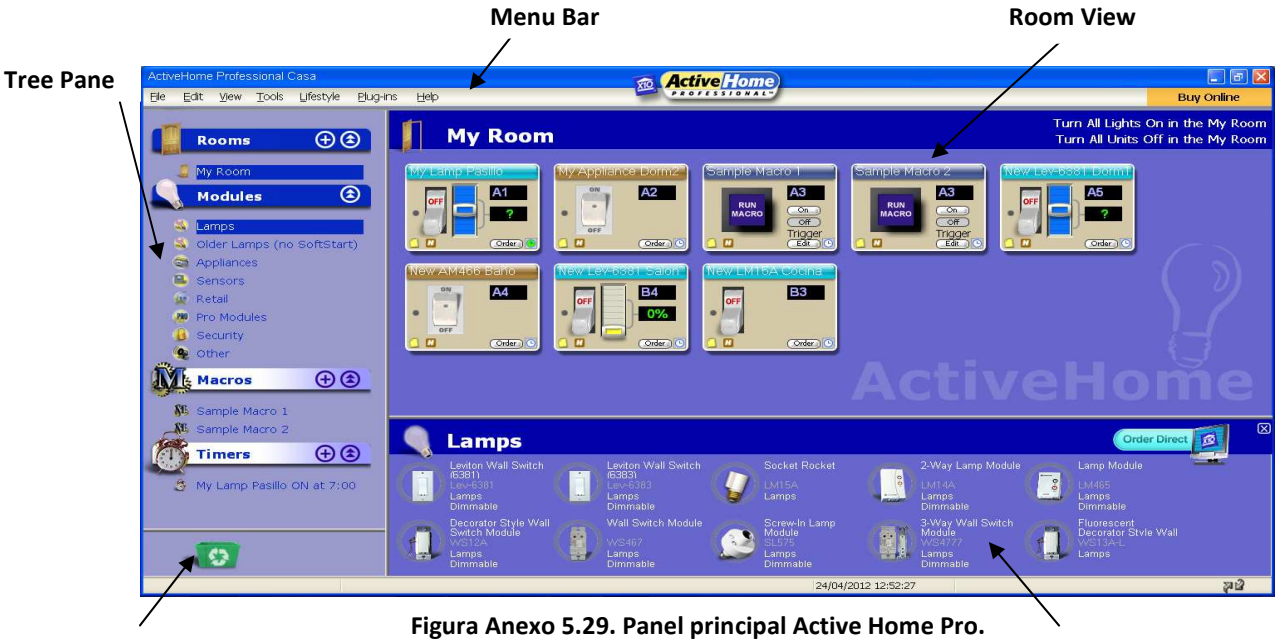


Figura Anexo 5.29. Panel principal Active Home Pro.

Como vemos esta ventana consta de 5 elementos principales:

- **Room View.** Es la ventana más importante en el software. Todos los módulos creados, para luces, macros, etc… aparecerán aquí para organizarlos y controlarlos.

- **Menu Bar.** Proporciona el acceso a todas las opciones y configuraciones existentes en todo el programa, además de facilitar formas distintas de ordenar y usar los módulos.

- **Tree Pane.** Proporciona una visualización general de las habitaciones, así como los diferentes tipos de módulos y las macros y eventos temporales creados, dicho de otra forma proporciona una visualización general del panel creado.

- **Active Pane.** Muestra diferentes opciones de configuración para el "Room View". Pueden aparecer dos cosas en este panel, el "Timer Designer" (programación de eventos temporizados) y "Module List" (lista de módulos).

- **Recycle Bin.** Cuando algo es eliminado en este software es dirigido a este archivo. Cuando se da un click a este símbolo, éste abre su contenido en "Room View", donde se pueden reestablecer ítems borrados o eliminarlos permanentemente del PC.

5.2.4. Control de módulos de luces, de expansión y otros.

Cuando se tiene una visión de los módulos creados, en la vista de "Room View" existe un control directo para cada uno de los módulos de lámparas, expansión y otros en la configuración de Active Home Pro. Estos controles se muestran en la representación virtual de estos módulos y serán diferentes dependiendo del tipo de dispositivo. A continuación se va a dar una breve descripción de estos controles.

- **Control para módulos de luces.**

Figura Anexo 5.30. Control para módulos de luces.

Una lámpara de luminosidad variable tiene dos controles: un switch de Encendido/Apagado y una barra deslizadora para controlar el nivel de brillantez. Si se incrementa el brillo de una lámpara apagada, el software la encenderá. Si en caso contrario se desliza el control de la lámpara a 0% el software la apagará, además de poder dar diferentes grados de brillo al dispositivo concreto.

- **Módulos de expansión.**

Estos módulos únicamente permiten encender o apagar el módulo en concreto utilizando para ello un switch ON/OFF.

Figura Anexo 5.31. Control para módulo de expansión.

- **Macros.**

Este tipo de acciones se crean para efectuar diversos tipos de actividades sobre grupos de dispositivos X10, cuando éstos se usan frecuentemente de forma similar lo cual es posible gracias a que **Active Home Pro** cuenta con una acción en su aplicación de poder crear estos eventos automáticos. Este

software no proporciona un límite sobre el número de aparatos a controlar desde una macro ni el número de macros.

Para crear una macro hay que pulsar en el menú Macro/Nueva Macro, aparecerá el generador de macros con la nueva macro y todos los dispositivos disponibles, como se indica en la siguiente figura.

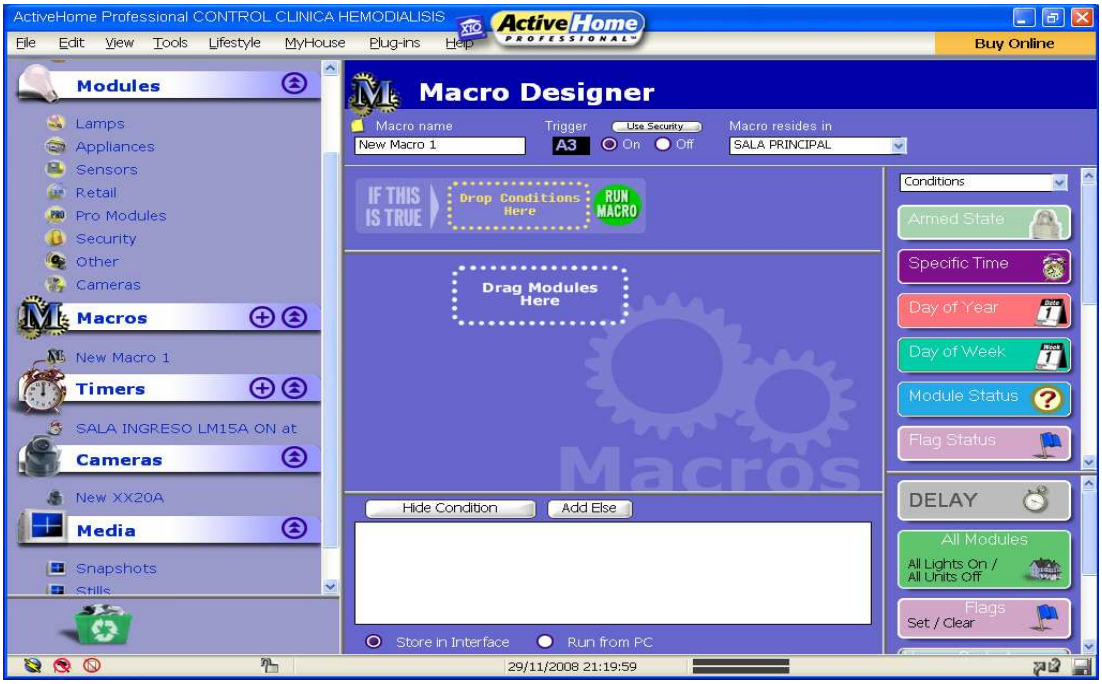


Figura Anexo 5.32. Ventana para la creación de macros (Macro Designer).

Una vez creada la macro nos aparecerá el siguiente módulo en nuestro panel.

Figura Anexo 5.33. Módulo para el manejo de una macro que aparece en el panel una vez creada ésta.

Cuando se da un clic al botón Run Macro, empezarán los eventos de Macro. Si la macro se ejecuta desde el PC (no está guardada en la memoria de la interfaz), se puede volver a pulsar el botón para detener el desarrollo de los eventos de la macro.

A continuación se va a dar una explicación de cómo crear estas macros empleando el diseñador de macros (Macro Designer).

Diseñador de Macros (Macro Designer).

Como ya hemos mencionado esta ventana se usa para crear y modificar macros. Se puede acceder a esta ventana de diferentes formas: dando un click en una Macro existente en la lista "Macros" del "Tree View", dando click en el botón (+) para crear una nueva Macro, usando el botón "Edit" en una Macro en la vista "Room View" y otras.

Barra de información (Information Bar).

Se puede acceder y cambiar la información de una Macro en la Barra de información en el extremo superior de la ventana Macro Designer.

Figura Anexo 5.34. Barra de información de la macro.

Que, como vemos permite escribir el nombre de la macro en "Macro name", lo que la macro hace o dónde se va a usar. Se le puede dar el nombre que se desea puesto que eso no afecta para nada en su funcionalidad.

La información que se introduce en el campo "Tigger" indica al software qué comando se quiere usar para accionar la Macro con un control remoto X10 u otro controlador. Este comando siempre tendrá un direccionamiento X10 más un ON y OFF. Por ejemplo, en este caso, la macro iniciará cuando el dispositivo con dirección "A7" esté en ON. También se puede crear una macro que se inicie cuando el dispositivo con dirección "A7" esté en OFF. Al contrario de los módulos regulares, dos Macros pueden compartir el mismo direccionamiento, mientras que una utilice el accionamiento ON y la otra el OFF.

Cuando se crea un evento Macro, éste se crea para una habitación concreta. Se puede cambiar de habitación sin que esto afecte a los comandos que la macro utiliza.

Cómo Activar una Macro.

Se puede activar una Macro de diferentes maneras:

➔ Hacer click en el botón "Run Macro" en la vista "Room View"

➔ Se puede programar el temporizador para enviar el código de accionamiento para que la Macro comience a ejecutarse a una hora específica.

➔ Presionar un botón en cualquiera de los controladores X10 o controles remotos.

➔ Programar a un sensor de movimiento para que envíe el comando de accionamiento.

Previamente nos aseguraremos que la macro este guardada en la interfaz.

Lista de módulos.

Para añadir comandos a una Macro, se arrastra el módulo que se quiere controlar de la Lista de Módulos (Module List) que está al lado derecho de la ventana Macro Designer, hacía la línea de tiempo de Macro (Macro TimeLine). En el extremo superior de la lista están los módulos de la habitación seleccionada. Estos módulos se cambian de habitación usando la lista de arrastre y los módulos también cambiarán.

En la lista de habitaciones los módulos de luminosidad variable (lámparas) se distinguen por una barra de título azul claro y los módulos de ampliación se muestran en marrón. Cada habitación también tiene una lista, un módulo especial que podría usarse para controlar todos los demás módulos de la respectiva habitación. Lo que es una manera sencilla de encender y apagar todos los módulos de una habitación a la vez.

Figura Anexo 5.35. Lista de módulos de habitaciones.

Bajo la lista de módulos de habitación, se muestra una línea especial de comandos.

El comando "DELAY" siempre está en la lista. Arrastrando este recuadro a la "Línea de tiempo" podremos programar un tiempo entre los comandos en la Macro.

Esta lista también tiene formas rápidas para controlar grandes grupos de módulos, el comando "All Modules". También hay un módulo de comando para cada código de casa que se esté utilizando actualmente en **Active Home Pro.**

Figura Anexo 5.36. Línea especial de comandos.

Línea de Tiempo de Macros.

Es donde se construye la Macro. Para aumentar eventos a la macro, se arrastra un módulo o un comando especial a la Línea de Tiempo desde la lista de módulos a la derecha de la pantalla. Cuando se suelta el módulo en la Línea de Tiempo, ya se puede seleccionar que se va a hacer con el módulo.

Figura Anexo 5.37. Línea de tiempo de Macros.

Los módulos para lámparas y otros módulos ajustables, tienen diferentes opciones. Para encender un módulo sólo se selecciona "Set Absolute" y se ajusta el nivel al que se encenderá (arrastrando totalmente a la derecha). Para apagarlo funciona de la misma manera.

Figura Anexo 5.38. Opciones de configuración de un módulo de lámpara en una macro [1].

También se puede usar el comando "Set To" para programar el módulo a un nivel de brillantez específico.

Figura Anexo 5.39. Opciones de configuración de un módulo de lámpara en una macro [2].

Los comandos de "Dim" y "Bright" trabajan por separado. Si se tiene un módulo ya encendido, se utiliza "Dim" o "Bright" para programar la atenuación o el brillo a un nivel específico. Se usa la barra deslizante para ajustar el nivel de luz, como vemos en la figura anterior.

Cuando se arrastra un módulo de aparato a la “Línea de Tiempo” se tienen sólo las opciones ON y OFF.

Figura Anexo 5.40. Opciones de configuración de un módulo de aparato en una macro.

Los temporizadores (Delay) son una parte importante de una Macro. Cuando se arrastra un temporizador hacia la “Línea de Tiempo”, se está indicando al programa que espere antes de realizar el próximo paso.

Para realizar una temporización, se ajusta el tiempo en el recuadro. Los temporizadores son importantes cuando se quieren automatizar varias actividades complejas en el hogar.

Figura Anexo 5.41. Ajuste del temporizador en una macro.

Cada vez que se va añadiendo algo a la Macro el software automáticamente llena la descripción de los comandos en el recuadro bajo la “Línea de Tiempo”. Es otra forma de mostrar que es lo que la Macro realizará al ejecutarla.

Figura Anexo 5.42. Cuadro de texto donde se van añadiendo los eventos que se van programando en la macro.

Cómo ejecutar una Macro con el PC apagado.

Bajo el recuadro de descripción de comandos hay dos casillas para marcar: “Store in Interface” y “Run from PC”. Si seleccionamos la primera opción la macro se almacenará en la interfaz, además de almacenarse en el PC. Cuando una macro es almacenada en la interfaz, se puede comenzar a ejecutar mediante una temporización o mediante un control remoto incluso cuando el PC está apagado.

Vista Completa de Macros.

Finalmente nos queda todo lo explicado anteriormente.

Figura Anexo 5.43. Vista completa de configuración de una macro.

Cuando seleccionamos para visualizar todas las macros nos aparece una ventana en la que cada macro se muestra con su descripción textual, información de su detonador, el tiempo total de ejecución y en qué habitación está trabajando. Para editar un programa Macro específico se escoge la opción “Edit” de su menú. También se observa un indicador de estado de la Macro: el icono del módulo indica si una macro está almacenada en la memoria de la interfaz **Active Home Pro**. Y el icono del reloj indica si hay temporizadores en el programa, estará de color verde si los hay y blanco en caso contrario.

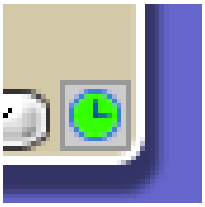


Figura Anexo 5.44. Icono que indica si hay temporizadores.

- **Módulos especiales.**

Existen algunos módulos que tienen funciones diferentes a las de encendido y apagado, como por ejemplo los empleados para que una alarma suene, sólo se tiene que hacer clic en la bocina.

Figura Anexo 5.45. Módulo empleado para alarma.

En cambio los sensores de movimiento no tienen ningún botón de control. Estos sólo envían comandos pero no los recibe. El programa únicamente muestra cuando están activos.

Figura Anexo 5.46. Módulo para sensores.

5.2.5. Programación de eventos.

La programación de eventos permite apagar o encender un dispositivo en una fecha determinada. Este hecho es la novedad más importante que introduce el software **Active Home Pro**. Para cada módulo o Macro en el sistema se pueden tener múltiples temporizadores para diferentes horas, días de la semana y fechas a lo largo del año. Se pueden programar eventos para el amanecer o anochecer, e incluso hacer que ciertos eventos diarios no se repitan exactamente a la misma hora.

Cómo abrir el Timer Designer (Programador de Tiempo).

Los temporizadores para módulos se crean en el recuadro “Timer Designer”. Para crear un evento se pulsa sobre la ventana de programación de eventos (reloj) del módulo X10 concreto.

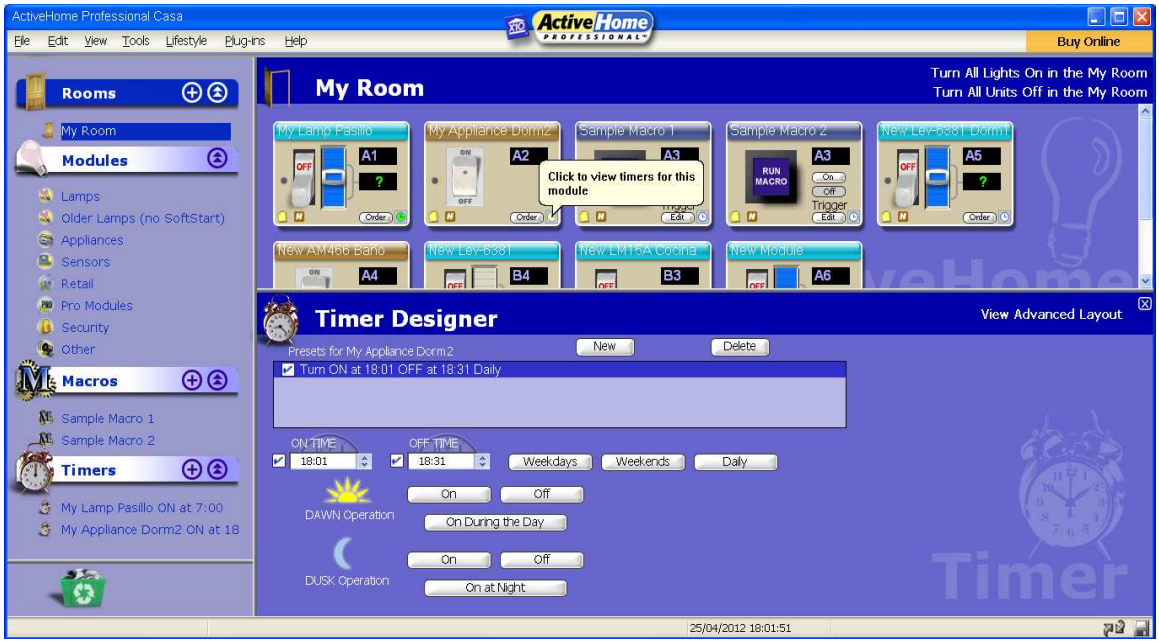


Figura Anexo 5.47. Ventana Timer Designer.

Cómo crear un Temporizador.

Después de abrir el “Timer Designer” se pulsa el botón “New” para añadir un temporizador a la lista. Este abrirá los controles de tiempo y añadirá una nueva línea describiendo el temporizador en la lista de texto.

Cómo programar los temporizadores.

Existen numerosas opciones a la hora de crear los temporizadores.

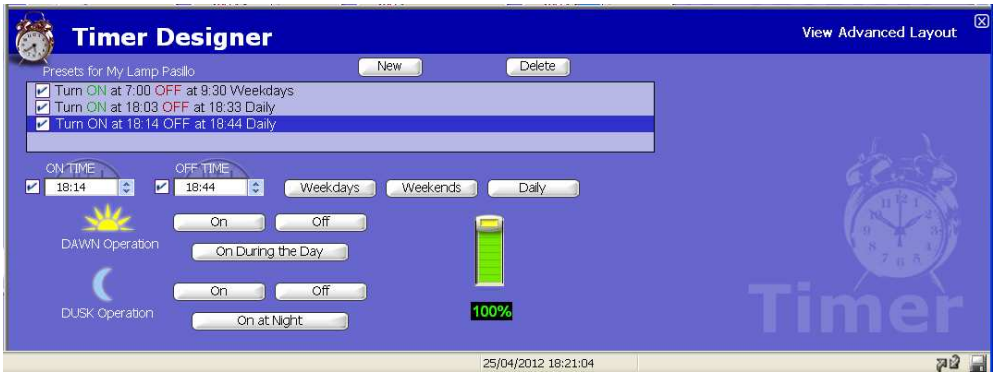


Figura Anexo 5.48. Configuración del temporizador.

Entre las que se encuentran las siguientes:

➔ **Horas de encendido o apagado.**

Al crear por primera vez un temporizador, la hora de encendido estará a una hora programada de fábrica y el tiempo de apagado será después de 30 minutos.

Figura Anexo 5.49. Horas de encendido y apagado.

Se emplean las fechas para cambiar la hora (en incrementos de 5 minutos), o se puede cambiar directamente en el campo de texto. Pero un temporizador no tiene por qué tener una hora de encendido y una hora de apagado, se puede programar el temporizador sólo para encender a una hora específica sin un apagado, simplemente retirando el visto de la casilla de apagado (Off Time).

➔ **Activación con fechas.**

Se pueden programar los temporizadores sólo para que trabajen sólo durante ciertas fechas. Se usan las flechas para cambiar las fechas, o al igual que en el caso de las horas, cambiando directamente la fecha en el campo de texto.

➔ **Días de la semana.**

El software dispone de una opción en la se programa los días que se desea que funcione el temporizador. Existen dos botones, uno para seleccionar que funcione el temporizador los días entre semana y otro para que funcione los fines de semana.

Figura Anexo 5.50. Días en los que funcionará el temporizador.

➔ **Nivel de luminosidad.**

Si se tiene un módulo de luminosidad variable se tiene también la opción de programar el nivel de luminosidad.

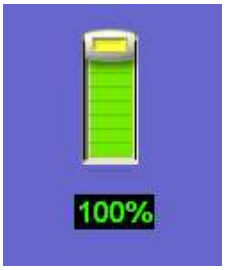


Figura Anexo 5.51. Nivel de luminosidad del aparato asociado al temporizador.

➔ **Opciones especiales.**

Cuando se programa un temporizador hay tres opciones de cómo hacerlo:

1. **Security.** Hace que el software ajuste el temporizador al azar, de tal manera que corra 30 minutos antes o 30 minutos después de la hora programada. Esto ayuda para que el funcionamiento del hogar se muestre natural, si se ha salido de la vivienda, para detener un posible intento de robo por ausencia.

2. **Store in interface.** Hace que el software guarde la programación de los temporizadores en la memoria de la interfaz, para que funcione incluso cuando el PC esté apagado.

3. **Repeat.** Indica al software que el comando para este temporizador debe ser enviado repetidas veces al módulo para asegurarse que éste funcione.

➔ **Amanecer y anochecer (Dusk and Down).**

El software proporciona información sobre las horas del amanecer y anochecer en todo el mundo, lo cual proporciona facilidad al programar un ajuste automático de la hora exacta en que se deben encender luces cuando se acortan o se alargan los días dependiendo de la estación del año.

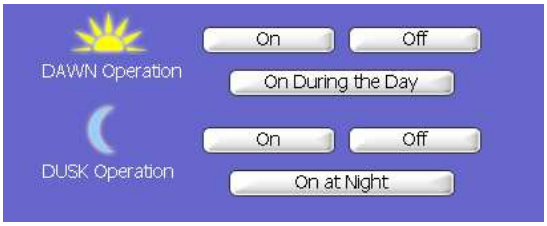


Figura Anexo 5.52. Programación del temporizador para que funcione de día o de noche.

Para ajustar la programación, se debe ingresar la información de la ciudad, longitud y latitud donde se vive en el menú de configuración (Configuration Menu).

Se puede escoger que ambos funcionen a la vez, o solamente uno, pero sólo se puede tener uno con el botón ON activado. Cuando se usa esta programación, no se necesita poner una hora exacta.

Los dos botones de “On During the Day” o “On at Night” son para tener disponible una manera rápida para mantener un módulo encendido durante todo el día o toda la noche. Si el botón marcado es “On at Night”, se encenderá al anochecer y se apagará al amanecer. Cuando esté marcado el botón “On During the Day” ocurrirá lo contrario.

Cuando hay pocos módulos temporizados, es fácil guardar la programación en la interfaz, pero cuando son muchos módulos la memoria no será suficientemente grande, en este caso será recomendable agrupar las actividades de amanecer y anochecer en una Macro.

➔ **Lista de Temporizadores.**

Cuando se está en el “Time Designer”, todos los temporizadores del módulo están detallados en una ventana de texto en la parte superior del recuadro.

Figura Anexo 5.53. Lista de temporizadores programados.

La lista describe cada temporizador y cuando habrá cambios de estado en el módulo. Si no desea un temporizador activo, solo hay que retirar el visto del casillero al lado derecho de éste. Para eliminar un temporizador simplemente se selecciona y se pulsa el botón de "Delete".

➔ **Guardando temporizadores.**

Cuando se ajusta la programación de los temporizadores, los cambios se guardan automáticamente. Como ya hemos comentado anteriormente podemos comprobar qué módulos están temporizados revisando el icono del reloj que hay en éstos, si dicho icono está en verde es que está temporizado y si está blanco es que no lo está.

Figura Anexo 5.54. Icono de temporización.

➔ **Ventana de la lista resumen de temporizadores.**

El "Timer Summary View" es una forma de observar todos los temporizadores que están programados, ya sea de una sola habitación o del conjunto de habitaciones. Para abrir esta ventana se pulsa en la barra de temporización ("Timers Bar") en el "Tree Pane" cuando no se tenga seleccionado ningún módulo. Así se abrirá la lista para la habitación escogida.

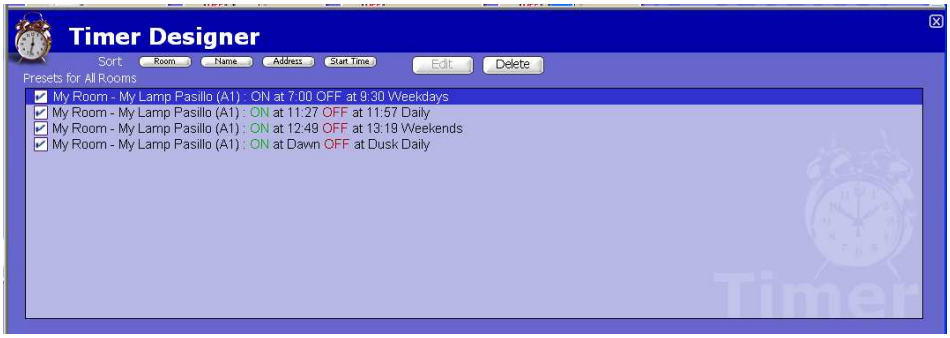


Figura Anexo 5.55. Ventana resumen de temporizadores.

En las listas se puede poner el visto o no ponerlo para encender o apagar los temporizadores. Se pueden usar los botones "Room", "Module", "Address" y "Start Time" para cambiar la información de los temporizadores en lista.

El registro de eventos ocurridos es indicado por la aplicación **Active Home Pro**, a través de histórico de eventos en la opción Activity Monitor indicado en la siguiente figura.

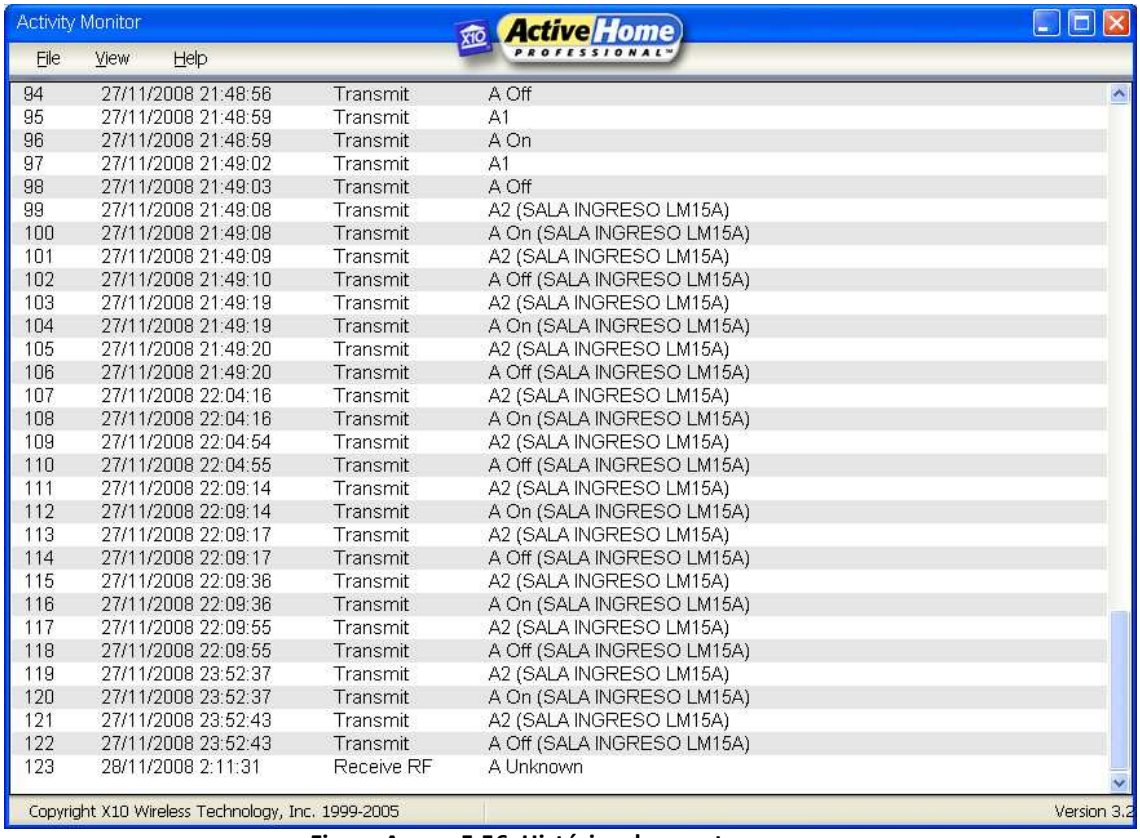


Figura Anexo 5.56. Histórico de eventos.

El software **Active Home Pro** también permite el monitoreo remoto a través de sus cámaras X10, sus dispositivos se conectan a la red eléctrica y este a su vez envía una señal RF de video al dispositivo Wireless Video Receiver VR, que se conecta a través del puerto del PC, y visualizada a través de la aplicación **Active Home Pro**.

En los últimos años **Active Home Pro** ha desarrollado parches para mejorar el control remoto del hogar añadiendo a la aplicación de **Active Home Pro** "My House Online". Esta funcionalidad de la aplicación permite conectarse remotamente desde cualquier parte del mundo a través de un acceso de internet hacia el hogar, y lograr visualizar localizaciones de la casa a través de las diferentes cámaras X10 instaladas y controlar los dispositivos configurados, el sistema permite simular que la casa está habitada cuando no hay personas en su interior, a través del encendido aleatorio de luces y sistemas sonoros que se comportan como si la vivienda estuviera realmente habitada.

Figura Anexo 5.57. Visualización de una cámara X10.

ANEXO 6.

Código completo.

Clase WiiRemote.

```
public abstract class WiiRemote implements WiiUseApiListener {

  public static void main (String [] args) {

    Wiimote[] wiimotes = WiiUseApiManager.getWiimotes(1,true);

    if(wiimotes.length != 0){
      wiimotes[0].activateMotionSensing();
      int fin;
      createAndShowGUI interfaz = new createAndShowGUI();
      interfaz.nombreArchivo();

      Lista listaEnl2 = new Lista();
      boolean bucle = true;
      int botones = 0;
      while(bucle){
            fin = interfaz.pedirFin();
            if(fin == 1){
              listaEnl2 = interfaz.devolverLista();
              botones = interfaz.devolverNumBot();
              bucle = false;
            }
          }

      MyListener mandowii = new MyListener();
      wiimotes[0].addWiiMoteEventListeners(mandowii);
      mandowii.recibirLista(listaEnl2,botones);

    }

    else{
      System.out.println("MANDO WIIMOTE NO DETECTADO");
      WiiUseApiManager.definitiveShutdown();
      System.exit(1);
    }
  }
}
```

Clase createAndShowGUI.

```
public class createAndShowGUI implements ActionListener{

 JFrame frame;
 JFrame frame2;
 JFrame frame3;
 JFrame frame4;
 JPanel panel;
 JPanel panel2;
 JPanel panel3;
 JPanel panel4;
 JPanel panel5;
 JPanel panel6;
 JPanel panel7;
 JPanel panel8;
 JPanel panel9;
 JPanel panel10;
 JPanel panel11;
 JLabel labelPpal;
 JLabel label2;
 JLabel label3;
 JLabel label4;
 JTextField campoTexto;
 JTextField campoTexto2;
 JTextField campoTexto3;
 JButton boton;
 JButton boton2;
 JButton boton3;
 JButton boton4;
 JButton boton5;
 JScrollPane scrollBar;
 Lista listaEnl = new Lista();
 int fin = 0;
 String archivo = null;
 double log2;
 double logNumDisp;
 double botPulsados;
 int botPulsadosEntero;
 int botonesTotales;

 public createAndShowGUI() {}

 public void nombreArchivo(){

   frame3 = new JFrame("Nombre del archivo");
   panel4 = new JPanel();
   label3 = new JLabel ("Introduzca el nombre del archivo de Active
   Home Pro (sin la extensión)");
   campoTexto3 = new JTextField(10);
   boton4 = new JButton("Siguiente");

   panel4.add(label3);
   panel4.add(campoTexto3);
   panel4.add(boton4);
   frame3.add(panel4);

   boton4.addActionListener(new ActionListener(){
```

```
    public void actionPerformed(ActionEvent e){
     archivo = campoTexto3.getText();
     if(archivo.equals("")){
           String str = "Debe introducir el nombre del archivo
           ActiveHome Pro" + "\n";
       JOptionPane.showMessageDialog(null, str);
     }
     else{
       crearInterfaz(e,archivo);
     }
     }
   });

   frame3.addWindowListener(new java.awt.event.WindowAdapter(){
      public void windowClosing(WindowEvent e){
     System.exit(0);
      }
   });

   frame3.pack();
   frame3.setLocationRelativeTo(null);
   frame3.setVisible(true);
}


public void crearInterfaz(ActionEvent e,String archivo){

   String comando;
   comando = "cmd /c " + archivo + ".ahx";
   PasoDeParametros param = new PasoDeParametros(comando);
   param.envioComando();

   frame = new JFrame("Numero de dispositivos");
   panel = new JPanel();
   labelPpal = new JLabel("Introduzca el número de dispositivos que
   tiene (deben ser mínimo 2)");
   campoTexto = new JTextField(10);
   boton = new JButton("Continuar");

   panel.add(labelPpal);
   panel.add(campoTexto);
   panel.add(boton);
   frame.add(panel);

   boton.addActionListener(this);

   frame.addWindowListener(new java.awt.event.WindowAdapter(){
     public void windowClosing(WindowEvent e){
      System.exit(0);
     }
   });

   frame.pack();
   frame.setLocationRelativeTo(null);
   frame.setVisible(true);
}


public void actionPerformed(ActionEvent e) {
   String cad;
```

```
   cad = campoTexto.getText();

   if(cad.equals("")){
      String str = "Debe introducir el número de dispositivos que
      tiene" + "\n";
      JOptionPane.showMessageDialog(null, str);
   }
   else{
       final int valorEnteroDeCad = Integer.parseInt(cad);
       botonesTotales = calcularBotonesPulsados (valorEnteroDeCad);

       if(boton == e.getSource()){
          crearVentana(cad,botonesTotales,valorEnteroDeCad);
       }
   }
}



private void crearVentana(String cad, final int botonesTotales,final
int valorEnteroDeCad) {

   int i;
   int a;
    panel2 = new JPanel(new FlowLayout());
   panel2.setLayout(new java.awt.GridLayout((2*valorEnteroDeCad), 120
   / 120));

   for(i=0;!cad.equals(String.valueOf(i));i++){
    a = i + 1;
    String nom1 = "dispositivo "+a;
    JPanel panel6 = new JPanel();
    JPanel panel7 = new JPanel();
    JPanel panel8 = new JPanel();
    JPanel panel9 = new JPanel();
    JPanel panel10 = new JPanel();
    JPanel panel11 = new JPanel();
    JLabel label3 = new JLabel();
    JLabel label4 = new JLabel();
    JButton boton3 = new JButton();
    final Choice seleccion1 = new Choice();
    final Choice seleccion2 = new Choice();
    label3.setText("Introduzca la direccion del" + nom1);
    label4.setText("Introduzca el tipo de dispositivo: 'On y Off' o
    'Dim y Bright'");
    boton3.setText(nom1);
    seleccion1.add("A1");seleccion1.add("A2");seleccion1.add("A3");sel
    eccion1.add("A4");seleccion1.add("A5");seleccion1.add("A6");selecc
    ion1.add("A7");seleccion1.add("A8");seleccion1.add("A9");seleccion
    1.add("A10");seleccion1.add("A11");seleccion1.add("A12");seleccion
    1.add("A13");seleccion1.add("A14");seleccion1.add("A15");seleccion
    1.add("A16");
    seleccion1.add("B1");seleccion1.add("B2");seleccion1.add("B3");sel
    eccion1.add("B4");seleccion1.add("B5");seleccion1.add("B6");selecc
    ion1.add("B7");seleccion1.add("B8");seleccion1.add("B9");seleccion
    1.add("B10");seleccion1.add("B11");seleccion1.add("B12");seleccion
    1.add("B13");seleccion1.add("B14");seleccion1.add("B15");seleccion
    1.add("B16");
    seleccion1.add("C1");seleccion1.add("C2");seleccion1.add("C3");sel
    eccion1.add("C4");seleccion1.add("C5");seleccion1.add("C6");selecc
```

```
ion1.add("C7");seleccion1.add("C8");seleccion1.add("C9");seleccion
1.add("C10");seleccion1.add("C11");seleccion1.add("C12");seleccion
1.add("C13");seleccion1.add("C14");seleccion1.add("C15");seleccion
1.add("C16");
seleccion1.add("D1");seleccion1.add("D2");seleccion1.add("D3");sel
eccion1.add("D4");seleccion1.add("D5");seleccion1.add("D6");selecc
ion1.add("D7");seleccion1.add("D8");seleccion1.add("D9");seleccion
1.add("D10");seleccion1.add("D11");seleccion1.add("D12");seleccion
1.add("D13");seleccion1.add("D14");seleccion1.add("D15");seleccion
1.add("D16");
seleccion1.add("E1");seleccion1.add("E2");seleccion1.add("E3");sel
eccion1.add("E4");seleccion1.add("E5");seleccion1.add("E6");selecc
ion1.add("E7");seleccion1.add("E8");seleccion1.add("E9");seleccion
1.add("E10");seleccion1.add("E11");seleccion1.add("E12");seleccion
1.add("E13");seleccion1.add("E14");seleccion1.add("E15");seleccion
1.add("E16");
seleccion1.add("F1");seleccion1.add("F2");seleccion1.add("F3");sel
eccion1.add("F4");seleccion1.add("F5");seleccion1.add("F6");selecc
ion1.add("F7");seleccion1.add("F8");seleccion1.add("F9");seleccion
1.add("F10");seleccion1.add("F11");seleccion1.add("F12");seleccion
1.add("F13");seleccion1.add("F14");seleccion1.add("F15");seleccion
1.add("F16");
seleccion1.add("G1");seleccion1.add("G2");seleccion1.add("G3");sel
eccion1.add("G4");seleccion1.add("G5");seleccion1.add("G6");selecc
ion1.add("G7");seleccion1.add("G8");seleccion1.add("G9");seleccion
1.add("G10");seleccion1.add("G11");seleccion1.add("G12");seleccion
1.add("G13");seleccion1.add("G14");seleccion1.add("G15");seleccion
1.add("G16");
seleccion1.add("H1");seleccion1.add("H2");seleccion1.add("H3");sel
eccion1.add("H4");seleccion1.add("H5");seleccion1.add("H6");selecc
ion1.add("H7");seleccion1.add("H8");seleccion1.add("H9");seleccion
1.add("H10");seleccion1.add("H11");seleccion1.add("H12");seleccion
1.add("H13");seleccion1.add("H14");seleccion1.add("H15");seleccion
1.add("H16");
seleccion1.add("I1");seleccion1.add("I2");seleccion1.add("I3");sel
eccion1.add("I4");seleccion1.add("I5");seleccion1.add("I6");selecc
ion1.add("I7");seleccion1.add("I8");seleccion1.add("I9");seleccion
1.add("I10");seleccion1.add("I11");seleccion1.add("I12");seleccion
1.add("I13");seleccion1.add("I14");seleccion1.add("I15");seleccion
1.add("I16");
seleccion1.add("J1");seleccion1.add("J2");seleccion1.add("J3");sel
eccion1.add("J4");seleccion1.add("J5");seleccion1.add("J6");selecc
ion1.add("J7");seleccion1.add("J8");seleccion1.add("J9");seleccion
1.add("J10");seleccion1.add("J11");seleccion1.add("J12");seleccion
1.add("J13");seleccion1.add("J14");seleccion1.add("J15");seleccion
1.add("J16");
seleccion1.add("K1");seleccion1.add("K2");seleccion1.add("K3");sel
eccion1.add("K4");seleccion1.add("K5");seleccion1.add("K6");selecc
ion1.add("K7");seleccion1.add("K8");seleccion1.add("K9");seleccion
1.add("K10");seleccion1.add("K11");seleccion1.add("K12");seleccion
1.add("K13");seleccion1.add("K14");seleccion1.add("K15");seleccion
1.add("K16");
seleccion1.add("L1");seleccion1.add("L2");seleccion1.add("L3");sel
eccion1.add("L4");seleccion1.add("L5");seleccion1.add("L6");selecc
ion1.add("L7");seleccion1.add("L8");seleccion1.add("L9");seleccion
1.add("L10");seleccion1.add("L11");seleccion1.add("L12");seleccion
1.add("L13");seleccion1.add("L14");seleccion1.add("L15");seleccion
1.add("L16");
seleccion1.add("M1");seleccion1.add("M2");seleccion1.add("M3");sel
eccion1.add("M4");seleccion1.add("M5");seleccion1.add("M6");selecc
```

```
 ion1.add("M7");seleccion1.add("M8");seleccion1.add("M9");seleccion
 1.add("M10");seleccion1.add("M11");seleccion1.add("M12");seleccion
 1.add("M13");seleccion1.add("M14");seleccion1.add("M15");seleccion
 1.add("M16");
 seleccion1.add("N1");seleccion1.add("N2");seleccion1.add("N3");sel
 eccion1.add("N4");seleccion1.add("N5");seleccion1.add("N6");selecc
 ion1.add("N7");seleccion1.add("N8");seleccion1.add("N9");seleccion
 1.add("N10");seleccion1.add("N11");seleccion1.add("N12");seleccion
 1.add("N13");seleccion1.add("N14");seleccion1.add("N15");seleccion
 1.add("N16");
 seleccion1.add("O1");seleccion1.add("O2");seleccion1.add("O3");sel
 eccion1.add("O4");seleccion1.add("O5");seleccion1.add("O6");selecc
 ion1.add("O7");seleccion1.add("O8");seleccion1.add("O9");seleccion
 1.add("O10");seleccion1.add("O11");seleccion1.add("O12");seleccion
 1.add("O13");seleccion1.add("O14");seleccion1.add("O15");seleccion
 1.add("O16");
 seleccion1.add("P1");seleccion1.add("P2");seleccion1.add("P3");sel
 eccion1.add("P4");seleccion1.add("P5");seleccion1.add("P6");selecc
 ion1.add("P7");seleccion1.add("P8");seleccion1.add("P9");seleccion
 1.add("P10");seleccion1.add("P11");seleccion1.add("P12");seleccion
 1.add("P13");seleccion1.add("P14");seleccion1.add("P15");seleccion
 1.add("P16");
 seleccion2.add("On y Off");
 seleccion2.add("Dim y Bright");

 panel7.add(label3);
 panel8.add(seleccion1);
 panel9.add(label4);
 panel10.add(seleccion2);
 panel11.add(boton3);

 panel6.add(panel7);
 panel6.add(panel8);
 panel6.add(panel9);
 panel6.add(panel10);
 panel6.add(panel11);

 panel2.add(panel6);

 boton3.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
        clicEnBoton2(e,seleccion1.getSelectedItem(),seleccion2.getS
      electedItem());
      }
 });
}

frame2 = new JFrame("Direcciones y tipos");
boton2 = new JButton("Añadir");
panel2.add(boton2);

scrollBar = new JScrollPane(panel2);

scrollBar.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConsta
nts.HORIZONTAL_SCROLLBAR_ALWAYS);
scrollBar.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstant
s.VERTICAL_SCROLLBAR_ALWAYS);
scrollBar.setViewportView(panel2);
frame2.add(scrollBar,java.awt.BorderLayout.CENTER);
```

```
   boton2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

          clicEnBotonAñadir(e,botonesTotales,valorEnteroDeCad);
    }
   });

   frame2.addWindowListener(new java.awt.event.WindowAdapter(){
    public void windowClosing(WindowEvent e){
     System.exit(0);
    }
   });

   frame2.pack();
   frame2.setLocationRelativeTo(null);
   frame2.setVisible(true);
}



private int calcularBotonesPulsados(int valorEnteroDeCad) {

   log2 = Math.log10(2);
   logNumDisp = Math.log10(valorEnteroDeCad);

   if(logNumDisp == 0){
      String str = "El número de dispositivos debe ser mínimo 2" +
   "\n";
      JOptionPane.showMessageDialog(null, str);
   }

   else{
      botPulsados = (logNumDisp)/(log2);
      if((logNumDisp)%(log2) != 0){
          botPulsadosEntero = (int)(botPulsados) + 1;
      }
      else{
          botPulsadosEntero = (int)(botPulsados);
      }
   }
   return botPulsadosEntero;
}


private void clicEnBoton2(ActionEvent e,String selec1,String selec2)
{

   String cad2 = null;
   String cad3 = null;
   cad2 = selec1;
   cad3 = selec2;
   boolean crear = listaEnl.buscarRepe(cad2);
   if(crear){
      String str = "El dispositivo ya está creado" + "\n";
      JOptionPane.showMessageDialog(null, str);
   }
   else{
      listaEnl = listaEnl.addLast(cad2,cad3,false,0,null,false);
   }
}
```

```
  private void clicEnBotonAñadir(ActionEvent e,int botonesTotales,int
  valorEnteroDeCad){

     fin = 1;
     frame.dispose();
     frame2.dispose();
     frame3.dispose();
     listaEnl.asignarBoton(botonesTotales,valorEnteroDeCad);
     return;
  }


  public Lista devolverLista(){

     return listaEnl;

  }

  public int devolverNumBot(){

     return botonesTotales;

  }

  public int pedirFin(){

     return fin;

  }

}
```

Clase Nodo.

```
public class Nodo {

 Nodo nodoDer;
 String codCasaDisp;
 String tipo;
 boolean estado;
 int porcBrillo;
 int []boton;
 boolean seleccionado;

 public Nodo(String codCasaDisp,String tipo, boolean estado,int
 porcBrillo,int []boton,boolean seleccionado) {
    this.codCasaDisp =  codCasaDisp;
    this.tipo = tipo;
    this.estado = estado;
    this.porcBrillo = porcBrillo;
    this.boton = boton;
    this.nodoDer = null;
    this.seleccionado = seleccionado;
 }

 public Nodo(){}

}
```

Clase Lista.

```
public class Lista {

 private Nodo primero;
 private Nodo ultimo;
 private int tamano;
 int i = 0;
 int j = 0;

 public Lista() {
  this.primero = null;
  this.ultimo = null;
  this.tamano = 0;
 }

 public boolean siVacio() {
  return (this.primero == null);
 }


 public Lista addLast(String codCasaDisp,String tipo, boolean
 estado,int porcBrillo,int []boton,boolean seleccionado) {

    if(siVacio()) {
      Nodo nuevo = new
      Nodo(codCasaDisp,tipo,estado,porcBrillo,boton,seleccionado);
      primero = nuevo;
      ultimo = nuevo;
      nuevo.nodoDer = null;
    }

    else {
      Nodo nuevo = new
      Nodo(codCasaDisp,tipo,estado,porcBrillo,boton,seleccionado);
      nuevo.nodoDer = null;
      ultimo.nodoDer = nuevo;
      ultimo = nuevo;
    }
    this.tamano++;
    return this;

 }


 public void asignarBoton(int botonesTotales,int numDispositivos){

  int matriz[][] = new int[numDispositivos][botonesTotales];
  boolean cambio = true;
  int i = 0;
  int j;

  for(j = matriz[i].length - 1; j>= 0; j--){
    for(i = 0; i < matriz.length; i++){
        if(cambio == true){
            matriz[i][j] = 1;
        }
        else{
```

```
            matriz[i][j] = 2;
        }

        if(((i+1) % (Math.pow(2,matriz[i].length-1-j))) == 0){

           if(cambio == true){
              cambio = false;
           }
           else{
              cambio = true;
           }
         }
     }
     cambio=true;

  }

  Nodo aux = null;
  if(tamano != 0){
     aux = primero;
     for (int k = 0; k < this.tamano; k++){
        aux.boton = matriz[k];
        aux = aux.nodoDer;
     }
  }
}



public Nodo buscar (String codCasaDisp) {

 Nodo aux = null;

 if(siVacio()){
    return null;
 }

 else{
    aux = primero;
    while((aux != null) &&
  (!((aux.codCasaDisp).equals(codCasaDisp)))){
           aux = aux.nodoDer;
    }
    if((aux != null) && ((aux.codCasaDisp).equals(codCasaDisp))){
           return aux;
    }
    else{
           return null;
    }
 }
}

public boolean buscarRepe (String codCasaDisp) {

 Nodo aux = null;

 if(siVacio()){
  return false;
 }
```

```
 else{
  aux = primero;
  while((aux != null) && (!((aux.codCasaDisp).equals(codCasaDisp)))){
      aux = aux.nodoDer;
  }
  if((aux != null) && (aux.codCasaDisp).equals(codCasaDisp)){
       return true;
  }
  else{
       return false;
  }
 }
}


public Nodo buscarTipo (String tipo) {

 Nodo aux = null;

 if(siVacio()){
    return null;
 }

 else{
    aux = primero;
    while((aux != null) && (!((aux.tipo).equals(tipo)))){
       aux = aux.nodoDer;
    }
    if((aux != null) && ((aux.tipo).equals(tipo))){
       return aux;
    }

    else{
          return null;
    }
  }
 }

 public Nodo buscarBoton (int []boton, int numBotones) {

  Nodo aux = null;

  if(siVacio()){
     return null;
  }

  else{
     boolean seguir = true;
     aux = primero;
     while((aux != null) && (aux.boton != null) && (seguir)){
    int []botonBuscado = new int[numBotones];
        botonBuscado = aux.boton;
        int i = 0;
        while((i <= botonBuscado.length - 1) && (botonBuscado[i] ==
        boton[i])){
             i++;
        }
        if(numBotones == 1){
         if(i == 1){
```

```
      seguir = false;
    }
    else{
      aux = aux.nodoDer;
    }
}
else if(numBotones == 2){
    if(i == 2){
      seguir = false;
    }
    else{
      aux = aux.nodoDer;
    }
}
else if(numBotones == 3){
    if(i == 3){
      seguir = false;
    }
    else{
      aux = aux.nodoDer;
    }
}
else if(numBotones == 4){
    if(i == 4){
      seguir = false;
    }
    else{
      aux = aux.nodoDer;
    }
}
else if(numBotones == 5){
    if(i == 5){
      seguir = false;
    }
      else{
        aux = aux.nodoDer;
      }
  }
  else if(numBotones == 6){
      if(i == 6){
        seguir = false;
      }
      else{
        aux = aux.nodoDer;
      }
  }
  else if(numBotones == 7){
      if(i == 7){
         seguir = false;
      }
      else{
         aux = aux.nodoDer;
      }
   }
   else if(numBotones == 8){
      if(i == 8){
         seguir = false;
      }
      else{
         aux = aux.nodoDer;
```

```
            }
          }
      }
      return aux;
        }
}


public Nodo buscarSeleccion () {

  Nodo aux = null;

  if(siVacio()){
    return null;
  }

  else{
    aux = primero;
    while((aux != null) && (!aux.seleccionado)){
          aux = aux.nodoDer;
    }
    if((aux != null) && (aux.seleccionado)){
          return aux;
    }
    else{
          return null;
    }
  }
}


 public void cambiarBrillo(String codCasaDisp, int porcBrillo){

  Nodo aux = null;

  if(siVacio()){
     System.out.println("La lista está vacía");
  }

  else{
     aux = primero;
     while((aux != null) &&
  (!((aux.codCasaDisp).equals(codCasaDisp)))){
          aux = aux.nodoDer;
     }
     if((aux != null) && ((aux.codCasaDisp).equals(codCasaDisp))){
          aux.porcBrillo = porcBrillo;
          if(aux.porcBrillo > 0){
                aux.estado = true;
          }
          else if(aux.porcBrillo == 0){
                aux.estado = false;
          }
        System.out.println("Se ha cambiado el brillo a " +
    aux.porcBrillo);
      }

      else{
```

```
            System.out.println("No existe el dispositivo pedido");

        }
    }
}


public void cambiarEstado(String codCasaDisp, boolean estado){

 Nodo aux = null;

 if(siVacio()){
   System.out.println("La lista está vacía");
 }

 else{
   aux = primero;
   while((aux != null) &&
 (!((aux.codCasaDisp).equals(codCasaDisp)))){
           aux = aux.nodoDer;
   }
   if((aux != null) && ((aux.codCasaDisp).equals(codCasaDisp))){
       if(estado == false){
           aux.estado = estado;
           aux.porcBrillo = 0;
         }
         else if(estado == true) {
           aux.estado = estado;
           aux.porcBrillo = 100;
         }
         System.out.println("Se ha cambiado el estado a " +
 aux.estado);
    }
    else{
          System.out.println("No existe el dispositivo pedido");

    }
 }
}



public void cambiarSeleccion(String codCasaDisp, boolean seleccion){

 Nodo aux = null;

 if(siVacio()){
    System.out.println("La lista está vacía");
 }

 else{
    aux = primero;
    while((aux != null) &&
 (!((aux.codCasaDisp).equals(codCasaDisp)))){
          aux = aux.nodoDer;
    }
    if((aux != null) && ((aux.codCasaDisp).equals(codCasaDisp))){
   aux.seleccionado = seleccion;
    }
    else{
```

```
                System.out.println("No existe el dispositivo pedido");

            }
          }
        }


        public void tamano() {
            JOptionPane.showMessageDialog(null, "El tamaño es:\n " +
            this.tamano);
        }


        public void imprimirBotones() {

         if(tamano != 0) {

            Nodo temp = primero;
            String str = "";

            for(int i = 0; i < this.tamano; i++) {
                int []boton = temp.boton;
                str = str + temp.codCasaDisp + " "+ temp.tipo + " ";
                for(int j = 0; j<boton.length;j++){
                   str = str + boton[j];
                }
                str = str + "\n";
                temp = temp.nodoDer;
             }
             JOptionPane.showMessageDialog(null, str);
          }
        }
}
```

Clase PasoDeParametros.

```
public class PasoDeParametros {

   String commando;
   public PasoDeParametros (String comando){
     this.comando=comando;
   }



 public void envioComando(){
   try{
     Runtime.getRuntime().exec(comando);
  } catch (IOException e) {
     System.out.println("Excepción: ");
     e.printStackTrace();
     System.exit(-1);
  }
 }
 }
```

Clase MyListener.

```
public class MyListener implements WiimoteListener {

  public static int mov_Z = 0;
  public static int mov_Y = 0;
  public static boolean A = false;
  public static boolean B = false;
  public static boolean Home = false;
  public static boolean Mas = false;
  public static boolean Menos = false;
  public static boolean Arriba = false;
  public static boolean Abajo = false;
  public static boolean Dcha = false;
  public static boolean Izq = false;
  public static boolean Uno = false;
  public static boolean Dos = false;
  public Lista listaEnl3 = new Lista();
  public Nodo nodo1 = new Nodo();
  public Nodo nodo2 = new Nodo();
  int contador = 0;
  int numBotones = 0;
  int []boton;

public void onIrEvent(IREvent arg0){
  System.out.println (arg0);
}

public void recibirLista(Lista listaEnl3, int numBotones){
  this.listaEnl3 = listaEnl3;
  this.numBotones = numBotones;
  boton = new int[numBotones];
}

public void onButtonsEvent (WiimoteButtonsEvent boton_evento){

 /*BOTON A*/
 if(boton_evento.isButtonAJustPressed()){
   A = true;
 }

 /*BOTON B*/
 if(boton_evento.isButtonBJustPressed()){
   B = true;
 }
 /*BOTON ABAJO*/
 if(boton_evento.isButtonDownJustPressed()){
   Abajo = true;
   listaEnl3.imprimirBotones();
 }
 /*BOTON ARRIBA*/
 if(boton_evento.isButtonUpJustPressed()){
   Arriba = true;
 }
 /*BOTON DERECHA*/
```

```
if(boton_evento.isButtonRightJustPressed()){
   Dcha = true;
}
/*BOTON IZQUIERDA*/
if(boton_evento.isButtonLeftJustPressed()){
   Izq = true;
}
/*BOTON MAS*/
if(boton_evento.isButtonPlusJustPressed()){
   Mas = true;
}
/*BOTON MENOS*/
if(boton_evento.isButtonMinusJustPressed()){
   Menos = true;
}
/*BOTON HOME*/
if(boton_evento.isButtonHomeJustPressed()){
   Home = true;
}
/*BOTON UNO*/
if(boton_evento.isButtonOneJustPressed()){
   Uno = true;
}
/*BOTON DOS*/
if(boton_evento.isButtonTwoJustPressed()){
   Dos = true;
}


while(((Uno) || (Dos)) && (contador < numBotones)){
   if(Uno){
      Uno = false;
      boton[contador] = 1;
      contador++;
   }
   else if(Dos){
      Dos = false;
      boton[contador] = 2;
      contador++;
   }
}



if(contador == numBotones){
   nodo1 = listaEnl3.buscarBoton(boton,numBotones);
   if(nodo1 == null){
         String cadena2 = "No existe el dispositivo que tiene
         asociados esos botones" + "\n";
      JOptionPane.showMessageDialog(null, cadena2);
   }
   else{
      String direccion = nodo1.codCasaDisp;
      boolean estado = nodo1.estado;
      String tipo = nodo1.tipo;
         int brillo = nodo1.porcBrillo;
      listaEnl3.cambiarSeleccion(direccion, true);
         String cadena = "Su dispositivo tiene las siguientes
         características: ";
         if(tipo.equals("On y Off")){
```

```
                cadena = cadena + " direccion: "+ direccion + " estado:
                " + estado + " brillo: " + brillo + " tipo: " + tipo +
                "\n" + "¿Qué desea hacer?" + "\n" + "encender o apagar:
                A " + "\n" + "Pulse Home cuando termine con el
                dispositivo" + "\n";
            JOptionPane.showMessageDialog(null, cadena);
        }
        else{
                cadena = cadena + " direccion: "+ direccion + " estado:
                " + estado + " brillo: " + brillo + " y es del tipo: "
                + tipo + "\n" + "¿Qué desea hacer?" + "\n" + "encender
                o apagar: A " + "\n" + "aumentar brillo: + " + "\n" +
                "disminuir brillo: - " + "\n" + "Pulse Home cuando
                termine con el dispositivo" + "\n";
            JOptionPane.showMessageDialog(null, cadena);
        }
     }
     contador = 0;
  }


if(A){
    nodo2 = listaEnl3.buscarSeleccion();
    if(nodo2 == null){
       String cadena2 = "No se ha seleccionado ningún dispositivo" +
     "\n";
       JOptionPane.showMessageDialog(null, cadena2);
   }
   else{
       if(nodo2.seleccionado){
          String direccion = nodo2.codCasaDisp;
          boolean estado = nodo2.estado;
          if(!estado){
         String comando;
         comando = "cmd /c ahcmd sendplc " + direccion +  " On";
         PasoDeParametros param = new PasoDeParametros(comando);
         param.envioComando();
         listaEnl3.cambiarEstado(direccion,true);
       }

       else{
         String comando;
         comando = "cmd /c ahcmd sendplc " + direccion + " Off";
         PasoDeParametros param = new PasoDeParametros(comando);
         param.envioComando();
         listaEnl3.cambiarEstado(direccion,false);
       }
        }
   }
     A = false;

}


if(Home){
    nodo2 = listaEnl3.buscarSeleccion();
    if(nodo2 == null){
      String cadena2 = "No se ha seleccionado ningún dispositivo" +
     "\n";
      JOptionPane.showMessageDialog(null, cadena2);
```

```
   }
   else{
     if(nodo2.seleccionado){
       String direccion = nodo2.codCasaDisp;
       listaEnl3.cambiarSeleccion(direccion,false);

       String cadena2 = "Se ha cambiado la seleccion a false del
    dispositivo con direccion " + direccion +  "\n";
       JOptionPane.showMessageDialog(null, cadena2);
     }
   }
   Home = false;
}


if(Mas){
   nodo2 = listaEnl3.buscarSeleccion();
   if(nodo2 == null){
     String cadena2 = "No se ha seleccionado ningún dispositivo" +
    "\n";
     JOptionPane.showMessageDialog(null, cadena2);
   }
   else{
     if(nodo2.seleccionado){
        String direccion = nodo2.codCasaDisp;
        int brillo = nodo2.porcBrillo;
        String tipo = nodo2.tipo;
        if(tipo.equals("Dim y Bright")){
      if(brillo == 0){
        String comando;
            comando = "cmd /c ahcmd sendplc " + direccion + " Bright
            010%";
        PasoDeParametros param = new PasoDeParametros(comando);
            param.envioComando();
                listaEnl3.cambiarBrillo(direccion,10);
          }
      if(brillo == 10){
        String comando;
            comando = "cmd /c ahcmd sendplc " + direccion + " Bright
            010%";
        PasoDeParametros param = new PasoDeParametros(comando);
            param.envioComando();
               listaEnl3.cambiarBrillo(direccion,20);
      }

      if(brillo == 20){
        String comando;
            comando = "cmd /c ahcmd sendplc " + direccion + " Bright
            010%";
        PasoDeParametros param = new PasoDeParametros(comando);
            param.envioComando();
            listaEnl3.cambiarBrillo(direccion,30);
      }
      if(brillo == 30){
        String comando;
            comando = "cmd /c ahcmd sendplc " + direccion + " Bright
            010%";
        PasoDeParametros param = new PasoDeParametros(comando);
        param.envioComando();
        listaEnl3.cambiarBrillo(direccion,40);
```

```
}
if(brillo == 40){
  String comando;
      comando = "cmd /c ahcmd sendplc " + direccion + " Bright
      010%";
  PasoDeParametros param = new PasoDeParametros(comando);
      param.envioComando();
      listaEnl3.cambiarBrillo(direccion,50);

}
if(brillo == 50){
  String comando;
      comando = "cmd /c ahcmd sendplc " + direccion + " Bright
      010%";
  PasoDeParametros param = new PasoDeParametros(comando);
      param.envioComando();
      listaEnl3.cambiarBrillo(direccion,60);

}
if(brillo == 60){
  String comando;
      comando = "cmd /c ahcmd sendplc " + direccion + " Bright
      010%";
  PasoDeParametros param = new PasoDeParametros(comando);
      param.envioComando();
      listaEnl3.cambiarBrillo(direccion,70);

}
if(brillo == 70){
  String comando;
      comando = "cmd /c ahcmd sendplc " + direccion + " Bright
      010%";
  PasoDeParametros param = new PasoDeParametros(comando);
      param.envioComando();
      listaEnl3.cambiarBrillo(direccion,80);

}
if(brillo == 80){
  String comando;
      comando = "cmd /c ahcmd sendplc " + direccion + " Bright
      010%";
       PasoDeParametros param = new PasoDeParametros(comando);
      param.envioComando();
      listaEnl3.cambiarBrillo(direccion,90);

}

if(brillo == 90){
  String comando;
  comando = "cmd /c ahcmd sendplc " + direccion + " On";
  PasoDeParametros param = new PasoDeParametros(comando);
  param.envioComando();
  listaEnl3.cambiarBrillo(direccion,100);

}
if(brillo == 100){
  String cadena2 = "El incremento es total, 100%";
  JOptionPane.showMessageDialog(null, cadena2);
```

```
      }
       }
       else{
        String cadena;
        cadena = "Este dispositivo no permite incrementar el brillo" +
    "\n";
        JOptionPane.showMessageDialog(null, cadena);
       }
    }
 }
 Mas = false;

}

if(Menos){
  nodo2 = listaEnl3.buscarSeleccion();
  if(nodo2 == null){
    String cadena2 = "No se ha seleccionado ningún dispositivo" + "\n";
    JOptionPane.showMessageDialog(null, cadena2);
 }
 else{
    if(nodo2.seleccionado){
       String direccion = nodo2.codCasaDisp;
       int brillo = nodo2.porcBrillo;
       String tipo = nodo2.tipo;
       if(tipo.equals("Dim y Bright")){
      if(brillo == 100){
        String comando;
        comando = "cmd /c ahcmd sendplc " + direccion + " Dim 010%";
        PasoDeParametros param = new PasoDeParametros(comando);
        param.envioComando();
        listaEnl3.cambiarBrillo(direccion,90);

      }
          if(brillo == 90){
        String comando;
        comando = "cmd /c ahcmd sendplc " + direccion + " Dim 010%";
        PasoDeParametros param = new PasoDeParametros(comando);
        param.envioComando();
        listaEnl3.cambiarBrillo(direccion,80);

      }
          if(brillo == 80){
        String comando;
        comando = "cmd /c ahcmd sendplc " + direccion + " Dim 010%";
        PasoDeParametros param = new PasoDeParametros(comando);
        param.envioComando();
        listaEnl3.cambiarBrillo(direccion,70);

      }
          if(brillo == 70){
        String comando;
        comando = "cmd /c ahcmd sendplc " + direccion + " Dim 010%";
        PasoDeParametros param = new PasoDeParametros(comando);
        param.envioComando();
        listaEnl3.cambiarBrillo(direccion,60);

      }
          if(brillo == 60){
        String comando;
```

```
  comando = "cmd /c ahcmd sendplc " + direccion + " Dim 010%";
  PasoDeParametros param = new PasoDeParametros(comando);
  param.envioComando();
  listaEnl3.cambiarBrillo(direccion,50);

}
   if(brillo == 50){
  String comando;
  comando = "cmd /c ahcmd sendplc " + direccion + " Dim 010%";
  PasoDeParametros param = new PasoDeParametros(comando);
  param.envioComando();
  listaEnl3.cambiarBrillo(direccion,40);

}
   if(brillo == 40){
  String comando;
  comando = "cmd /c ahcmd sendplc " + direccion + " Dim 010%";
  PasoDeParametros param = new PasoDeParametros(comando);
  param.envioComando();
  listaEnl3.cambiarBrillo(direccion,30);

}
   if(brillo == 30){
  String comando;
  comando = "cmd /c ahcmd sendplc " + direccion + " Dim 010%";
  PasoDeParametros param = new PasoDeParametros(comando);
  param.envioComando();
  listaEnl3.cambiarBrillo(direccion,20);

}
   if(brillo == 20){
  String comando;
  comando = "cmd /c ahcmd sendplc " + direccion + " Dim 010%";
  PasoDeParametros param = new PasoDeParametros(comando);
  param.envioComando();
  listaEnl3.cambiarBrillo(direccion,10);

}
   if(brillo == 10){
  String comando;
  comando = "cmd /c ahcmd sendplc " + direccion + " Off";
  PasoDeParametros param = new PasoDeParametros(comando);
  param.envioComando();
  listaEnl3.cambiarBrillo(direccion,0);
  listaEnl3.cambiarEstado(direccion, false);

}

if(brillo == 0){
  String cadena2 = "La atenuación de brillo es total, 0%";
  JOptionPane.showMessageDialog(null, cadena2);

}
 }
 else{
String cadena;
   cadena = "Este dispositivo no permite atenuar el brillo" +
   "\n";
JOptionPane.showMessageDialog(null, cadena);
 }
```

```
      }
    }
    Menos = false;
  }
}

public void onMotionSensingEvent (MotionSensingEvent movimiento){

Orientation orientacion = movimiento.getOrientation();

mov_Z = (int)orientacion.getPitch();
mov_Y = (int)orientacion.getRoll();

if(mov_Z <= -55 && B==true){
  System.out.println("Estoy GIRANDO el mando hacia ARRIBA");
  B = false;
  try{Thread.sleep(2000);}
  catch (InterruptedException e){}
}
else if(mov_Z > 60 && B==true){
  System.out.println("Estoy GIRANDO el mando hacia ABAJO");
  B = false;
  try{Thread.sleep(2000);}
  catch (InterruptedException e){}
}
else if(mov_Y <= -50 && B==true){
  try{Thread.sleep(2000);}
  catch (InterruptedException e){}
}
else if(mov_Y > 50 && B==true){
  try{Thread.sleep(2000);}
  catch (InterruptedException e){}
}
}



public void onExpansionEvent (ExpansionEvent e){}


public void onStatusEvent (StatusEvent e){}


public void onDisconnectionEvent (DisconnectionEvent e){}



public void onNunchukInsertedEvent (NunchukInsertedEvent e){}


public void onNunchukRemovedEvent (NunchukRemovedEvent e){}


public void onGuitarHeroInsertedEvent (GuitarHeroInsertedEvent e){}


public void onGuitarHeroRemovedEvent (GuitarHeroRemovedEvent e){}
```

```
public void onClassicControllerInsertedEvent
(ClassicControllerInsertedEvent e){}

public void onClassicControllerRemovedEvent
(ClassicControllerRemovedEvent e){}
}
```