UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA SUPERIOR DE TELECOMUNICACIÓN

PROYECTO FINAL DE CARRERA

DESARROLLO DE UNA LIBRERÍA DE GESTIÓN DE REDES DOMÓTICAS X10 EN JAVA J2SE 1.6

AUTOR William Wallace San Paulo

TUTOR José Ignacio Moreno Novella

Octubre 2010

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1 - INTRODUCCIÓN Y OBJETIVOS

INTRODUCCIÓN Y MOTIVACIÓN

El término domótica se puede definir como la disciplina o técnica que busca conseguir la automatización de una vivienda, aportando así una serie de servicios como pueden ser la gestión energética, la seguridad, el confort, las comunicaciones, etcétera, utilizando para ello redes interiores y exteriores de comunicación, cableadas o inalámbricas, proporcionando una gran flexibilidad en el control de todo el sistema desde dentro y fuera del hogar.

Para entender mejor la penetración de la domótica y sus ventajas, primero se va a realizar un pequeño estudio sobre el consumo y producción de energía eléctrica en España.

La figura 1 tenemos la gráfica con los datos anuales de los últimos quince años del consumo y producción eléctrica en España. El eje de ordenadas contiene MWh anuales, en positivo es la energía consumida en España mientras que en negativo son las exportaciones. (Neida Boscán)

Lo primero que llama la atención, al (Java Main Website)observar la gráfica siguiente, es el enorme crecimiento del consumo eléctrico durante estos años. En quince años el consumo eléctrico ha aumentado casi un 70%. En ese mismo periodo la población solo ha aumentado un 15%, por lo que se puede afirmar que se consume mucha más energía por persona.

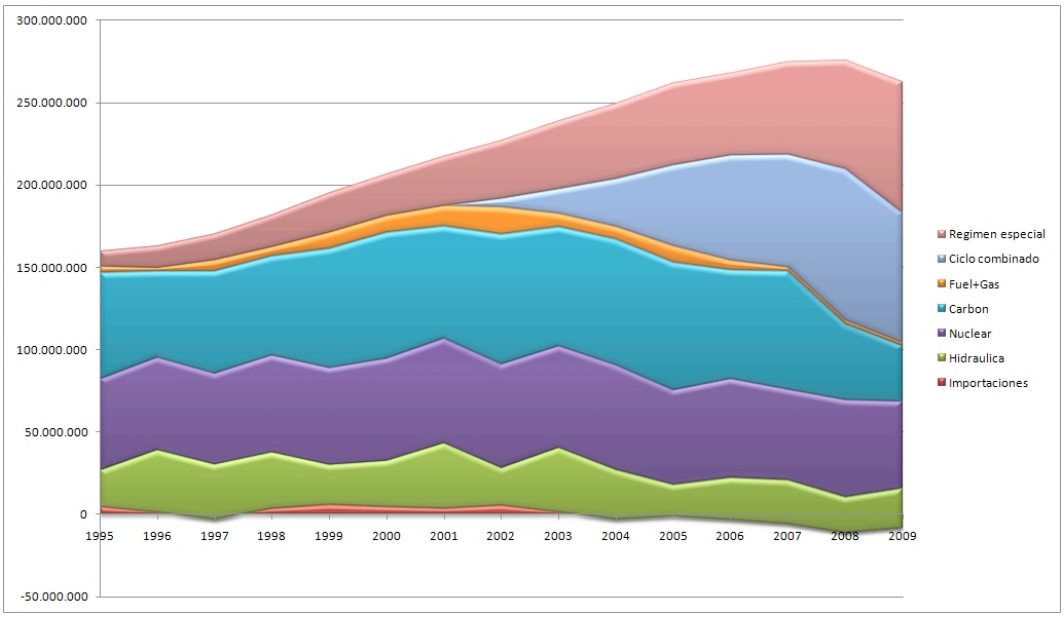

Figura 1 - Evolución del consumo eléctrico en España

El creciente consumo de energía y la limitación de los recursos energéticos generan efectos negativos en el medio ambiente que se reflejan en dos aspectos:

- Económico: los precios de la energía tienden a subir, por lo que un control del consumo energético incrementa significativamente el ahorro para el usuario.

- Ecológico: el usuario puede disminuir el impacto negativo sobre su entorno si disminuye su consumo de energía.

La gestión energética de una vivienda es cada día más compleja, ya que como hemos visto el consumo es cada vez mayor. Las viviendas poseen cada vez más electrodomésticos, equipos informáticos y audiovisuales, sistemas de iluminación, calefacción, aire acondicionado, etc. A continuación veremos que una forma de ahorrar energía, haciendo un uso más eficiente de esta, puede ser instalaciones domóticas en los hogares.

Para consumir de forma eficiente la energía es necesario conocer como la estamos consumiendo y establecer estrategias para su utilización. En este punto entra la domótica, proporcionando información sobre el consumo y ayudándonos a gestionar la energía mejor.

La domótica está enfocada a la utilización de los componentes del hogar cuando es única y exclusivamente necesario, es decir, se puede afirmar que como consecuencia se consigue un ahorro energético y un aprovechamiento racional de la energía para la vivienda. Por lo que gracias a la dómótica, realizando la gestión eficiente de la energía se consigue como resultado el ahorro energético.

Las aplicaciones domóticas más utilizadas y su forma de ahorro son por ejemplo:

- La **iluminación**: Así sólo se encenderán las luces cuando sea necesario. Para ello se pueden utilizar sensores de movimiento, que activen la luz cuando se detecte presencia en la habitación. También se puede regular la intensidad. Además para casas con jardín y porche, se puede programar que permanezcan encendidas durante un espacio de tiempo determinado.

- La **programación** de ciertos electrodomésticos haciendo que por ejemplo la lavadora o el lavavajillas funcionen cuando la tarifa energética es más reducida.

- **Los sistemas de climatización**, se puede regular la temperatura de cada estancia de manera individual. Los nuevos dispositivos tienen también la opción de rebajar la temperatura interior si aumenta el calor en la calle. El control remoto permite regular el termostato a distancia con una sola llamada telefónica.

- **Control de toldos, cortinas y persianas** para que aun estando fuera del hogar se pueda conseguir un mejor aprovechamiento de la luz y el calor del sol.

- **Gestión del jardín y del control del riego** por sensores meteorológicos, incluso para distintas parcelas, o sistemas de riego.

- **Sistemas de vigilancia y seguridad** que gestionan la protección y aviso en caso de alarma, o la vigilancia remota pudiendo visualizar el estado de la vivienda desde cualquier dispositivo móvil. También se puede automatizar el accionamiento de rejas protectoras y cierres a las horas deseadas.

De entre todas las tecnologías presentes en el mercado, X10 es la más veterana y limitada de ellas, y aunque nuevos desarrollos como ZigBee o tecnologías más veteranas y con gran aceptación en el mercado como LonWorks o KNX/EIB son más potentes y ofrecen una mayor flexibilidad en la configuración de entornos domóticos, X10 sigue siendo extremadamente popular especialmente en el mercado de bricolaje y soluciones que requieren de mínima configuración y sencillez.

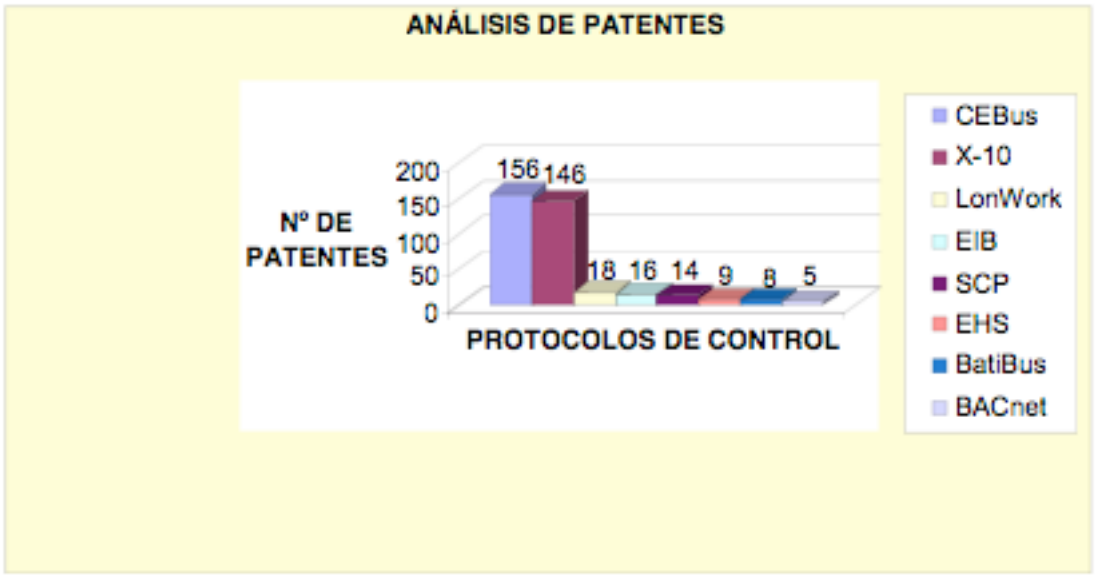

Figura 2 - Patentes presentas en 2009 por tecnología domótica

Como muestra la anterior figura, durante el año 2009 se presentaron casi tantas patentes relacionadas con X10 como en CEBus, y muchas más de las relacionadas con otras teconlogías que ya forman parte del conglomerado KNX/EIB o LonWorks. Además, cabe notar que tanto KNX/EIB como LonWorks requieren del cableado completo de la vivienda, y por eso sólo son populares en entornos empresariales e industriales en los que el gasto de este despliegue se compensa por el ahorro energético.

Aún más, y como pude observar en el primer congreso de redes domóticas celebrado en la Universidad Pierre et Marie Curie de París en 2007, las alternativas a X10 tienden a centrarse en entornos residenciales de clase alta, en las que las necesidades de automatización son más evidentes y es muy superior la cantidad de dispositivos a controlar (elementos de riego, decenas de persianas y lumínicas, sistemas de seguridad integrados, etcétera).

En aquel congreso y en este proyecto se presenta X10 como una solución de bajo coste, fácil despliegue y configuración, y de suficiente popularidad como para que su instalación y manejo sean inmediatamente intuitivos.

Sin embargo, la sencillez de X10 acaba en su instalación, y el control de estas redes mediante el software comercial actual es muy limitada. Aún proyectos Open Source como The Java X10 project aducen de una innecesaria complejidad en el desarrollo de aplicaciones que deseen comunicarse con estas redes, así como una fuerte limitación en cuanto a monitorización y fiabilidad. De hecho, fue la limitación de este API en concreto la primera motivación para el desarrollo de un API de control que internalizase los principios de simplicidad y funcionalidad de X10. Aún más, la futura penetración en el mercado de soluciones basadas en ZigBee (802.15.4) serían incompatibles con las redes X10 ya presentes en los hogares de los early adopters de la domótica, y se presentaba la necesidad de sistemas de control ampliamente compatibles con soluciones de terceros.

Es por lo tanto el objetivo de este proyecto el presentar una solución de software que permita una comunicación con redes X10 igual de sencilla que lo es su instalación y despliegue. Además, este proyecto permitirá el control y monitorización de estas redes desde cualquier dispositivo, no sólo desde el propio PC conectado a la red sino desde una enorme variedad de dispositivos, y al implementar el acceso a esta red como un servicio, su integración con redes domóticas alternativas serán extremadamente simple.

OBJETIVOS

Como hemos visto, X10 presenta ciertas ventajas con respecto al resto de tecnologías domóticas del mercado:

- No requiere del cableado de la vivienda, puesto que hace uso de la red eléctrica para sus comunicaciones. Esto a la vez que una virtud es un defecto, puesto que la red eléctrica no está pensada para la transmisión de información.

- Su instalación, despliegue y configuración son muy sencillos, y no son necesarios conocimientos técnicos para instalar una red X10 sencilla.

- Es muy fácil comprender la nomenclatura de las direcciones X10 proporcionada a los dispositivos: una letra entre la A y la P representa el código de área, mientras que un número entre el 1 y el 16 representa el código del dispositivo dentro del área.

- El coste de los adaptadores necesarios es muy bajo, debido en especial a la veteranía de esta tecnología.

Figura 3 - Módulo X10 de control de lumínicas

Sin embargo, X10 presenta algunas limitaciones que trataremos más en detalle en los próximos capítulos, en especial a las que trataremos de dar solución son:

- El software comercial que suele acompañar a las instalaciones X10 es ActiveHome, un programa muy antiguo (1998) que aún en su época estaba desfasado. Este programa no permite el acceso directo a la red, no presenta un uso amigable y no presenta ningún tipo de soluciones de integración con software de terceros. Además, la conexión a la red X10 debe realizarse desde un ordenador directamente conectado a ella, sin posibilidad de acceso remoto o desde ordenadores que no tengan Windows instalado.

- El protocolo inalámbrico X10 Wireless está completamente desfasado, y sigue un estándar cerrado no apoyado por el IEEE (al igual que X10 PLC). La entrada en el mercado de soluciones inalámbricas como ZigBee hacen deseable la comunicación entre redes X10 y estas nuevas soluciones, puesto que el nicho de mercado de "early adopters" es similar.

El objetivo de este proyecto es, por tanto, el diseñar un API de desarrollo de aplicaciones X10 que traslade la simplicidad del hardware y la red X10 al software de control y monitorización de estas redes, supliendo las carencias del software comercial actual. En especial, trataremos el acceso remoto a la red X10 de la forma más simple posible, para que nuestra solución sea compatible con aplicaciones programadas en cualquier lenguaje de programación y ejecutadas sobre la mayor variedad posible de dispositivos (teléfonos móviles, PDA, diversos sistemas operativos).

Como veremos más adelante, el acceso a los dispositivos de pasarela hacia la red X10 y su comunicación con estos no es evidente, y el protocolo de comunicación es a menudo contra intuitivo y y errático. Transformar este acceso difícil en una solución elegante y lo más simple y compatible posible es nuestro gran objetivo.

El segundo objetivo será diseñar el API de cara a una futura integración con otras soluciones domóticas, implementando éste como un servicio accesible de forma independiente y ejecutado en segundo plano, permitiendo el acceso simple y transparente a la red X10 y su futura integración con otras redes inalámbricas como ZigBee, Bluetooth o incluso WiFi.

MEDIOS MATERIALES

Como parte del acuerdo del laboratorio del departamento de Telemática de la Universidad Carlos III de Madrid y la empresa Nlaza, ésta instaló el entrenador domótico mostrado en la Figura 4.

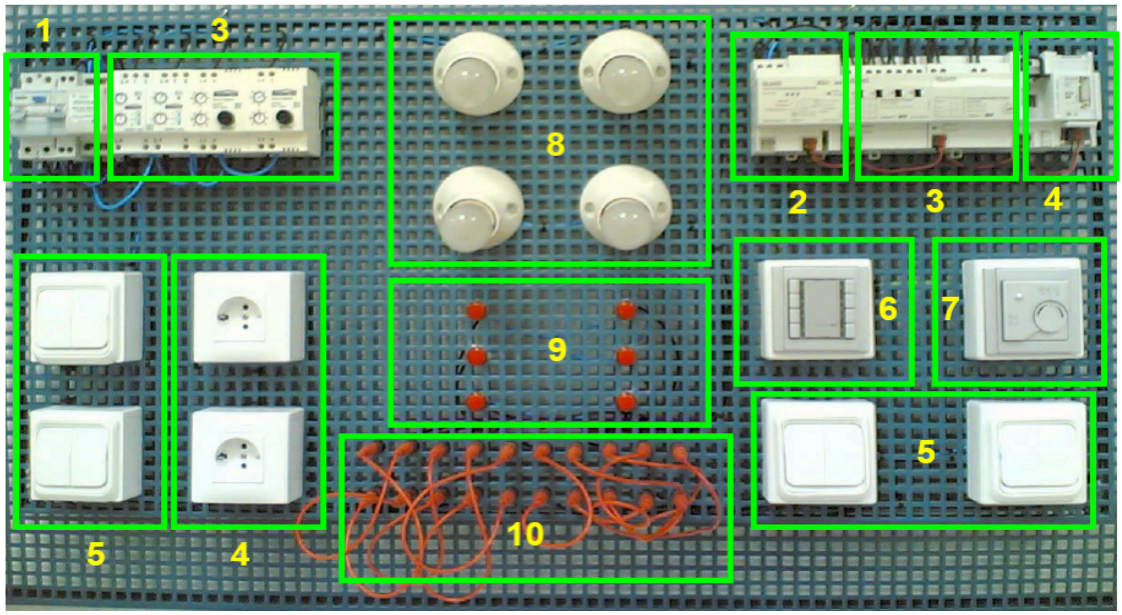

Figura 4 - Entrenador X10 - KNX/EIB proporcionado por NLaza

Este entrenador está formado por dos secciones independientes: una red de elementos X10 y otra de elementos KNX/EIB. La sección que interesa en este proyecto es la formada por elementos X10:

1. Elemento de protección común a todo el panel.

3. Cuatro actuadores. Dos de ellos para el control de elementos lumínicos, dos para el control de dispositivos con posiciones de apagado y encendido. Detallados en la Tabla 1 - Listado de componentes X10 utilizados.

4. Dos enchufes que servirán de nodos de comunicación con la red. En cualquier de ellos puede conectarse el módulo de comunicaciones CM11A.

5. Cuatro interruptores pre-configurados para actuar sobre dos bombillas y dos LED. Estos interruptores incluyen el módulo TMD4 detallado en la Tabla 1 - Listado de componentes X10 utilizados.

8. Bombillas controladas por la red X10 y por la red KNX. Las dos de la izquierda son las que pertenecen a la red X10.

9. LED que representan elementos simples en la red, con sólo dos acciones posibles sobre ellos (apagado y encendido).

Elemento	Descripción
 **Figura 5 - Interruptor Marmitek AD10 DIN**	Módulo que responde a comandos básicos de apagado y encendido. (ON, OFF, ALL ON y ALL OFF)
 **Figura 6 - Atenuador de brillo Marmitek LD11 DIM**	Permite el control de los dispositivos lumínicos actuando frente a comandos básicos de encendido, apagado y control de intensidad.
Micro Módulo TMD4  **Figura 7 - Micromódulo Marmitek TMD4**	Convierte hasta 4 interruptores convencionales en interruptores compatibles con X10 capaces de enviar y recibir comandos de encendido, apagado y control de brillo.

Elemento	Descripción
Interfaz CM11A  **Figura 8 - Interfaz de comunicación Marmitek CM11A**	Este módulo permite la gestión, monitorización y comunicación entre redes X10 y otros dispositivos a través del puerto serie o USB.
 **Figura 9 - Filtro de señales X10**	Este módulo filtro para carril DIN está diseñado para evitar todo tipo de interferencias. Se instala en el cuadro eléctrico principal de la vivienda a continuación del diferencial principal y antes de los magnetotérmicos, de esta forma garantiza el aislamiento de todo el circuito de la vivienda, no solo de las perturbaciones externas sino también de las que pudiera originar su propia instalación.

Tabla 1 - Listado de componentes X10 utilizados

Conectado al módulo CM11A disponemos de un ordenador personal con sistemas operativos Windows XP SP2 y Ubuntu 7.10 instalados. La configuración completa de este equipo puede consultarse en el Anexo 1 - Configuración del entorno de desarrollo.

El esquema completo de la plataforma de desarrollo puede verse en la Figura 10 - Esquema de interconexión básica de elementos, mientras que el esquema de pruebas de toda la plataforma puede consultarse en la Figura 11 - Esquema completo de interconexión de elementos.

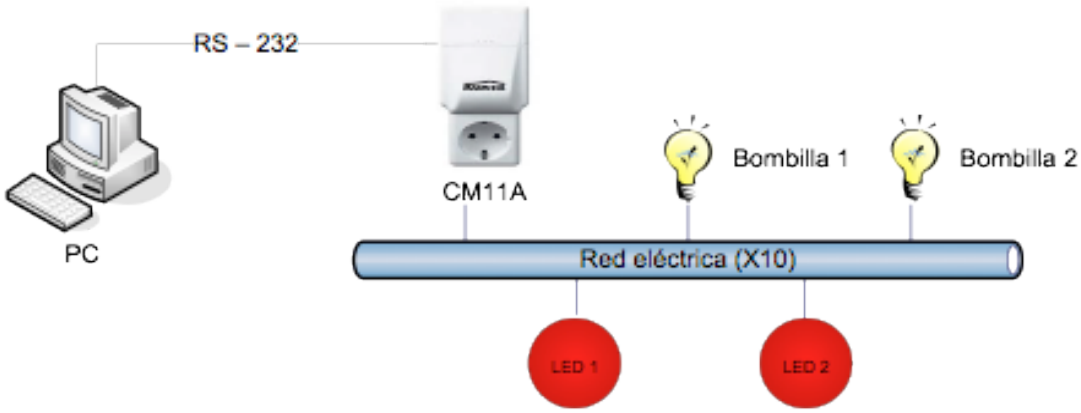

Figura 10 - Esquema de interconexión básica de elementos

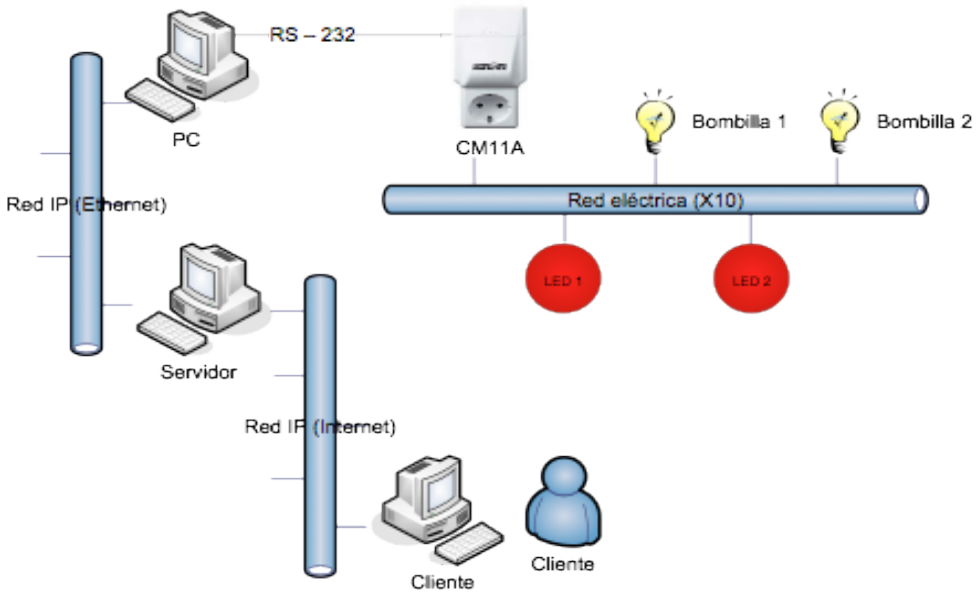

Figura 11 - Esquema completo de interconexión de elementos

ORGANIZACIÓN DE LA MEMORIA

Fase 1. Estudio y Documentación sobre la tecnología X10 y el protocolo de comunicación CM11A

- Breve Introducción a la historia de X10.
- Descripción técnica de X10 PLC.
- Descripción técnica de X10 Wireless.
- Descripción del protocolo de comunicaciones a través de puerto serie con el CM11A.

Fase 2. Descripción del API de desarrollo de aplicaciones y servicios X10.

- Breve introducción al escenario de desarrollo.
- Descripción de los módulos del API y ventajas con respecto a alternativas comerciales y de código abierto.
- Descripción de las posibilidades de desarrollo que abre el API

Fase 3. Desarrollo de aplicaciones de ejemplo

- Desarrollo de una aplicación de control y monitorización local.
- Desarollo de una aplicación de control y monitorización remota.
- Observaciones sobre uso concurrente de estas aplicaciones y posibilidades.

- Presentación de un escenario práctico simple de integración con 802.15.4 y del desarrollo teórico de una integración más completa.

Fase 4. Conclusiones y trabajos futuros

- Presentación final sobre ventajas e inconvenientes de este API.

- Ampliaciones futuras, en especial de integración con redes 802.15.4 y ZigBee.

CAPÍTULO 2 - ESTADO DEL ARTE

INTRODUCCIÓN A X10

HISTORIA

En 1970 se funda la empresa Pico Electronics en Glenrothes, Escocia. El objetivo de esta compañía es el desarrollo de soluciones electrónicas que reduzcan el coste o amplíen la funcionalidad de dispositivos electrónicos populares en el mercado de la época, como calculadoras o reproductores de vinilo. Su primer proyecto es el desarrollo de un microchip que englobe toda la funcionalidad, a nivel de lógica, necesaria para el funcionamiento de una calculadora de bolsillo. En aquel momento, las calculadoras necesitaban de al menos 5 microchips, utilizar sólo uno reducía considerablemente su coste de fabricación, y varias empresas se interesaron por este proyecto, tales como Casio, Bowmar y Litton. (Rye)

Sin embargo, para 1974 el coste de fabricación de estos circuitos integrados se reduce de 20$ a menos de 1$, lo que lleva a Pico Electronics a plantearse el desarrollo de productos finales, y no sólo de circuitos integrados con poco margen de beneficios.

De los mayores éxitos de la compañía, X-9 tenía como objetivo el desarrollo de un selector de pistas en vinilos LP. Sin embargo, este proyecto era demasiado ambicioso como para que Pico Electronics lo llevase a cabo por sí sola, y en asocación con el mayor productor de reproductores de vinilo del momento, BSR, fundaron la joint venture Accutrac Ltd. Este reproductor de vinilos incorporaba no sólo la novedad de poder seleccionar pistas dentro de un vinilo, sino un control remoto conocido como “clicker” que funcionaba en la frecuencia de ultrasonidos. Fue este proyecto el que inspiró a los ingenieros de Pico Electronics a desarrollar un sistema de control remoto y automatización para distintos elementos del hogar, desde alarmas hasta control de luces, persianas, aspersores o climatización. Este sería su décimo proyecto, conocido como X10.

Desarrollado por Pico Electronics en 1975, X10 es una tecnología de transmisión de datos por la red eléctrica. Su principal objetivo es permitir el control remoto y automatización de distintos elementos dentro del hogar, tales como persianas, luces, alarmas y algunos electrodomésticos. Dadas sus limitaciones, de las que hablaremos en detalle más adelante, su aplicación a entornos industriales es complicada, y de hecho existen otras tecnologías en el mercado muy superiores en este aspecto. Esta tecnología fue finalmente comercializada al público en 1978 bajo varios nombres dependiendo del distribuidor. Debido a su éxito inicial, Pico Electronics vuelve a aliarse con BSR para fundar una empresa que tenga la capacidad de no sólo desarrollar el producto, sino también de distribuirlo. Esta empresa continua existiendo hoy en día bajo el nombre X10 Ltd., y su producto estrella se vende como "X10 PowerHouse System", o como es más conocido popularmente: X10.

A pesar de su longevidad, o quizás precisamente por ello, X10 sigue siendo una tecnología popular hoy en día debido a su bajo coste y fácil despliegue. Su uso continúa extendiéndose en la automatización de pequeños hogares y apartamentos, así como en PYMES donde es necesario el control y monitorización de pequeños clúster de equipos, tales como elementos informáticos, luces o climatización.

DESCRIPCIÓN TÉCNICA

TRANSMISIÓN Y CODIFICACIÓN DE LA INFORMACIÓN

Debido a su longevidad, la codificación de datos en X10 no es binaria per se, y la codificación de la información es única de X10, sin que se haya extendido como estándar en otras aplicaciones o tecnologías. (X10 Ltd.)

X10 no permite la transmisión de datos entre distintos dispositivos, es decir, no permite crear una red local de datos como Ethernet. Su único objetivo es permitir que un dispositivo dentro de la red pueda enviar un comando a otro y que este ejecute la acción pedida, como encenderse, apagarse o modificar su

comportamiento de forma gradual. Por ello, el bit no es la menor unidad de información, sino el comando en sí, que como veremos más adelante, casi siempre sigue el mismo formato.

Sin embargo, actualmente las telecomunicaciones toman el bit como mínima medida de información, y será interesante hacer esta analogía a partir de este punto para entender mejor el funcionamiento de las redes X10.

En X10, la codificación binaria se realiza mediante un pulso de 120 KHz y 1 ms de duración en el cruce por cero de la señal de corriente alterna de la red eléctrica de 50 Hz en Europa y 60 Hz en Estados Unidos de América, con un offset máximo con respecto al cruce por cero de 200 µs. En la Figura 12 - Ilustración de pulso X10 para 1 binario, se muestra la distinta codificación para unos y ceros. El uno se codifica mediante un pulso de 1 ms de duración en el primer cruce por cero y nada en el segundo, mientras que el cero se representa mediante la ausencia de este pulso. En recepción, la ventana de aceptación se encuentra aproximadamente entre 200 µs y 900 µs. Si durante este periodo se reciben más de 48 ciclos de señal de portadora a 120 KHz se considera que ha llegado un uno, de lo contrario se interpreta el bit como cero.

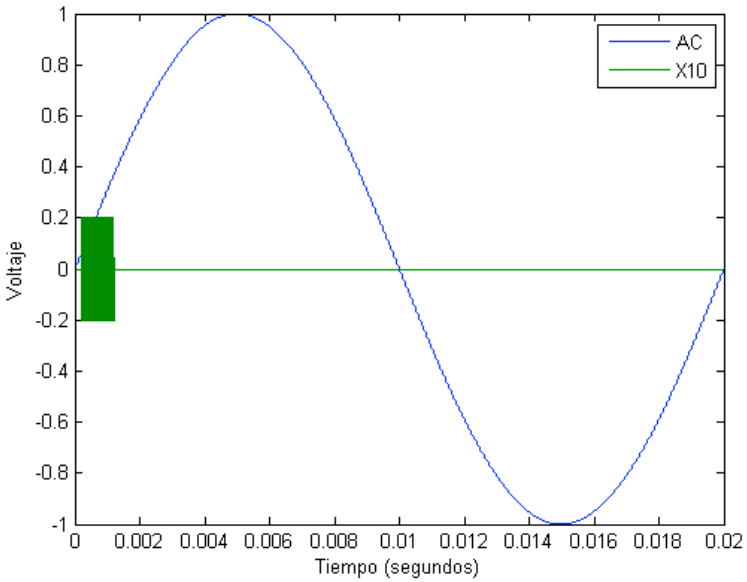

Figura 12 - Ilustración de pulso X10 para 1 binario

DIRECCIONAMIENTO

En X10 existen un total de 16 comandos, de los cuales 13 son los llamados comandos básicos o comandos no extendidos. Estos 13 comandos permiten emitir órdenes básicas a los distintos dispositivos de la red X10, bien sea a uno en concreto, a todos ellos o a un determinado grupo. (X10 Ltd.)

Para entender el funcionamiento de los comandos, es necesario entender cómo se organizan los dispositivos dentro de una red X10. Cada dispositivo posee una dirección única formada por su código de área o de hogar, y su código de dispositivo, ambos formados por 4 bits para una dirección completa codificada en 8 bits, permitiendo por lo tanto que hasta 16 dispositivos coexistan dentro de una misma subred, y que se puedan formar hasta 16 subredes dentro de un mismo hogar o red eléctrica. Para facilitar el uso y configuración de estos dispositivos, se utiliza una notación alfabética para los códigos de área y numérica para la notación de dispositivos con las equivalencias binarias que se muestran en la Tabla 2 - Equivalentes binarios a los códigos X10 de zona y dispositivo

Los dispositivos actuadores, tales como interruptores o controles remotos, tienen asignada no una dirección propia, sino la dirección del dispositivo sobre el que actúan, que denominaremos pasivo. Por ello, en una misma área pueden coexistir hasta 16 dispositivos pasivos, y tantos actuadores como deseemos.

El código de área u hogar fue diseñado para evitar interferencias de redes X10 adyacentes que compartiesen la misma red eléctrica, o dicho de otra manera, evitar que cuando el vecino encendiese las luces del porche a nosotros se nos encendiesen los aspersores del jardín. Es aquí donde podemos ver una de las primeras limitaciones de esta tecnología: la enorme limitación en cuanto a número de dispositivos por red, pensados para ser un máximo de 16. Nada impide que varias zonas coexistan dentro de un mismo hogar o espacio de convivencia, sin embargo la gestión de este tipo de redes podría ser muy compleja, y se aleja del diseño inicial.

Código de Zona	Equivalente Binario	Código de dispositivo
A	0110	1
B	1110	2
C	0010	3
D	1010	4
E	0001	5
F	1001	6
G	0101	7
H	1101	8
I	0111	9
J	1111	10
K	0011	11
L	1011	12
M	0000	13
N	1000	14
O	0100	15
P	1100	16

Tabla 2 - Equivalentes binarios a los códigos X10 de zona y dispositivo

La Figura 13 muestra como dentro de un mismo hogar se han dividido los dispositivos en dos áreas: la zona C y la zona D. Los dispositivos de una zona no responderán a los comandos de los dispositivos de otra, y esto puede ser muy útil para evitar que comandos dirigidos a todos los dispositivos de una zona afecten a subgrupos que queremos que se comporten de forma independiente. Como podemos ver en la figura, los interruptores están emparejados con sendas luces, mientras que el enchufe C-5 no está asignado a ningún interruptor en concreto. Sin embargo, como veremos más adelante, el interruptor C-12 puede enviar comandos dirigidos a todos los dispositivos de su zona, como apagado o encendido general. De esta manera, al ejecutarse el comando de apagado general desde el interruptor, la luz C-12 y el enchufe C-5 de apagarían o desactivarían, mientras que la luz D-5 permanecería en el mismo estado.

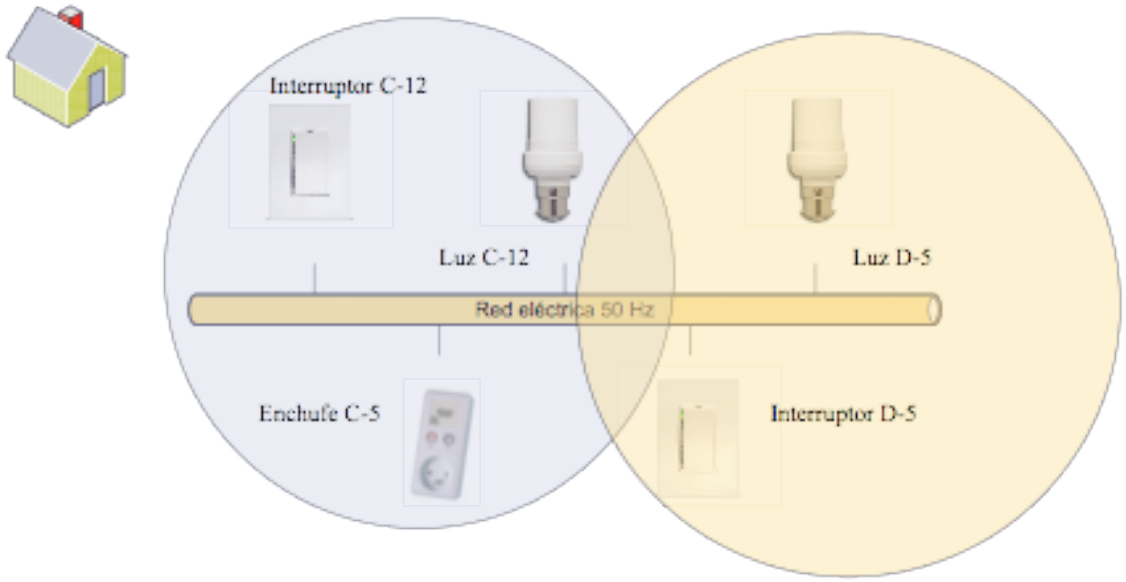

Figura 13 - Ejemplo de dos zonas conviviendo en la misma red eléctrica

COMANDOS BÁSICOS

En X10 existen un total de 15 comandos posibles, de los cuales 12 son comandos comunes a todas las redes X10 y que llamaremos comandos básicos o no extendidos, y 3 son comandos extendidos que dependen de una implementación en particular y que caen fuera del estándar de uso, dejándose a criterio de usuario la forma en la que usarlos. (X10 Ltd.)

Al haber 15 comandos, bastan 4 dígitos binarios para su codificación. La Tabla 3 - Listado completo de comandos X10 y equivalente binario, muestra el listado de comandos en redes X10 ordenados por código binario igual que en la tabla de códigos de zona y dispositivo.

Para evitar comportamientos anómalos en la red, se introduce un mecanismo de redundancia para evitar que se activen por error dispositivos que no deberían, o que se ejecute la orden incorrecta en el dispositivo de destino.

Cada comando se divide en dos tramas que se envían por duplicado, separadas por 3 ciclos de alterna. La primera trama es la trama de direccionamiento, que indica qué dispositivo deberá activarse, y su estructura es la mostrada en la Figura 14. En los casos en los que el comando esté dirigido a un conjunto de dispositivos, el código de dispositivo es irrelevante y será ignorado por todos los dispositivos. El bit de función indica si la trama es de direccionamiento (con valor cero) o de comando (con valor 1). El código de inicio lo veremos en detalle más adelante.

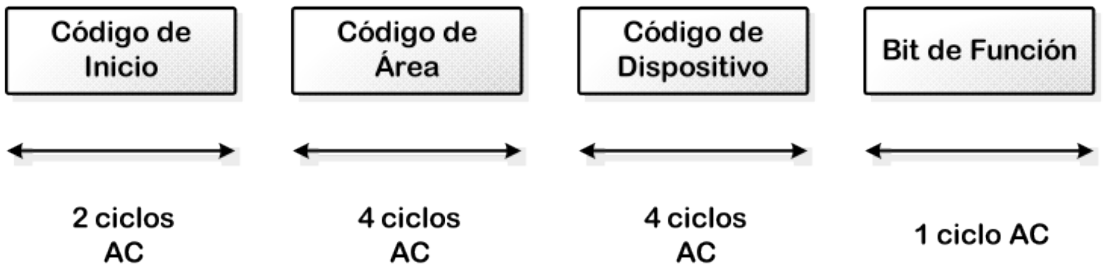

Figura 14 - Estructura de una trama de direccionamiento

La segunda trama, o trama de comando, sigue una estructura análoga a la anterior, aunque en este caso después del código de área se incluye el código del comando a ejecutar y el valor del bit de función es uno. Esta estructura se muestra en la **Error! Reference source not found.**.

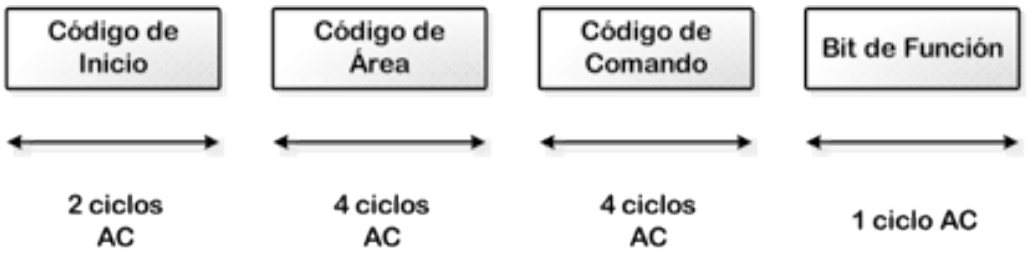

Figura 15 - Estructura de una trama de comando

Comando	Código Binario	Descripción
ALL LIGHTS OFF	0110	Apagado de todas las luces del área.
Status OFF	1110	Apagado del sistema de estado.
ON	0010	Encender o habilitar dispositivo.
EXT. COM. 3	1010	Comando extendido 3
ALL LIGHTS ON	0001	Encendido de todas las luces del área.
HAIL ACK.	1001	En respuesta a Hail Request, para evitar interferencias en un área.
BRIGHT	0101	Aumenta el brillo de una luz en una unidad de 25.
STATUS ON	1101	Encendido del sistema de estado.
EXT. COM. 1	0111	Comando Extendido 1
STATUS REQ.	1111	Petición de estado.
OFF	0011	Apagar o deshabilitar dispositivo.
NO USE	1011	Sin usar.
ALL OFF	0000	Apagar todos los dispositivos del área.
HAIL REQUEST	1000	Pide a los dispositivos del área que respondan con Hail Ack para evitar interferencias de otras redes.
DIM	0100	Atenuar luz en una unidad de 25.
EXT. COM. 2	1100	Comando extendido 2.

Tabla 3 - Listado completo de comandos X10 y equivalente binario

El código de inicio es un tanto particular, puesto que no utiliza la misma codificación que el resto de códigos de área, dispositivo o comando. En este caso, este código está formado por 3 pulsos de 120 KHz en los cruces por cero de la señal AC, seguidos de un cruce por cero sin pulso, tal y como se muestra en la Figura 16. Por lo tanto, aunque el código de inicio está formado por 3 pulsos de 120 KHz, son necesarios sólo 2 ciclos completos de la señal AC para transmitirlo. (X10 Ltd.)

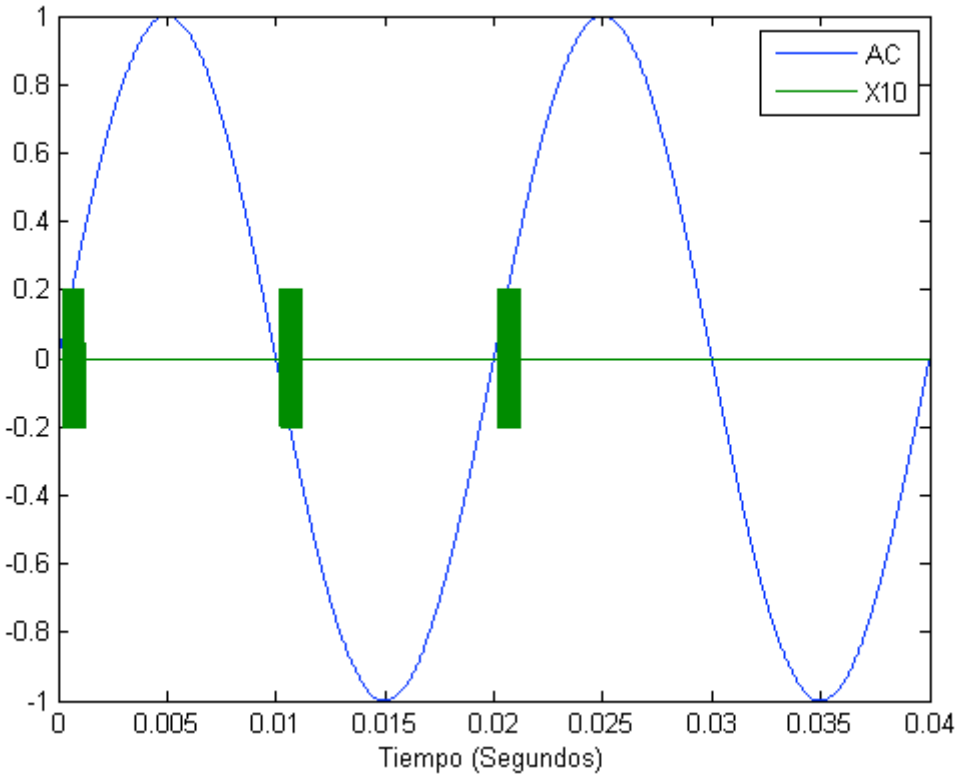

Figura 16 - Código de inicio de una trama X10

Como ya hemos dicho, cada una de estas tramas se envía por duplicado, dejando 3 ciclos de alterna vacíos. El proceso completo del envío de un comando se muestra en la Figura 17.

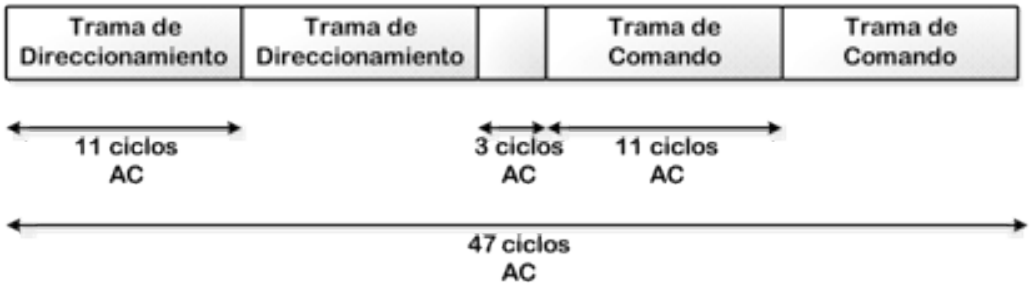

Figura 17 - Envío de comando X10

COMANDOS EXTENDIDOS

Los comandos extendidos tienen su propio formato de trama, mostrado en la "Figura 7 – Estructura Trama de Comando Extendido", de forma que son "invisibles" para aquellos dispositivos que no tienen la habilidad para responder a ellos. Más adelante detallaremos el funcionamiento del comando extendido 1, que permite ejecutar acciones mucho más complejas. (X10 Ltd.)

Por otro lado, el hecho de que los comandos extendidos sean muy superiores en longitud a los comandos básicos exige la implementación de un control de acceso al medio para evitar colisiones entre tramas. Algunos de los comandos extendidos son críticos para la seguridad, y por lo tanto se debe asegurar, en la medida de los posible, que llegaran a destino. Se implementa, para este caso, un sistema similar al usado en redes Ethernet, en el que el emisor escucha el canal de comunicaciones a la vez que envía el mensaje para comprobar si ha habido una colisión. Primero, el emisor espera una cantidad aleatoria de cruces por cero de la señal AC, entre 8 y 10. Después, comienza a enviar el mensaje, y en caso de colisión vuelve a empezar el proceso. La colisión se detecta comprobando que en los cruces por cero en los que se debería enviar el bit 0 no aparece un pulso de 120 KHz debido a la interferencia de otra trama. Por supuesto, este sistema de detección de colisiones no es perfecto, puesto que aunque garantiza que el

mensaje del emisor se ha enviado correctamente, no garantiza que no haya ocurrido una colusión que haya destruido o corrompido otro mensaje.

El comando extendido 2 sigue la misma estructura que los comandos básicos, salvo que la segunda trama viene seguida de varios bytes de datos, por ejemplo temperatura. Este comando sirve únicamente para que un dispositivo envíe información a otro, como por ejemplo sensores de temperatura y humedad. La estructura de la primera trama del comando extendido 2 es análoga a la de los comandos básicos, y la de la segunda trama es la mostrada en la Figura 18. El contenido de los datos y su uso en redes X10 se deja a criterio del diseñador, proponiendo este esquema sólo como una sugerencia en el intercambio de datos entre dispositivos en casos que no cubran el resto de comandos del estándar.

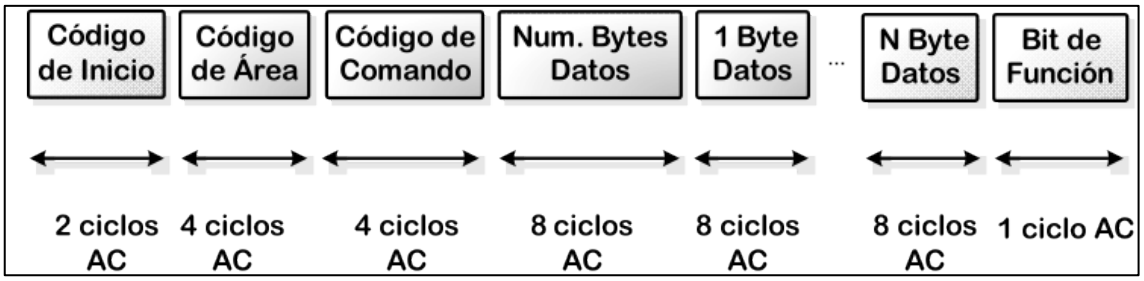

Figura 18 - Estructura de trama para comandos de tipo 2

El comando extendido 3 se deja a criterio del diseñador por completo. Siguiendo el mismo esquema que el comando extendido 2, su uso se centra en intercambios de información que afecten a la seguridad de la vivienda o de la red, como por ejemplo la activación de la alarma o de sensores asociados a ella.

El comando extendido 1, sin embargo, se detalla con minuciosidad en el estándar y su uso está regulado. El formato de tramas es diferente de todos los anteriores, y en este caso se envía una única trama por duplicado que sigue el esquema mostrado en la Figura 19. El campo “Código Extendido” tiene el valor igual al código binario correspondiente al Comando Extendido 1 seguido del bit de función con valor 1.

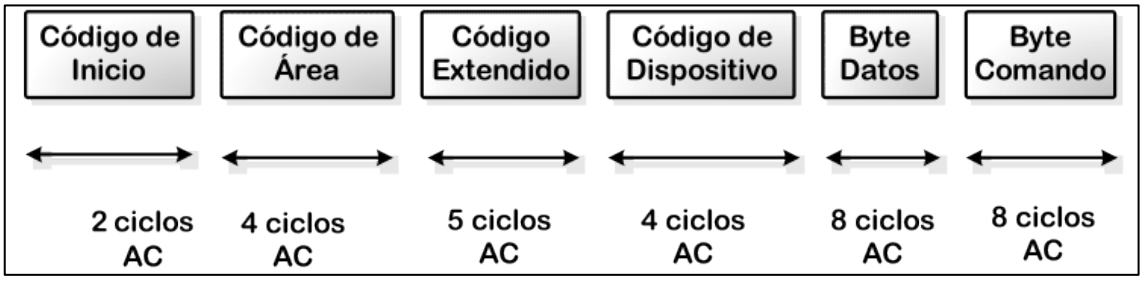

Figura 19 - Estructura de tramas para comandos extendidos de tipo 1

Los comandos extendidos se dividen en 6 grupos diferentes, que se detallan a continuación. El tipo de comando se indica con los 4 bits más significativos del byte de comando, el tipo de comando se codifica usando los restantes y el campo de Byte de Datos como apoyo cuando es necesario. En los siguientes apartados, se señala con X los bits ignorados y con D aquellos que tienen significancia para el comando.

En los siguientes apartados se detallan por medio de tablas todos los comandos disponibles en cada tipo de comando extendido. Los bits de las tablas que muestran estos comandos están representados empezando por el bit más significativo. En el caso de X10, cabe recordar que el bit más significativo es el primero en ser enviado / recibido, y que debido a la veteranía del sistema no se tuvo en consideración el dilema de utilizar Little Endian o Big Endian.

El tipo 2 de comando extendido 1 está reservado para comandos de seguridad, que el usuario puede rellenar a su propia discreción. Recordemos que la diferencia entre el comando extendido 1 y el 3, que mencionamos antes, estriba en que el comando extendido 1 se centra en el intercambio de datos entre dispositivos mediante pooling de datos. El comando extendido 3 está reservado para comandos de seguridad que no requieran de este intercambio.

TIPO 0 – CONTROL DE PERSIANAS Y CORTINAS

Byte Datos								Byte Comando								Descripción
X	X	X	D	D	D	D	D	0	0	0	0	0	0	0	1	Abrir persiana a nivel indicado por datos. Permitir protección solar.
X	X	X	D	D	D	D	D	0	0	0	0	0	0	1	0	Limitar grado de apertura a indicado en byte datos para protección solar.
X	X	X	D	D	D	D	D	0	0	0	0	0	0	1	1	Abrir persiana sin tener en cuenta protección solar.
X	X	X	X	X	X	X	X	0	0	0	0	0	1	0	0	Abrir todas las persianas en el código de área actual.
X	X	X	X	X	X	X	X	0	0	0	0	0	1	0	1	Abrir todas las persianas de todas las áreas.
L4	L2	L1	D	D	D	D	D	0	0	0	0	0	1	1	1	Incluir esta unidad en el estilo de vida indicado por L4, L2 y L1. D es el grado de apertura cuando se active el estilo de vida.
L4	L2	L1	X	X	X	X	X	0	0	0	0	1	0	0	0	Ejecuta el modo de vida indicado por L4, L2 y L1.
L4	L2	L1	X	X	X	X	X	0	0	0	0	1	0	0	1	Borrar unidad de estilo de vida indicado por L4, L2 y L1.
X	X	X	X	X	X	X	X	0	0	0	0	1	0	1	0	Excluir unidad de todos los estilos de vida.
X	X	X	X	X	X	X	X	0	0	0	0	1	0	1	1	Cerrar todas las persianas en el área, activar protección solar.
X	X	X	X	X	X	X	X	0	0	0	0	1	1	0	0	Cerrar todas las persianas, activar protección solar.
X	X	X	X	X	X	X	X	0	0	0	0	1	1	1	0	Autocomprobación de dirección X10. Levantar persiana durante 1 segundo si correcto.
X	X	X	X	X	X	X	X	0	0	0	0	1	1	1	1	Autocomprobación de dirección Earom. Subir durante un segundo y bajar durante 1 segundo.

Tabla 4 - Listado de comandos extendidos de tipo 0

L4	L2	L1	Descripción	L4	L2	L1	Descripción
0	0	0	Al levantarse	1	0	0	Por la noche
0	0	1	Al marcharse	1	0	1	De vacaciones
0	1	0	Al volver	1	1	0	Especial 1
0	1	1	Al dormir	1	1	1	Especial 2

Tabla 5 - Modos de vida en comandos extendidos de tipo 0

TIPO 1 – CONTROL Y MONITORIZACIÓN DE SENSORES

Byte Datos								Byte Comando								Descripción
X	X	X	X	X	X	X	X	0	0	0	1	0	0	0	1	Petición de luz ambiente media.
X	X	X	X	X	X	X	X	0	0	0	1	0	0	1	0	Petición de temperatura instantánea.
X	X	X	X	X	X	X	X	0	0	0	1	0	0	1	1	Petición de estado.
X	X	X	X	X	X	X	X	0	0	0	1	0	1	0	0	Petición de luz instantánea.
X	X	X	X	X	X	X	X	0	0	0	1	0	1	0	1	Petición de temperatura media (últimos 16 minutos).
I2	I1	P	P	P	P	P	P	0	0	0	1	1	0	1	1	Datos de luz ambiente.
T	T	T	T	T	T	T	T	0	0	0	1	1	1	0	0	Datos de temperatura.
X	X	X	X	X	X	X	X	0	0	0	1	1	1	0	1	Datos de estado.

Tabla 6 - Listado de comandos extendidos de tipo 1

Cuando un sensor devuelve datos, incluye su propia dirección en los campos de área y dispositivo de la trama, puesto que no conoce los datos del dispositivo que realiza la petición para devolverle una respuesta, y espera que este sea el que monitorice la respuesta direccionada de esta manera. Esta es sólo otra pequeña muestra más de las limitaciones del sistema.

I2	I1	P32	P16	P8	P4	P2	P1	Datos de luz Ambiente
0	0	D	D	D	D	D	D	Rango de 0 – 630 en incrementos de 10.
0	1	D	D	D	D	D	D	Rango de 0 – 6300 en incrementos de 100
1	0	D	D	D	D	D	D	Rango de 0 – 63000 en incrementos de 1000
1	1	D	D	D	D	D	D	Rango de 0 – 630000 en incrementos de 10000

Tabla 7 - Datos de luminosidad

TIPO 3 – MÓDULOS DE CONTROL (ATENUADORES Y DISPOSITIVOS)

Byte Datos								Byte Comando								Descripción
G1	G0	0	X	X	X	X	X	0	0	1	1	0	0	0	0	Incluir en grupo G1G0 al nivel de salida actual.
G1	G0	1	X	S3	S2	S1	S0	0	0	1	1	0	0	0	0	Incluir en grupo G1G0 al nivel de salida actual.
T1	T0	B5	B4	B3	B2	B1	B0	0	0	1	1	0	0	0	1	Establecer valor por defecto.
G1	G0	B5	B4	B3	B2	B1	B0	0	0	1	1	0	0	1	0	Incluir en grupo G1G0, con el valor por defecto indicado.
X	X	X	X	X	X	X	X	0	0	1	1	0	0	1	1	Encender todas las unidades del área.
X	X	X	X	X	X	X	X	0	0	1	1	0	1	0	0	Apagar todas las unidades del área.
0	0	0	0	G3	G2	G1	G0	0	0	1	1	0	1	0	1	Eliminar unidad del/los grupo/s.
1	1	1	1	G3	G2	G1	G0	0	0	1	1	0	1	0	1	Eliminar unidades del área del/los grupo/s.
G1	G0	0	0	X	X	X	X	0	0	1	1	0	1	1	0	Ejecutar comando de grupo.
G1	G0	1	0	S3	S2	S1	S0	0	0	1	1	0	1	1	0	Ejecutar comando de grupo.
G1	G0	0	1	X	X	X	X	0	0	1	1	0	1	1	0	Apagar unidades del grupo.
G1	G0	1	1	S3	S2	S1	S0	0	0	1	1	0	1	1	0	Apagar unidades del grupo.
X	X	0	0	X	X	X	X	0	0	1	1	0	1	1	1	Petición estado a la unidad.
X	X	0	1	X	X	X	X	0	0	1	1	0	1	1	1	Petición de estado desde la unidad.
G1	G0	1	0	0	0	0	0	0	0	1	1	0	1	1	1	Petición de estado a las unidades del grupo.
G1	G0	1	1	S3	S2	S1	S0	0	0	1	1	0	1	1	1	Petición de estado a las unidades del grupo.
A1	A0	B5	B4	B3	B2	B1	B0	0	0	1	1	1	0	0	0	Ack de unidad a petición de estado.
G1	G0	B5	B4	B3	B2	B1	B0	0	0	1	1	1	0	0	1	Ack de grupo a petición de estado.
X	X	X	X	X	X	X	X	0	0	1	1	1	0	1	0	Ack de grupo a petición de estado.
X	X	X	X	X	X	C1	C0	0	0	1	1	1	0	1	1	Configurar módulos para ACK de mensajes extendidos (C0) y básicos (C1).
G1	G0	0	B/A	X	X	X	X	0	0	1	1	1	1	0	0	Aumentar brillo / atenuar luces en grupo.
G1	G0	1	B/A	S3	S2	S1	S0	0	0	1	1	1	1	0	0	Aumentar brillo / atenuar luces en grupo.

Tabla 8 - Listado de comandos extendidos de tipo 3

En los comandos de tipo 3 se hace uso de ciertos campos que antes no estaban presentes:

- Campo G:
  - Dirección del grupo de dispositivos codificada con 2 bits, un mismo dispositivo puede pertenecer a hasta 4 grupos diferentes.
- Campo S:
  - Referencia de grupo, las direcciones de grupo pueden ser relativas a una referencia de 4 bits.
- Campo B:
  - Uno significa encendido, cero apagado.
  - Para atenuadores, el campo B indica el nivel de brillo en 63 niveles.
    - Si se indica el máximo nivel de brillo, la lumínica se enciende inmediatamente a ese nivel.
    - Si se escoge otro nivel de brillo, la lumínica se ajusta gradualmente.
- Campo T:
  - Tiempo durante el cual el cambio tiene lugar (3,7 segundos, 30 segundos, 1 minuto y 5 minutos).

TIPO 4 – EXTENSIÓN DE DIRECCIONAMIENTO SEGURO

Byte Datos								Byte Comando								Descripción
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	0	0	0	0	0	Abrir persiana a nivel indicado por datos. Permitir protección solar.
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	0	0	0	0	1	La unidad sólo se activa si coinciden código de área y código de seguridad.
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	0	0	0	1	0	Permite la ejecución de los comandos básicos: apagar todos, encender, apagar y aumentar / disminuir el brillo.
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	0	0	0	1	1	Ejecuta comandos extendidos de tipo 3.
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	0	0	1	0	0	Enciende la unidad.
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	0	0	1	0	1	Apaga la unidad.

Tabla 9 - Listado de comandos extendidos de tipo 4

Este tipo de mensajes tienen la peculiaridad de que requieren que la unidad se active previamente a enviar cualquier comando. Primero, a la unidad se le asigna una dirección segura (formada por los bits A7 a A0) bien de forma manual, bien de forma automática. La asignación automática ocurre la primera vez que la unidad recibe un comando extendido de tipo 4 en funciones 0, 1 o 2, o la primera vez que recibe un comando extendido de tipo 5, siempre y cuando los campos de códigos de área y de dispositivo coincidan con los suyos. En caso de asignación automática, la unidad no puede responder a comandos básicos o extendidos de tipo 3.

Después, la unidad debe ser activada para ponerse en modo escucha de mensajes. Se envía una primera trama de tipo 4 activando la unidad. Después, se pueden enviar tramas de tipo 4 en funciones 3, 4, 5 ó 6. El código de los comandos básicos o extendidos de tipo 3 se codifica en el campo que antes indicaba el código de dispositivo. En el caso de los mensajes extendidos de tipo 3, el byte de datos funciona igual que en estos.

La unidad se "desactiva" cuando recibe un mensaje de tipo 4 y función 0 ó1, que activa otra unidad después de haber recibido el comando dirigido a ella (se pueden activar varias unidades y después enviar un comando), o cuando el código de área coincide pero no el de seguridad (salvo que se trate de un comando extendido de tipo 3).

TIPO 5 – EXTENSIÓN DE DIRECCIONAMIENTO SEGURO PARA GRUPOS

Byte Datos								Byte comando								Descripción
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	0	0	0	0	Ejecutar Grupo 0 (referencia en el código de unidad)
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	0	0	0	1	Ejecutar Grupo 1
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	0	0	1	0	Ejecutar Grupo 2
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	0	0	1	1	Ejecutar Grupo 3
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	0	1	0	0	Apagar unidades grupo 0
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	0	1	0	1	Apagar unidades grupo 1
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	0	1	1	0	Apagar unidades grupo 2
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	0	1	1	1	Apagar unidades grupo 3
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	1	0	0	0	Aumentar brillo grupo 0
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	1	0	0	1	Aumentar brillo grupo 1
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	1	0	1	0	Aumentar brillo grupo 2
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	1	0	1	1	Aumentar brillo grupo 3
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	1	1	0	0	Atenuar brillo grupo 0
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	1	1	0	1	Atenuar brillo grupo 1
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	1	1	1	0	Atenuar brillo grupo 2
A7	A6	A5	A4	A3	A2	A1	A0	0	1	0	1	1	1	1	1	Atenuar brillo grupo 3

Tabla 10 - Listado de comandos extendidos de tipo 5

CARACTERÍSTICAS DE LA LÍNEA

Las líneas X10 no son fiables, en el sentido en que no es posible para el emisor determinar si una trama llegó a su destino correctamente o no. Los dispositivos que tan sólo pueden transmitir e interpretar comandos básicos no poseen ningún tipo de sistema de acceso al medio que permita saber si ha ocurrido una colisión durante el envío de una trama, y aunque poseen un sistema de detección de errores por redundancia, tampoco poseen ningún sistema de corrección de errores. Sin embargo, esto es así porque se espera que, de corromperse o malinterpretarse un comando, el usuario debería poder corregirlo inmediatamente, considerando siempre que puede ver o sentir el efecto de los comandos que ejecuta y que es el usuario el agente que está detrás de su ejecución. Esta es una suposición muy atrevida, puesto que el usuario no siempre podrá comprobar el efecto de los comandos que envía por la red y es muy posible que existan sistemas automatizados que ejecuten comandos X10 en función del entorno o de un esquema predeterminado por el usuario.

Para resolver este grave problema surgen los comandos extendidos de X10. Dos de ellos se dejan a discreción del cliente en cuanto a su implementación en sistemas avanzados, aunque el comando extendido definido en el estándar cubre todas las obvias necesidades que surgen de un sistema limitado por 13 comandos no fiables. En este caso, sí se utiliza una técnica de acceso al medio que permite detectar colisiones. El emisor debe comprobar que en ninguno de los cruces por cero de la señal AC existe un pulso de 120 KHz al enviar un bit de valor cero. En caso de encontrar este pulso, esto significaría que ha ocurrido una colisión y deberá esperar entre 8 y 10 semiciclos de la señal de AC para reenviar la trama. Sin embargo, esto sigue sin garantizar que la transmisión se vaya a realizar sin errores o que el receptor reciba el comando correctamente, puesto que no se envían acuses de recibo.

Veamos un pequeño análisis de la probabilidad de error. Sea p la probabilidad de error en un bit o ciclo de AC, entonces la probabilidad de error en un comando básico es:

$$P_e = 1 - (1 - p)^{47}$$

La probabilidad de error de bit de los comandos extendidos se espera mucho menor gracias al sistema de control de acceso al medio, sin embargo requieren de un mayor número de ciclos:

$$P_e = 1 - (1 - p')^{62}$$

Ahora calcularemos el ratio de transmisión para comandos básicos y extendidos en Europa y Estados Unidos. En este caso, la posición geográfica del hogar donde se instale X10 es importante, puesto que la frecuencia de la corriente eléctrica es diferente para Europa (50 Hz) y Estados Unidos (60 Hz).

Región	**Comando básico**	**Com. Ext. 1**	**Com. Ext. 2**	**Com. Ext. 3**
Europa	1,064 com / s	0,806 com/s	0,714 com/s	0,714 com/s
EEUU	1,277 com / s	0,968 com / s	0,857 com / s	0,857 com/s

Tabla 11 - Caudal en líneas X10 por regiones

WIRELESS X10

Wireless X10 hace referencia a un conjunto de tecnologías desarrolladas por X10 Ltd. para permitir la interacción de dispositivos inalámbricos con la red cableada. No existe, de momento, ningún método para comunicar redes inalámbricas de este tipo de dispositivos con la red cableada de X10. Es decir, cada uno de estos dispositivos inalámbricos se debe emparejar con otro dispositivo conectado a la red eléctrica que haga de proxy.

X10 Ltd. tiene, a fecha de este proyecto, muy diversos dispositivos inalámbricos que vende bajo su marca X10 Wireless. Operando a una frecuencia de 2,4 GHz pero sin respetar ninguno de los estándares del IEEE, así como mandos a distancia operando a 310 MHz en Estados Unidos y a 433 MHz en la Unión Europea. Además de estos dispositivos de radio frecuencia, también tienen algunas soluciones que hacen uso de comunicaciones ópticas mediante infrarrojos. En todos los casos, es necesario un dispositivo proxy que traduzca los comandos enviados o dirigidos a estos dispositivos desde o hasta la red X10 cableada.

El principal problema que plantea esta solución es, como ya se ha mencionado antes, que la tecnología inalámbrica proporcionada por X10 no permite formar redes inalámbricas que interactúen con las redes cableadas a través de bridges, o puedan tener un funcionamiento autónomo, y por lo tanto se mantienen todas las limitaciones de las que ya hemos hablado en el apartado anterior, en especial el bajo caudal u fiabilidad que ofrecen estas líneas.

Figura 20 - Ejemplo de interruptor inalámbrico X10 Wireless

X10 Ltd. pone a disposición de sus clientes dos paquetes de software, 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.

ActiveHome se distribuye junto con el dispositivo CM11A de X10 Ltd., que ofrece un interfaz de control y monitorización de la red a través de puerto serie a 4800 baudios, sin bit de paridad, 8 bits de datos y uno de parada. Este dispositivo utiliza su propio protocolo de comunicaciones, con lo que los mensajes X10 no se envían directamente a través del puerto serie, sino una versión simplificada de estos (aunque utiliza el mismo código binario para códigos de área, dispositivo y función), mediante un protocolo de parada y espera. El dispositivo CM11A permite el envío tanto de comandos básicos como extendidos, y añade además la funcionalidad de macro de comandos, pudiendo programar el dispositivo para ejecutar una serie de comandos a determinadas horas del día. Para aprovechar toda esta funcionalidad, se hace uso del software ActiveHome, del que se puede ver una captura en la Figura 21. Este software fue desarrollado en 1998, y por lo tanto ha quedado obsoleto en muchas facetas tales como facilidad de uso y compatibilidad, y está disponible sólo para sistemas operativos Microsoft Windows.

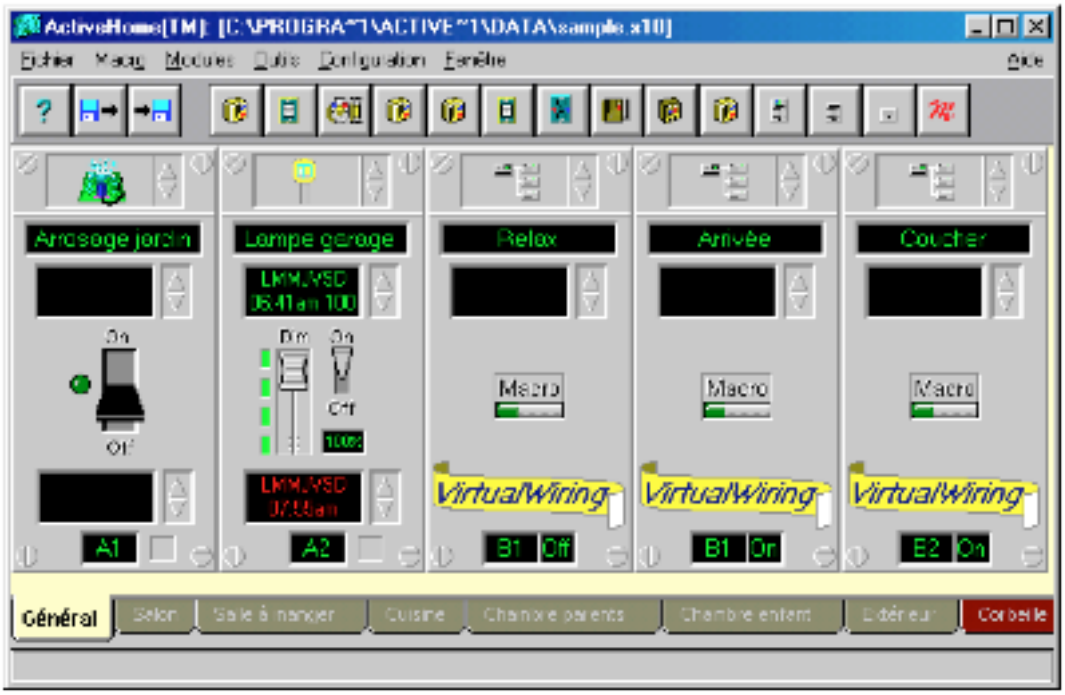

Figura 21 - Captura de pantalla de software Active Home

Por otro lado, X10 Ltd. también proporciona los dispositivos CM15A, que hace uso del puerto USB en vez del puerto serie, y CM17A, que utiliza un enlace de radiofrecuencia para comunicar ordenador y red X10. Ambos se distribuyen junto con ActiveHome Pro, una versión actualizada estéticamente de ActiveHome, pero que no proporciona grandes novedades en cuanto a funcionalidad con respecto a esta.

OBSERVACIONES FINALES

X10 es, como hemos visto, una tecnología práctica y de bajo coste que permite el control remoto y automatizado de los distintos elementos eléctricos dentro del hogar, tales como lumínicas, persianas, sensores, electrodomésticos o aspersores. Sin embargo, al no ser un estándar abierto, todas las soluciones dependen de la misma compañía: X10 Ltd., y por lo tanto su difusión, promoción y evolución se ven severamente limitadas.

Es llamativo que esta tecnología a penas haya evolucionado en los 30 años que lleva en el mercado, y que además haya sobrevivido intacta tantísimo tiempo. Esto se ha debido a que toda la competencia durante ese tiempo se ha centrado en productos de mucho mayor coste, tales como las soluciones proporcionadas por las tecnologías KNX/EIB y LonWorks, orientadas al mercado empresarial y de viviendas de lujo, y que requieren de cableado específico, de coste prohibitivo para hogares que no sean de nueva construcción.

Las limitaciones de esta tecnología no son, a pesar de su popularidad, nada despreciables. El caudal en comandos por segundo es extremadamente bajo, siendo muy perceptible el retardo entre le ejecución de un comando y la acción por parte del dispositivo receptor (un segundo o más), siendo este, a opinión del autor, el principal motivo por el que esta tecnología no ha tenido una mayor difusión en el mercado. Además, su funcionamiento no siempre es óptimo, puesto que al viajar la señal X10 por la red eléctrica, está sujeta a las impedancias a altas frecuencias introducidas por los diversos elementos de la red eléctrica, desde conmutadores hasta electrodomésticos como los ordenadores personales, que pueden interferir de forma severa con la señal, llegando incluso a atenuarla por debajo del umbral de detección o introduciendo ruido de alta frecuencia que haga inviable esta solución. Existen diversos productos para evitar estos efectos adversos en la red, pero el diagnóstico de estos problemas y su solución no están al alcance de usuarios sin los debidos conocimientos técnicos, encareciendo por lo tanto el producto.

Otras de las limitaciones, como por ejemplo la incapacidad del conjunto de tecnologías inalámbricas de formar una red inalámbrica propia o de funcionar de manera autónoma, así como la baja calidad del software de control y monitorización proporcionado por X10 Ltd., tienen una solución más sencilla.

Gracias a los dispositivos CM11A, CM15A y CM17A es posible desarrollar software que se encargue de la monitorización y control de la red y que proporcione además compatibilidad con otras tecnologías, efectivamente actuando de bridge entre tecnologías inalámbricas tales como 802.15.4, Bluetooth o WiFi, en conjunto con un ordenador personal o dispositivo electrónico debidamente adaptado.

Existen diversos desarrollos de software, algunos de ellos de código libre como HEYU, que intentan aumentar la funcionalidad de los dispositivos de control que ya hemos mencionado, por ejemplo permitiendo la carga de macros más complejas en la red, construcción de interfaces de usuario más intuitivas, o compatibilidad con otros sistemas operativos, especialmente aquellos basados en UNIX como las distribuciones de GNU/Linux o MacOS. Sin embargo, no es fácil encontrar una solución que se haya construido con la idea de proporcionar compatibilidad con otras tecnologías además de reinventar el software ActiveHome, de ser así, puede que el bajo coste y sencillez de despliegue de X10 junto con la potencia de soluciones inalámbricas más extendidas acercase el hogar inteligente al consumidor medio.

CAPÍTULO 3 - EL API DE DESARROLLO PROTOCOLO_X10

CONFIGURACIÓN DEL ENTORNO

La configuración completa del entorno de desarrollo se indica en el Anexo 1 - Configuración del entorno de desarrollo. Aunque la mayor parte del API es compatible con diversos sistemas operativos, en especial basados en UNIX, la mayor parte de las pruebas se realizaron bajo Windows XP Service Pack 2, aunque también se comprobó su funcionamiento bajo la distribución de GNU/Linux Ubuntu 7.10.

A partir de este punto consideraremos que el entorno de desarrollo está formado por un ordenador personal con la siguiente configuración y elementos instalados:

- Sistema Operativo Windows XP.
- Java J2SE JDK 1.6.
- Librería externa javax.comm para la comunicación con el dispositivo CM11A mediante puerto serie.
- Librería protocolo X10 desarrollada en este proyecto.
- Puerto RS-232 o USB conectado a un dispositivo CM11A de acceso a la red X10 PLC.
- Red X10 de pruebas, en especial con elementos lumínicos compatibles con los comandos Bright y Dim.

Como ya describimos en la sección de medios materiales, disponemos de elementos básicos que aceptan sólo los comandos de apagado y encendido, así como de lumínicas que reaccionan ante comandos Bright y Dim. Sin embargo, no disponíamos de elementos que aceptasen el uso de comandos extendidos, y su escasa presencia en el mercado así como incompatibilidad con el software de control comercial hicieron que optásemos por no implementar estos servicios.

Si están desarrollados sin embargo otros comandos básicos de los cuales no se pudo comprobar su correcto funcionamiento en la red X10, pero que sí fueron bien interpretados y aceptados por el dispositivo de acceso CM11A.

En el Anexo 3 – Presupuesto se muestra un presupuesto completo para el despliegue de un entorno de pruebas con los mismos elementos que el entrenador desarrollado por la empresa NLaza. Este presupuesto no tendrá en cuenta las horas necesarias para la instalación y despliegue de todo el entorno de desarrollo. Gracias al uso de herramientas open source o de acceso gratuito el coste se centrará en el precio de adquisición a fecha de presentación de este proyecto.

DESCRIPCIÓN GENERAL Y OBJETIVOS

Aunque en un sentido amplio esta herramienta se tratará como un API de desarrollo de aplicaciones para el control y monitorización de redes X10, también puede considerarse un driver para el dispositivo CM11A con el que interactúa. El API se encuentra dentro del paquete ‘protocolo_X10’ y se accede a su funcionalidad de forma externa mediante la importación del package protocolo_X10.

La biblioteca protocolo_X10 está inspirada en el funcionamiento del software comercial ActiveHome, el más utilizado de forma comercial para el control de redes domóticas X10. Sin embargo, el objetivo de este API es permitir el desarrollo de aplicaciones que vayan más allá de las funcionalidad de este programa y que a la vez brinden un entorno de desarrollo intuitivo y simple que oculte la complejidad del protocolo de comunicación con el dispositivo CM11A que describiremos más adelante.

Las principales diferencias con respecto al software comercial son:

- ActiveHome no permite acceso remoto al servidor domótico conectado al CM11A.

- ActiveHome no permite acceso concurrente a la red X10 PLC a través del CM11A, y por lo tanto está limitado a un sólo cliente conectado directamente al dispositivo.

- ActiveHome permite la programación de macros y eventos sólo dentro del espectro permitido por el dispositivo CM11A, y aunque permite definir escenarios de acción formados por varios dispositivos, su implementación es poco flexible.

- ActiveHome no permite el acceso directo al dispositivo CM11A, con lo que no es ampliable ni configurable.

- ActiveHome no permite la localización automática del puerto serie o USB al que esté conectado el gateway de acceso a la red X10.

El objetivo de esta librería es el de posibilitar el fácil desarrollo y despliegue de aplicaciones que puedan gestionar y vigilar una red domótica X10 tanto de forma local como remota. Todos los comandos básicos son aceptados, así como los comandos básicos dirigidos a varios dispositivos para un mejor aprovechamiento de la red. Los comandos extendidos no fueron implementados en este sistema por varios motivos que ya se han expuesto, entre ellos la incapacidad de probar su correcto funcionamiento con el material del que disponíamos así como su limitado uso comercial y penetración en el mercado.

Esta librería permite el control del dispositivo CM11A a través del puerto serie o puerto USB, tanto para el envío de comandos como para la monitorización de la red. Para ello, hacemos uso de la librería externa javax.comm para comunicación mediante puerto serie, del protocolo de comunicación con el dispositivo CM11A (que es diferente del protocolo de comunicaciones X10 PLC), y de mecanismos de gestión de concurrencia que permitan el acceso por parte de varios clientes de forma simultánea, tanto remotos como locales.

El gran objetivo de este proyecto es que el control de la red X10 sea extremadamente sencillo, y pueda implementarse mediante el intercambio de cadenas de caracteres, de forma que sea inmediatamente accesible a cualquier desarrollador. La mayor ventaja de X10 es su simplicidad, permitiendo configurar la dirección de un dispositivo sin más que establecer su código de área y de dispositivo (por ejemplo: A7) y actuar sobre él mediante una serie de comandos básicos. Mantener esta simplicidad en el uso del API ha sido primordial en su desarrollo.

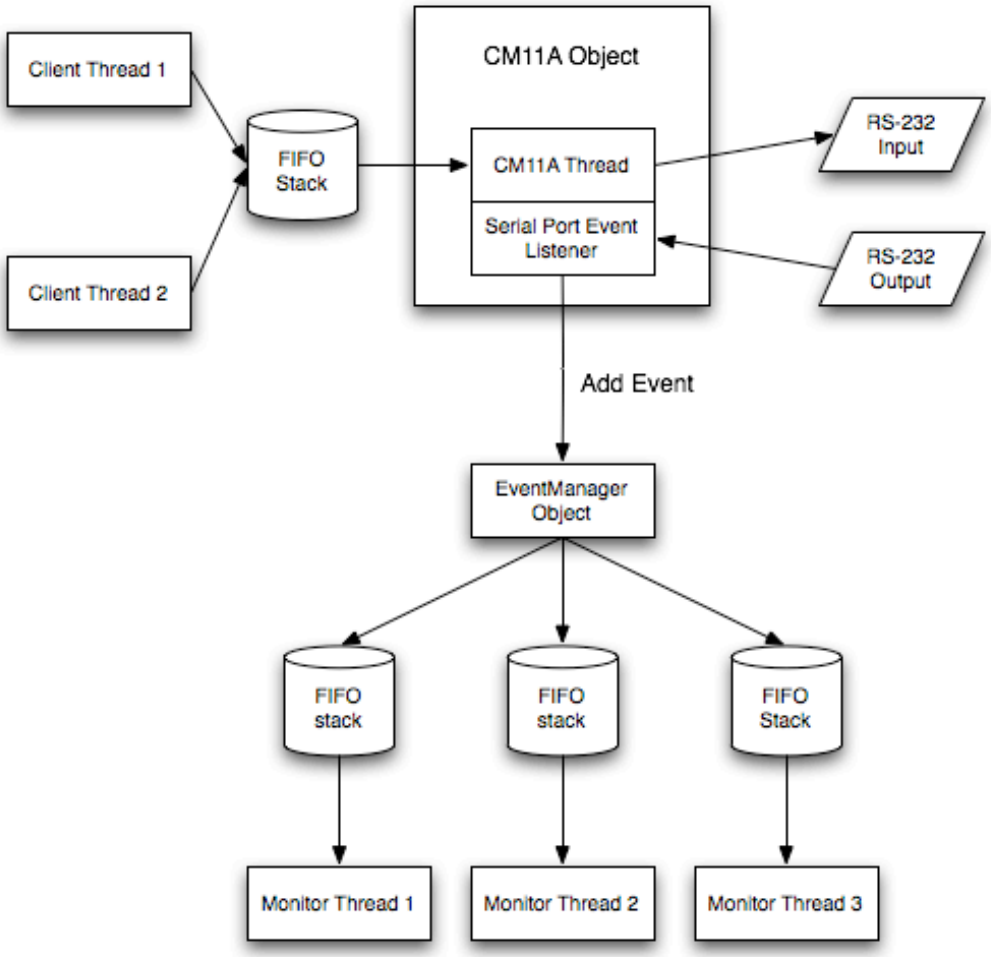

Figura 22 - Descripción general de la librería protocolo_X10

La Figura 22 muestra, de forma esquemática, el esquema implementado en la libraría protocolo_X10 desarrollada en este proyecto. Los hilos cliente y monitor son hilos externos a la librería, y desde donde se espera que se haga la comunicación con los servicios de la librería, aunque como veremos más adelante también se han desarrollado dos servidores de sockets que permiten un acceso más sencillo a la red X10.

En este diagrama, las flechas indican la dirección en la que viajan los datos, que pueden ser objetos Java de la clase protocolo_X10.Command propios de la librería, o variables byte y short para la comunicación con el puerto serie. No entraremos aquí en el detalle de las comunicaciones, que dejaremos para secciones posteriores.

Por último, las pilas FIFO son pilas de objetos protocolo_X10.Command síncronas, preparadas para su acceso concurrente por parte de varios hilos e implementadas mediante herramientas proporcionadas por la librería java.concurrent incluida en la distribución del SDK en la que se basa este proyecto, y por lo tanto ya conocidas por los desarrolladores Java que puedan aprovechar este desarrollo.

En los siguientes apartados describiremos el protocolo de comunicaciones con el dispositivo CM11A para el envío y recepción de datos con la red X10, así como las librerías externas en las que hemos apoyado el desarrollo y cómo estas han permitido el desarrollo de un código compacto y sencillo. Después continuaremos con una descripción más detallada de cada uno de los módulos que componen el API de desarrollo y cómo se realiza la comunicación entre estos módulos y con procesos externos. Así mismo haremos hincapié en las ventajas y posibilidades de desarrollo que ofrece esta librería frente a otras opciones, en concreto con respecto a ActiveHome como software de control ofrecido de forma comercial.

EL PROTOCOLO CM11A

DESCRIPCIÓN GENERAL

La comunicación con el dispositivo CM11A de Marmitek se realiza mediante un protocolo de parada y espera binario que permite no sólo la comunicación con la red X10 PLC, si no también la configuración del propio dispositivo para la asignación de macros y tareas programadas. El objetivo de este proyecto es el estudio de la comunicación con la propia red, y puesto que la programación de este tipo de tareas es más sencilla y fiable a nivel del propio servidor domótico, nos centraremos sólo en la comunicación de comandos básicos y en la monitorización de la red, dejando la programación de tareas y macros para aplicaciones que hagan uso de este API.

Como ya hemos visto, las dispositivos de las redes X10 tienen direcciones formadas por una letra y un número, que corresponden al código de área y de dispositivo respectivamente. Sólo puede haber 16 códigos de área y 16 dispositivos por área compartiendo una misma red eléctrica. La codificación binaria de estas direcciones se muestra en la Tabla 13.

Los parámetros de configuración del dispositivo CM11A para la comunicación a través de puerto serie se muestran en la Tabla 12. El API de desarrollo ya está configurado por defecto con estos valores, de forma que el desarrollador que haga uso de él no deba preocuparse de la configuración a bajo nivel.

Parámetro	Configuración
Baud Rate	4800 bps
Paridad	No
Bits de datos	8
Bits de parada	1

Tabla 12 - Configuración del enlace RS-232 para la comunicación con el dispositivo CM11A

Código de área	Código de dispositivo	Valor Binario
A	1	0110
B	2	1110
C	3	0010
D	4	1010
E	5	0001
F	6	1001
G	7	0101
H	8	1101
I	9	0111
J	10	1111
K	11	0011
L	12	1011
M	13	0000
N	14	1000
O	15	0100
P	16	1100

Tabla 13 - Valor binario para los códigos de área y dispositivo para su comunicación a través del puerto serie

Como ya hemos dicho, no todos los códigos de función han sido implementados, y sólo haremos uso de los comandos básicos. La Tabla 14 contiene todos los códigos de función aceptados por el API. Como se puede ver, también utilizan 4 bits para su codificación.

Código de función	Descripción	Valor Binario
ALL_UNITS_OFF	Apaga todos los dispositivos dentro de un área.	0000
ALL LIGHTS ON	Enciende todas las luces de un área.	0001
ON	Enciende el o los dispositivos a los que se dirige.	0010
OFF	Apaga el o los dispositivos a los que se dirige.	0011
DIM	Reduce la intensidad de brillo en el porcentaje indicado o porcentaje por defecto.	0100
BRIGHT	Aumenta la intensidad de brillo en el porcentaje indicado o porcentaje por defecto.	0101
ALL LIGHTS OFF	Apaga todas las luces de un área.	0110
HAIL REQUEST	Pide la dirección de todos los dispositivos conectados a la red. No todos los dispositivos son compatibles con este comando.	1000
HAIL ACK	Asiente la petición previa.	1001
PRE SET DIM	Disminuye el brillo hasta su valor por defecto.	1010
PRE SET BRIGHT	Aumenta el brillo hasta su valor por defecto.	1011
STATUS ON	El dispositivo envía informes periódicos sobre su estado. No todos son compatibles con este comando.	1011
STATUS OFF	El dispositivo deja de enviar informes periódicos sobre su estado. No todos son compatibles con este comando.	1110
STATUS REQUEST	Petición de estado del dispositivo. No todos los dispositivos son compatibles con este comando.	1111

Tabla 14 - Descripción y valor binario de los comandos X10 permitidos en el API

ENVÍO DE COMANDOS

El envío de comandos al dispositivo CM11A diferencia entre dos supuestos: el envío del listado de dispositivos al que está dirigido un comando y el envío del código del comando a ejecutar, junto con el número de veces a ejecutarlo en caso de que sea necesario (casos DIM y BRIGHT).

Para diferenciar entre ambos casos se utiliza un byte de cabecera que se envía siempre antes del byte que indica el dispositivo al que se dirige el comando o el comando en sí. Esta cabecera tiene el formato que se muestra en la Tabla 15.

Cabecera							
B7	B6	B5	B4	B3	B2	B1	B0
Nivel					1	F/A	E/S

Tabla 15 - Byte de cabecera para el envío de datos al dispositivo CM11A

Los bits de 7 a 3 (los 5 de mayor precedencia) indican el nivel por el que debe aumentarse o disminuirse el brillo (comandos BRIGHT y DIM) y permanecerán a cero en el resto de casos. A pesar de que con 5 bits se pueden representar hasta 32 niveles diferentes, el protocolo del CM11A permite sólo 22 niveles de ajuste. Esta excepción inesperada es otro de los motivos por los cuales es necesario hacer que la capa de comunicaciones con el dispositivo de enlace CM11A sea lo más transparente posible para los programadores de aplicaciones.

El bit B2 siempre tiene valor 1 para permitir la correcta sincronización con el CM11A.

El bit B1 indica si el siguiente byte contendrá datos de direccionamiento (código de área y de dispositivo) tomando el valor 0, o de función (código de área y de función) tomando el valor 1.

El bit B0 indica si el comando es Extendido (valor 1) o Básico (valor 0). En lo que concierne a este proyecto, este bit tomará siempre el valor 0.

En el caso de envío de códigos de dispositivo, esta cabecera toma siempre el valor 0x04.

El envío de un comando básico se realiza en varias etapas. El PC primero envía el código de dispositivo al que se dirige el comando. Se puede repetir este envío de código de área + código de dispositivo tantas veces como dispositivos a los que se dirija el comando. Este código va precedido de la cabecera correspondiente, y tiene el formato que se muestra en la Tabla 16.

Byte de direccionamiento							
B7	B6	B5	B4	B3	B2	B1	B0
Código de Área				Código de dispositivo			

Tabla 16 - Byte de direccionamiento para el envío de comandos al dispositivo CM11A

La respuesta del dispositivo CM11A se tratará de un Checksum para comprobar la correcta recepción del comando. Este checksum se calcula como la suma del byte de cabecera con el formado por el código de área y dispositivo.

Por ejemplo, para enviar un comando dirigido al dispositivo A2 enviaríamos al interfaz la trama 0x046E indicando que se trata de un código de direccionamiento (0x04) seguido del código de área (6) y del código de dispositivo (E). La respuesta del dispositivo CM11A será el checksum de este comando, es decir (0x04 + 0x6e) & 0xFF, es decir 0x72.

Si el checksum es correcto, el servidor domótico que implementa el API enviará el código 0x00 indicando que la transmisión ha sido correcta, que será respondido a su vez por el CM11A con el código 0x55 que indica que está listo para recibir más datos.

En caso de que el checksum sea incorrecto, el servidor responderá reenviando el mensaje (en el ejemplo anterior, 0x046e) hasta que el checksum devuelto por el CM11A sea correcto.

Si el comando va dirigido a varios dispositivo, basta con repetir este proceso para cada uno de ellos, sin embargo es importante recordar que sólo se podrán listar

dispositivos que pertenezcan a una misma área, y que no pueden enviarse comandos de forma simultánea a dispositivos que pertenezcan a áreas diferentes (por ejemplo A2 y B15).

Después de haber enviado el o los códigos de dispositivo, el servidor domótico debe enviar el código del comando que desea enviar. Este caso es algo más complejo que el anterior, puesto que es necesario indicar no sólo el código de área y de comando, si no en el caso de los comandos DIM y BRIGHT el porcentaje por el que se desea reducir o aumentar el brillo de una lumínica o dispositivo susceptible a este comando (como una persiana). Este mensaje, como el anterior, estará formado por una cabecera y un código, con el formato de cabecera que se ha mencionado antes y el byte siguiente tomando el formato que se muestra en la Tabla 17.

Byte de función							
B7	B6	B5	B4	B3	B2	B1	B0
Código de Área				Código de función			

Tabla 17 - Byte de función para el envío de comandos al CM11A

La comunicación es análoga al caso de envío de tramas de direccionamiento, es decir que primero se envía la cabecera y el byte de función, se espera el checksum y se procede en función de su valor exactamente igual que en el caso anterior.

La siguiente figura muestra el intercambio de mensajes entre el servidor domótico y el dispositivo CM11A a través del puerto serie para el envío de un comando BRIGHT con valor de 72%. Cabe notar que el valor de 72% de brillo no corresponde a un nivel real, si no que será redondeado al nivel 16 de 22 que corresponde a un 72,73% de brillo. De nuevo, esta clase de aproximación contra intuitiva será resuelta por el API de desarrollo, que permitirá ajustar los niveles por porcentaje sin preocuparse del cálculo necesario en las capas de comunicación inferiores.

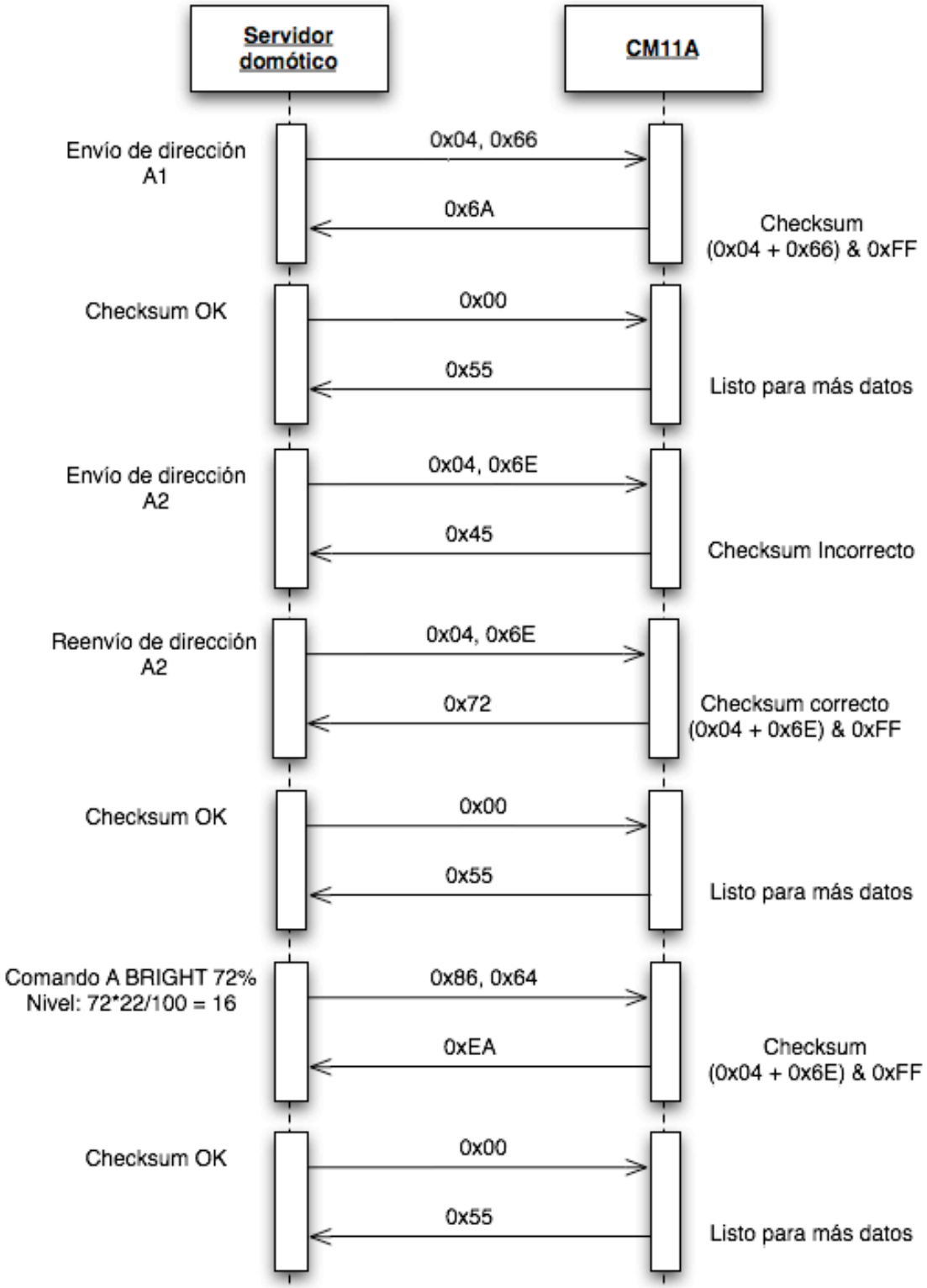

Figura 23 - Ejemplo de envío de un comando X10 a través del puerto serie

MONITORIZACIÓN

Cuando se produzca un evento en la red X10 PLC, como por ejemplo la activación de un interruptor, el dispositivo CM11A notificará al servidor domótico mediante el envío de una trama especial que indica que debe enviar información a éste. Esta trama se llama DATA POLL y toma el valor 0x5A.

Cuando el servidor esté listo para la recepción de estos datos, enviará una trama de asentimiento (también en formato byte) con valor 0xC3.

La respuesta del dispositivo CM11A tiene longitud variable dependiendo del evento, y no transmite éste de igual manera que el servidor lo hace hacia el CM11A. Primero se envía un byte con la longitud total en bytes del evento, sin tener en cuenta este primer envío. La longitud total máxima permitida es de 8 bytes. Sigue a este envío un segundo envío que indica que bytes de los que siguen corresponden a direcciones (tomarán valor 0) y cuales a funciones. Por ejemplo, si este byte tomase el valor 0x04 (tercer byte a 1, resto a cero) indicaría que los dos primeros bytes son direcciones y el tercero corresponde a la función. En el caso de funciones compuestas (BRIGHT y DIM), el byte de función son en realidad 2, uno indicando el código de área y de función y otro indicando el nivel al que se ha configurado el brillo. El byte que indica el nivel no cuenta a efectos del segundo byte (aquel que indica el tipo de cada envío) pero sí de cara al del número total de bytes.

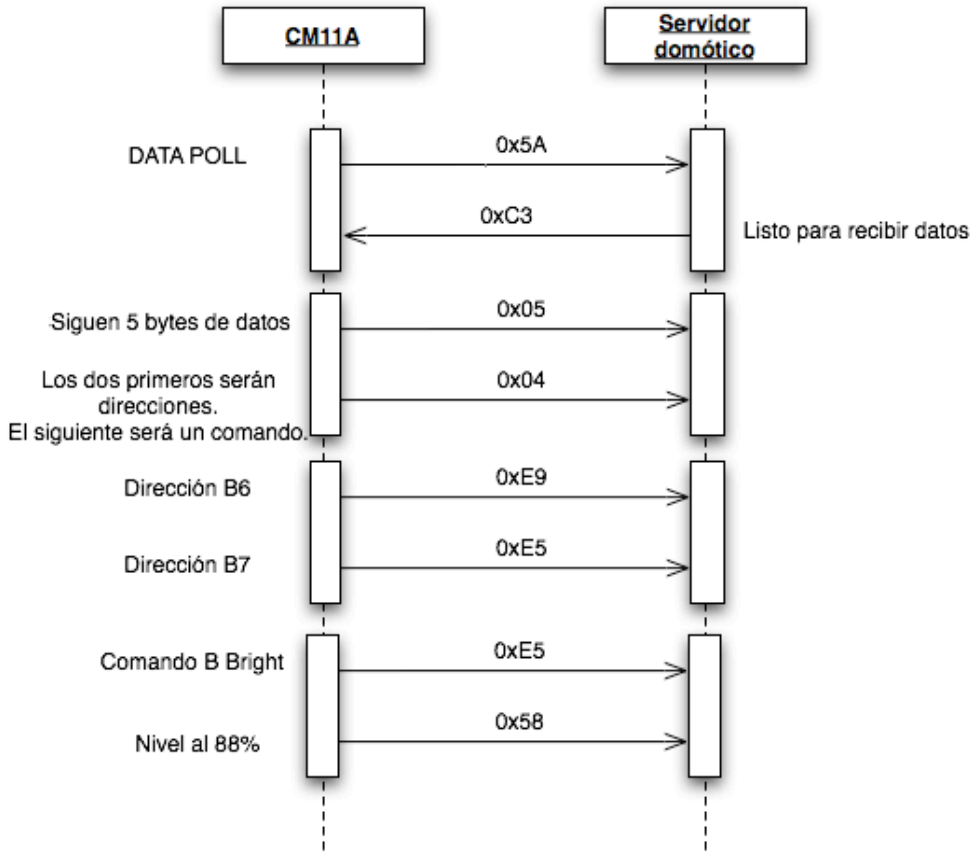

Figura 24 - Ejemplo de recepción de un evento desde el dispositivo CM11A

Cabe destacar que en este caso el nivel ya no se indica mediante 5 bits, sino mediante 1 byte completo que indica el nivel de brillo al que está el dispositivo de entre los 210 posibles, siendo el menor el nivel 1 y el mayor el nivel 210. Esto es algo particular, puesto que los comandos de Bright y Dim sólo permiten ajustar 22 niveles diferentes, aunque el dispositivo acepta más. La razón detrás de esto es que la lumínica o dispositivo susceptible a aceptar varios niveles de funcionamiento (por ejemplo persianas) puede disponer de un ajuste manual que no corresponda con uno de los 22 niveles a los que permite ajustar el nivel el protocolo de envío del CM11A. De nuevo, este comportamiento contra intuitivo es resuelto por el API.

CONFIGURACIÓN DE FECHA Y HORA

En la primera comunicación con el CM11A, éste le pedirá la configuración de fecha y hora al servidor domótico mediante el envío el byte TIME REQ (0xA5). Esto sirve para la configuración de macros y eventos programados en el propio dispositivo, pero que como ya hemos mencionado se alejan de los objetivos de este proyecto.

La respuesta del servidor a esta petición está formada por dos bytes, uno de cabecera llamado TIME HEADER (0x9B) seguido de 6 bytes que indican la hora (con resolución de segundos) y el día del año. Puesto que esta configuración es irrelevante para nuestros propósitos, el dispositivo siempre se configura con estos valores a cero.

DESCRIPCIÓN DE LOS MÓDULOS EN DETALLE

LIBRERÍAS EXTERNAS

JAVAX.COMM

La instalación de esta librería externa puede verse en el Anexo 1 - Configuración del entorno de desarrollo. Esta librería permite:

- Enumeración de puertos tanto serie como paralelos.
- Configuración del puerto (caudal, bits de parada y paridad, etcétera).
- Transferencia de datos a través del puerto RS-232.
- Control de flujo de software.
- Control del límite de buffer de recepción.
- Notificación de eventos asíncrona.

El javadoc de esta librería acompaña a la distribución digital de este proyecto, pero puede descargarse desde la fuente mencionada en las referencias. En concreto, en este proyecto nos apoyaremos especialmente en la interfaz SerialPortEventListener, implementada por la clase principal de nuestra librería llamada CM11A. Este interfaz permite reaccionar mediante eventos asíncronos a la presencia de datos en el canal de entrada. Gracias a esta función, podremos tratar los datos de entrada evitando espera activa en el puerto de escucha.

Otra clase en las que nos apoyaremos será CommPortIdentifier, que nos permite tanto obtener un puerto serie por referencia (por ejemplo COM1, COM2, etcétera) como el listado de todos los puertos serie de la máquina. Esta última característica es especialmente útil puesto que permite localizar de forma automática el puerto al que está conectado el dispositivo CM11A, sin necesidad de intervención del usuario y ofreciendo una característica que no está presenta en la versión comercial de software de control.

Por último, haremos uso de la clase SerialPort para la configuración y conexión con el puerto serie al que esté conectado el dispositivo CM11A. Esta clase permite abrir conexiones DataInputStream y DataOutputStream de acceso al puerto serie. Como ejemplo de uso, a continuación se incluye un ejemplo de conexión y configuración del puerto serie para su uso con el dispositivo CM11A.

Un ejemplo de configuración del puerto serie mediante la librería javax.comm puede verse en el Anexo 2 - Manual de uso. En este ejemplo se utiliza la configuración necesaria para la comunicación con el dispositivo CM11A.

JAVA.IO

De la conocida librería de java java.io para el envío y recepción de datos con periféricos haremos uso de DataInputStream y DataOutputStream para la comunicación con el puerto serie.

Esta librería viene incluida en la distribución estándar del SDK de Java, la documentación de su API así como del resto de librerías Java puede encontrarse enlazada en las referencias.

JAVA.UTIL.CONCURRENT.ARRAYBLOCKINGQUEUE

Durante el desarrollo de este proyecto se hizo uso de varias técnicas de control de concurrencia, entre ellas la sincronización de acceso a métodos, paso de mensajes, semáforos y hasta cerrojos. Sin embargo, la librería java.util.concurrent proporciona un mecanismo de control de acceso perfecto para nuestras necesidades en la clase ArrayBlockingQueue.

Los objetos ArrayBlockingQueue son pilas FIFO con escritura y lectura bloqueantes, y que por lo tanto permiten el acceso de varios hilos de forma simultánea. Estos objetos han sido utilizados para la implementación de la pila de entrada de comandos así como las pilas de monitorización de las que hablaremos más adelante.

JAVA.NET

Nos apoyaremos en esta librería para el acceso remoto al servidor domótico mediante las clases Socket y ServerSocket, que permiten la transferencia de objetos serializables mediante el protocolo de transporte TCP/IP.

En concreto haremos uso de estas clases para la implementación de un servidor básico que permita la comunicación con clientes remotos mediante el paso de Strings, aunque como veremos más adelante se podrían haber tomado otras opciones más complejas para esta comunicación.

MÓDULOS DEL API

En esta sección discutiremos los módulos que componen el núcleo del API de desarrollo, así como algunas clases que se han incluido como apoyo y ejemplo de implementaciones complejas del sistema.

Como ya hemos discutido en los apartados anteriores y al comienzo de este capítulo, el API debe cumplir ciertos requisitos:

- Encapsulado de comandos que permita la definición de Comandos sin necesidad de conocer el protocolo de comunicaciones con el dispositivo CM11A. Además, estos comandos deben poderse construir a partir de cadenas de caracteres que describan el comando, de forma que su definición y manipulación resulte lo más sencilla e intuitiva posible.

- Encapsulado de dispositivos, que permita la definición de dispositivos sin necesidad de conocer el protocolo de comunicaciones con el dispositivo CM11A. De nuevo, esta definición debe poderse hacer mediante cadenas de caracteres a fin de hacer transparente la comunicación con la red.

- Sistema de comunicación con el dispositivo CM11A que permita el envío de comandos a la red X10 PLC, así como la monitorización de eventos en ésta. Además, este sistema de comunicación debe ser implementado como un servicio fácilmente accesible por otros procesos de control y monitorización, tanto remotos como locales, y la comunicación con este servicio debe poderse realizar mediante el intercambio de cadenas de caracteres.

- Configuración tanto manual como automática del puerto serie conectado al dispositivo CM11A.

- Mecanismos de acceso concurrente que permitan que varios agentes actúen y monitoricen la red sin interferir entre ellos o con agentes físicos que actúen directamente sobre la red (por ejemplo, activando un interruptor).

LA CLASE UNIT

La clase UNIT permite encapsular la información necesaria sobre un dispositivo de la red X10. En concreto, esta clase dispone de constructores para crear un objeto Unit que represente directamente un dispositivo de la red a partir de su representación en texto plano (por ejemplo "A1") o mediante el byte que representa su código de área y dispositivo según la notación del protocolo CM11A descrito al principio de este capítulo.

Esta clase permite además que otras clases accedan de forma directa a la representación de la dirección de este dispositivo en varios formatos:

- En texto plano: "A1"
- En representación binaria según el protocolo del CM11A.
- Únicamente la representación binaria de su código de área.
- La representación binaria que permite su transmisión a través del puerto serie, incluyendo la cabecera bien formada.

Además, la clase Unit dispone de los mecanismos necesarios para la comparación de unidades entre sí (para comprobar si dos unidades comparten dirección) e implementa el interfaz java.io.Serializable para su transmisión vía Sockets.

LA CLASE COMMAND

La clase Command permite encapsular toda la información necesaria sobre un comando X10. En concreto, esta clase dispone de los siguientes constructores:

- Constructor vacío.
    - Usado cuando el contenido del comando no es conocido aún o está sujeto a cambios.
- A partir de un array de objetos Unit, el byte de función y el nivel de brillo.
    - El array de objetos Unit representa todas las unidades a las que se dirigirá el comando.
    - El byte de función tal y como viene definido en el protocolo de comunicación del CM11A.
    - El nivel de brillo es un parámetro opcional, y sólo es necesario cuando el comando lo requiere (DIM y BRIGHT). Este número debe encontrarse entre 0 y 100, aunque el dispositivo CM11A sólo diferencie entre 22 niveles distintos, de cara a una representación más transparente al protocolo de comunicación con el CM11A.
- A partir de una representación en texto plano.
    - La representación en texto plano puede incluir un número indefinido de dispositivos a los que va dirigido el comando, y acepta los comandos que requieren que se indique un nivel de actuación (BRIGHT y DIM).
    - Ejemplo: “A1 A2 A7 BRIGHT 57”.

La clase Command permite también otras operaciones esenciales para el buen funcionamiento de la plataforma:

- Añadir, eliminar unidades del listado de unidades al que va dirigido el comando.

- Definir el listado de unidades al que va dirigido el comando.

- Definir y modificar el comando a ejecutar.

- Comprobar si el comando se ha definido por completo o falta el listado de unidades o la descripción del comando a ejecutar.

- Devolver el byte de función del comando tal y como se representa en el protocolo de comunicación con el CM11A.

- Devolver la trama a enviar al dispositivo CM11A para la ejecución del comando, con la cabecera bien formada.

- Devolver la representación como String del comando, incluyendo el listado de unidades a las que se dirige.

- Implementación del interfaz java.io.Serializable.

LA CLASE EVENTMANAGER

Esta clase permite la monitorización de la red por parte de varios agentes, tanto locales como remotos. Para ello hace uso de objetos java.util.concurrent.ArrayBlockingQueue definidos antes. Su funcionamiento es extremadamente sencillo, y permite un seguimiento preciso de los eventos generados en la red sin necesidad de utilizar mecanismos de concurrencia más complejos como sistemas de paso de mensajes o métodos sincronizados.

Por cada servidor existe un único objeto EventManager que gestiona la distribución de eventos en la red a los clientes que están monitorizándola. A este objeto pueden suscribirse clientes sin más que llamando a un método que les devolverá una pila ArrayBlockingQueue en la que el objeto EventManager volcará los eventos que le pase el sistema de control del dispositivo CM11A, y que podrán acceder sin miedo a interferir con otros agentes.

Desconectarse del servicio de monitorización es igualmente sencillo, y tampoco interfiere con el correcto funcionamiento de los demás agentes.

La principal razón por la que se optó por implementar una pila FIFO por cliente en vez de una pila común con distintas referencias a ésta es porque la gestión de las distintas referencias estaba sujeta a problemas de concurrencia, y además al ser la lectura bloqueante en caso de tener muchos agentes el rendimiento general podía ser inferior.

Como contrapartida, agentes poco activos que después de suscribirse no lean eventos de su pila pueden provocar un bajo rendimiento al acumularse en la memoria del servidor objetos Command en sus respectivas pilas. Para evitar que esto se convierta en un problema grave, las pilas tienen un tamaño limitado a 100 eventos por pila.

Además, gracias a la implementación mediante un objeto Vector, el código de implementación es muy intuitivo y fácil de mantener, uno de los grandes objetivos de este proyecto al pretender de servir de base a futuros proyectos de control de redes domóticas.

LA CLASE CM11A

Esta es la clase principal de la librería X10 desarrollada en este proyecto. Su funcionamiento se basa en el interfaz SerialPortEventListener, que permite escuchar eventos asíncronos en el enlace con el dispositivo CM11A mediante puerto serie.

La clase CM11A dispone de dos constructores, ambos requieren del paso por parámetros de dos objetos: un objeto EventManager para la comunicación con los hilos cliente de monitorización y un objeto ArrayBlockingQueue para la recepción de comandos desde los hilos cliente de control. La diferencia entre ambos constructores radica en el parámetro opcional que indica el puerto serie al que está conectado el dispositivo CM11A. Éste parámetro se pasa como String (por ejemplo "COM1") para que las aplicaciones que se ejecuten sobre este API no requieran de conocimiento sobre el funcionamiento interno de la librería javax.comm. En caso de que éste parámetro no esté presente, si intentará localizar el puerto serie de forma automática de la siguiente forma:

- Se obtendrá una lista de todos los puertos disponibles en el servidor domótico. Esta lista incluye tantos los puertos RS-232 como USB, con los que es compatible tanto el dispositivo CM11A como la librería protocolo_X10.

- Se intenta establecer una conexión con cada uno de ellos, fallando ésta si el puerto está ya en uso o no hay dispositivo conectado a él.

- Si la conexión tiene éxito, se configura el enlace según los parámetros del CM11A y se realiza el envío de la trama 0x0466 que corresponde al envío de la dirección A1 como parte del envío de un comando.

- Si se recibe una respuesta en los siguientes 2 segundos con el checksum correcto de este envío, la trama DATA POLL o la trama TIME REQ, se considera que se ha establecido el enlace con éxito.

- Este envío de prueba no afecta a las siguientes comunicaciones con el dispositivo, puesto que no se enviará el comando Checksum OK.

- Esta funcionalidad debe utilizarse con prudencia, puesto que podría afectar al correcto funcionamiento de otros dispositivos conectados al algún puerto serie RS-232 o USB.

Una vez realizado el correcto enlace con el dispositivo CM11A, el hilo de ejecución CM11A permanecerá a la escucha de eventos en el canal mediante la implementación del método serialEvent (SerialPortEvent event). Además, en el método run del hilo nos mantendremos a la espera de entradas de comandos a través del objeto ArrayBlockingQueue. Aunque la lectura sobre esta pila FIFO es bloqueante, esto no afecta a la escucha asíncrona sobre el canal de comunicaciones, y por lo tanto se garantiza un buen funcionamiento del sistema.

Al generarse un evento en el enlace por puerto serie, se sigue el diagrama de flujo siguiente para averiguar qué acción tomar sobre los datos recibidos.

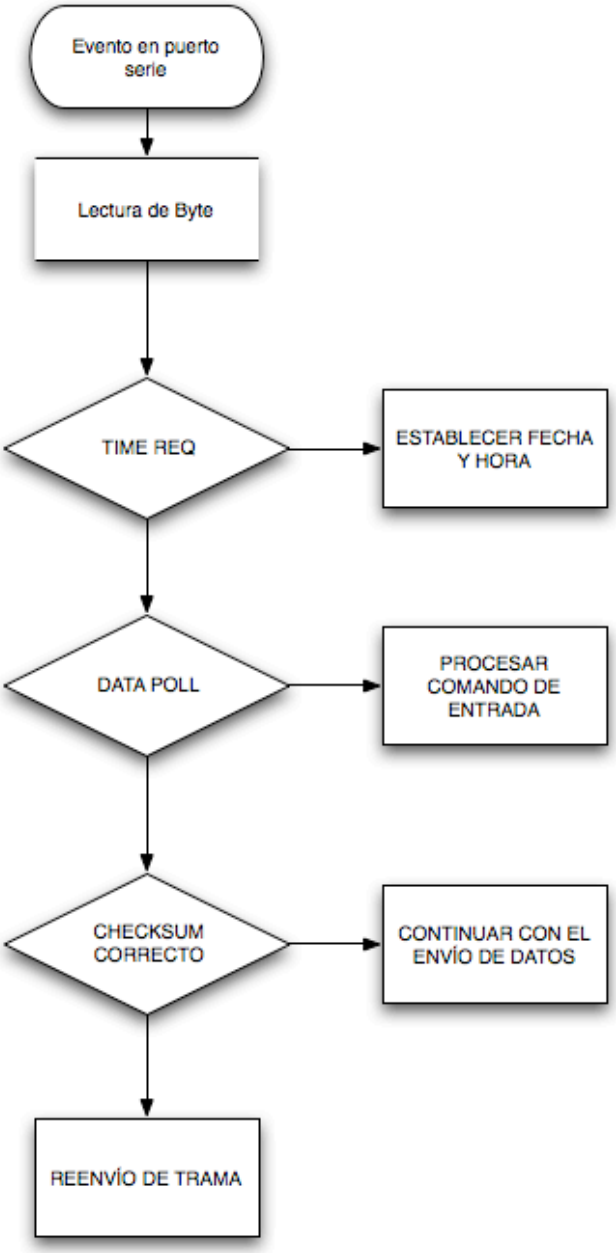

Figura 25 - Diagrama de flujo para la lectura de datos del puerto serie

En caso de recibir el byte TIME REQ, se llevan a cabo las acciones necesarias para establecer la fecha según el convenio que hemos descrito anteriormente, es decir que se envía la fecha correspondiente al 1 de Enero a las 00.00 mediante el envío de 6 bytes de valor 0.

En el caso de un poll de datos, se procesa el comando de entrada siguiendo el esquema que ya se ha descrito en este capítulo en la descripción del protocolo CM11A. Debido a un funcionamiento inesperado del dispositivo CM11A, la recepción de estos datos no es tan intuitiva como era de esperar, y aunque todo poll de datos se referirá únicamente a un comando y nunca a varios, este comando puede no haber sido procesado por el CM11A por completo al pedir el servidor domótico esta información.

En el caso de que el CM11A haya procesado todos los datos del comando antes de que el servidor domótico indique está listo para recibirlos, el intercambio se realizará con normalidad y siguiendo el esquema de la Figura 25.

En caso de que el dispositivo no haya podido procesar el comando completo antes de que el servidor señale que está listo para recibir los datos, el data poll será incompleto y tendrá uno de dos formatos.

- El mensaje contendrá información sólo sobre los dispositivos a los que va el dirigido el comando. Esta información puede ser completa (listando todos los dispositivos) o parcial (listando sólo algunos).

- El mensaje tendrá el mismo formato que el anterior, sólo que en vez de contener información sobre los dispositivos a los que se dirige el comando, contendrá la información sobre el comando ejecutado (función y nivel de acción si procede).

Una vez que el servidor domótico haya procesado el mensaje parcial, si se trata sólo de las direcciones de los dispositivos deberá volver a enviar la trama PC READY (0xC3) indicando que está listo para el resto del mensaje. De esta forma, la implementación del método de procesado de datos es recurrente, y finaliza sólo cuando el mensaje se ha recibido por completo.

Una vez finalizado el procesado de datos de entrada, se envía el comando completo al gestor de eventos a través de una llamada al objeto EventManager. Como ya se ha explicado, este objeto se encargará de añadir el comando a todas las pilas bloqueantes correspondientes a los clientes en escucha.

El último caso corresponde a la recepción de un checksum correspondiente al envío de un comando siguiendo el esquema de la Figura 25. En este caso se comprueba si el checksum ha sido correcto. De serlo, se envía a trama OK (0x00) para señalar que la comprobación ha sido correcta y se lee el siguiente byte que debería ser la trama READY indicando que el CM11A está listo para el resto del comando o el siguiente comando. Si se recibe la trama READY, se da orden de enviar la siguiente trama de datos (si la hay).

En este caso, de nuevo, el comportamiento del CM11A es errático y en vez de recibir esta trama podríamos recibir un DATA POLL. En caso de recibir la trama DATA POLL tendremos que procesar los datos de entrada antes de poder continuar con el envío de datos.

Si el checksum es incorrecto, modificaremos el índice del array de datos a enviar para que se reenvíe el último mensaje. Hablaremos más sobre este índice a continuación.

Como ya hemos dicho, para la recepción de comandos desde los hilos de control se hace uso de una pila FIFO síncrona implementada como un objeto ArrayBlockingQueue. Al aparecer un nuevo objeto Command en esta pila, su procesado se realizará de la siguiente manera:

- Se añade el objeto Command a una pila FIFO de comandos a enviar al dispositivo CM11A de forma inmediata implementada a través de un objeto java.util.Vector.

- Se comprueba si se están enviando datos, si no es así se traduce el comando a enviar a un array de elementos short int (elementos de 2 bytes) que contiene la información a enviar al dispositivo CM11A de forma secuencial, es decir primero la cabecera y código de los dispositivos y al final la cabecera y código de la función.

- Se ejecuta el envío de la primera trama de datos.

- Al terminar el envío de datos, se comprueba si hay más objetos en el Vector de comandos y se envían éstos también.

- Cuando cualquier de estos short se envían de forma errónea, el índice del array de short int con los datos a enviar se modifica para repetir el envío de la última trama.

Es en este punto donde radica la verdadera complejidad de la comunicación con el dispositivo CM11A, puesto que el envío de datos puede verse interrumpido por uno o varios DATA POLL por eventos surgidos en la red. Es por este motivo que se utiliza una segunda pila FIFO como buffer interno en el envío de datos, y un índice en el envío de los datos de un determinado comando para poder continuar con su envío en caso de interrupción del servicio y para poder hacer un reenvío en caso de fallo en el servicio.

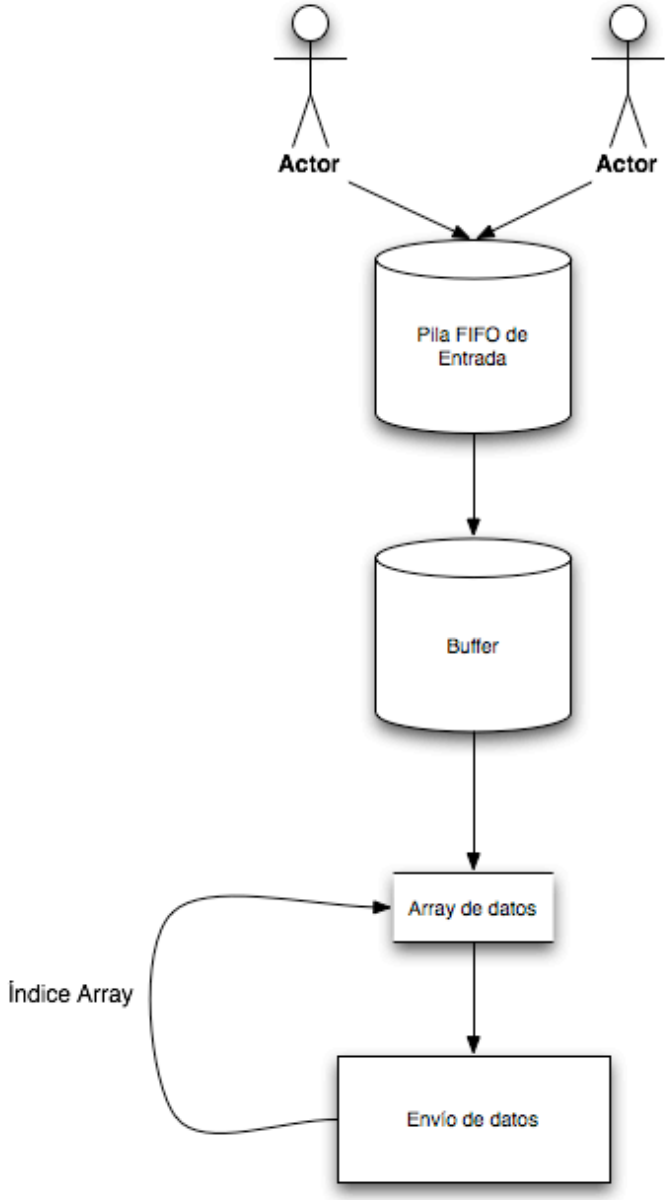

Figura 26 - Esquema de envío de datos

La anterior figura resume todo el esquema de envío de datos desde cualquiera de los hilos de control:

1. Un actor o hilo de control introduce un objeto Command en la pila FIFO de entrada implementada mediante un objeto ArrayBlockingQueue.

2. El hilo de control principal CM11A lee el comando de esta pila e introduce el objeto Command obtenido en el buffer FIFO de comandos a enviar.

3. De esta pila se obtiene el siguiente comando a enviar, que se traduce en un array de datos que contiene las cabeceras correspondientes a las tramas de dispositivo y de función en orden.

4. Este array se controla mediante un índice, que permite reenviar datos en caso de error. Al enviar una trama se establece el checksum esperado que será comprobado por el método de procesado de datos de entrada, que será el que controle el índice en caso de fallo.

5. Se envían datos hasta que no hay más comandos en el buffer.

MÓDULOS DE COMUNICACIÓN

Aunque la comunicación de comandos de forma local ya es muy sencilla a través del manejo de dos pilas FIFO bloqueantes, una de entrada y otra de salida, uno de los grandes objetivos de este proyecto era permitir la comunicación con el servicio de control de la red X10 sin más que mediante la transmisión y recepción de cadenas de texto. Además, aunque tanto los objetos Command, Unit y las propias pilas son serializables y por lo tanto fáciles de comunicar de forma remota, su manejo requiere de algo de estudio sobre la estructura y funcionamiento internos de la librería, y nuestra intención es que incluso el acceso remoto sea extremadamente sencillo de implementar.

Para ejemplificar esta comunicación se desarrolló un servidor de sockets que permite la monitorización y control de la red de una forma extremadamente sencilla. Este servidor se incluye en el paquete estándar como servicio de despliegue rápido de soluciones y como ejemplo de aplicación. Por su simplicidad y facilidad de uso agruparemos todas las clases que forman este servidor en un sólo apartado, y no las hemos incluido en ninguno de los diagramas anteriores por considerarse un añadido al núcleo de funcionalidad de esta librería, y no una parte esencial de ésta.

Para el inicio del servicio de acceso remoto (o local) mediante Sockets se hace uso de la clase X10SocketServer. Esta clase hereda de Thread y permite la inicialización de los servidores de monitorización y gestión de la red X10.

Su ejecución se estructura en los siguientes pasos:

1. Inicialización del hilo principal CM11A, junto con la creación de los objetos EventManager y ArrayBlockingQueue que se utilizarán para la comunicación con este hilo.

2. Inicialización del servidor de monitorización en el puerto 2610, pasándole por parámetro un objeto ServerSocket de escucha en el puerto mencionado y el objeto EventManager necesario para la comunicación de eventos en la red.

3. Inicialización del servidor de gestión en el puerto 2611, pasándole por parámetro un objeto ServerSocket de escucha en el puerto mencionado y el objeto ArrayBlockingQueue para el envío de comandos al hilo de ejecución.

El servidor de monitorización se implementa a partir un objeto X10MonitorServer. La clase X10MonitorServer hereda de Thread, y presenta un comportamiento estándar para un servidor de sockets: se mantiene a la escucha de nuevas conexiones, y por cada nueva conexión crea un hilo X10MonitorClient que se encarga de la comunicación con el cliente. Su ejecución termina cuando se ejecuta el método shutdown de X10SocketServer.

Los hilos X10MonitorClient creados a partir de peticiones en el puerto 2610 toman por parámetros el objeto Socket creado para la comunicación con el agente externo así como una referencia al objeto EventManager para la comunicación con la red X10. Al ser inicializados se añadirán a la lista de monitores de EventManager mediante el método addNetworkListener() y proporcionarán a través el Socket información sobre los comandos ejecutados en la red en forma de cadena de caracteres gracias al método toString de los objetos Command obtenidos a través de la pila FIFO proporcionada por el objeto EventManager.

Como ya se ha mencionado en apartados anteriores, la comunicación podría hacerse directamente a través de objetos Command y Unit, o incluso mediante una pila bloqueante común, pero la implementación mediante cadenas de caracteres garantiza la compatibilidad con procesos programados en otros lenguajes como por ejemplo C, y además mantiene la simplicidad en el proceso de comunicación.

El servidor de gestión se implementa a partir de un objeto X10CommandInputServer que sigue el mismo esquema que el anterior servidor. En este caso se inicializan hilos X10CommandInputClient que se encargan de recibir las peticiones de los agentes externos y de comunicárselas al hilo CM11A de control, todo de forma transparente para el proceso cliente.

Los hilos X10CommandInputClient son creados a partir de peticiones en el puerto 2611, tomando como parámetro el objeto Socket creado para la comunicación con el agente externo así como una referencia al objeto ArrayBlockingQueue a través

del cual se implementa la pila FIFO síncrona de comunicación de comandos con la red X10.

Estos hilos leen del socket cadenas de caracteres que representan comandos que transmitir a la red. Gracias al constructor a partir de Strings del objeto Command, la transición de cadena de caracteres a objeto Command es extremadamente simple aún en la programación de este servidor de Sockets. Los comandos aceptados siguen todos la misma estructura, listando primero los dispositivos a los que se dirige el comando, después la función y por último el nivel de acción en porcentaje en caso de ser necesario, todos separados por espacios.

Por ejemplo:

"A1 A2 A7 BRIGHT 70"

El esquema completo formado por estos dos servidores permite que procesos externos puedan comunicarse de una forma extremadamente sencilla con la red X10, y que además esta comunicación se haga mediante uno de los medios más compatibles posibles como es la comunicación mediante cadenas de caracteres.

Es más, un desarrollador que quisiera hacer uso de este servicio no tendría más que familiarizarse con la nomenclatura de direcciones de dispositivo y comandos para poder hacer pleno uso de ello, simplificando enormemente el acceso a esta herramienta.

POSIBILIDADES DE DESARROLLO

Las posibilidades de desarrollo proporcionadas por esta herramienta son varias, y gracias a su facilidad de uso es accesible desde distintos tipos de aplicación. En el siguiente capítulo veremos algunos ejemplos concretos que muestran la facilidad de integración con la librería desarrollada en este proyecto.

Gracias al sistema de comunicación mediante sockets y cadenas de caracteres, aplicaciones desarrolladas en otras plataformas o lenguajes de programación siguen pudiendo comunicarse con la red X10 a través de un servidor domótico. Esta facilidad de integración permite que, por ejemplo, aplicaciones desarrolladas para móviles de última generación, tanto smartphones como modelos que sólo permiten aplicaciones J2ME o Symbian, puedan comunicarse con el servidor domótico sin preocuparse de que el protocolo de comunicaciones no sea compatible con su plataforma o de la integración con una librería más compleja.

Además, la integración mediante pilas FIFO implementadas a través de objetos ArrayBlockingQueue es muy sencilla, al forma parte estos objetos de la distribución general del SDK de Java y ser conocidos ya para los desarrolladores Java. Esto permite la extensibilidad del modelo, de ser necesario, a formatos que hagan uso de CORBA, RMI u otros sistemas de sincronización y comunicación remota más complejos. De nuevo, gracias al uso de medios estándar de comunicación se garantiza la facilidad de integración y modularización de la herramienta.

Por último, gracias a su despliegue como servicio dentro del servidor domótico, esta librería puede desplegarse en servidores multimedia que hagan de home theater en el hogar, liberando así al usuario de estar obligado a mantener encendido su ordenador personal para la comunicación con la red domótica. Los servidores que hacen uso de la plataforma JINI, por ejemplo, son perfectamente compatibles con esta solución, aunque por desgracia no ha sido posible comprobarlo directamente.

Por supuesto, la simplicidad tiene un coste, y no se han implementado muchos servicios que podrían ser deseables, como el almacenamiento de las unidades de un hogar en una base de datos, acceso seguro mediante SSL u otros protocolos de encriptación de comunicaciones y acceso restringido. Esta librería sigue siendo

compatible con desarrollos que necesiten de estas características, pero deja su implementación a los desarrolladores que deseen implementarlos.

Otra limitación es la falta de acceso a la programación de macros en el dispositivo CM11A, y es uno de los grandes puntos reservados para desarrollos futuros de esta librería de acceso a redes X10.

CAPÍTULO 4 - APLICACIONES DE EJEMPLO

INTRODUCCIÓN

En este capítulo veremos algunas de las aplicaciones de ejemplo que se han desarrollado para probar la plataforma. Todas las aplicaciones son sencillas, y ninguna aprovecha al máximo las posibilidades que ofrece el API. Sin embargo, entre ellas sí muestran la facilidad de uso y acceso.

Las aplicaciones de ejemplo muestran la facilidad del desarrollo de aplicaciones tanto locales como remotas y la mínima configuración necesaria para integrarse con la librería desarrollada en este proyecto. Dos de estas aplicaciones implementan un interfaz gráfico de usuario simplificado que muestra la facilidad para monitorizar y gestionar una red X10 de forma simplificada, y sin que el usuario deba llevar a cabo ningún ajuste del programa, haciéndolo inmediatamente accesible.

Adjunto a este documento en su versión digital encontrará el código fuente de todos los ejemplos para su mejor estudio, aunque en general el código es lo suficientemente sencillo como para que no tenga que recurrir a éstos archivos para comprender totalmente el funcionamiento de estos servicios.

APLICACIÓN BÁSICA EN LOCAL

Esta aplicación básica se conecta de forma local directamente al hilo CM11A, ejecutándolo con los parámetros adecuados y tomando la entrada por teclado del usuario como comandos a ejecutar sobre la red domótica X10. Tanto ésta como el resto de aplicaciones de ejemplo están programadas usando la misma versión de Java que la librería (1.6).

Para la programación de esta aplicación es necesario configurar el equipo como se indica en el Anexo 1, y una vez terminada esta configuración.

Además, y debido a que se hace uso de una pila FIFO síncrona para la comunicación con la red X10, la clase en la que se implemente el acceso al hilo CM11A debe extender de la clase java.io.Thread.

Si suponemos que el dispositivo CM11A está conectado al puerto COM1, para conectarnos a él e iniciar el hilo CM11A basta con llamar, desde un try - catch, al constructor de EventManager y de ArrayBlockingQueue y pasárselos para parámetos al constructor de CM11A.

Estos 3 simples comandos dan inicio al hilo CM11A, que utilizará el objeto EventManager para transmitir los eventos que ocurran en la red y el objeto commandStack para recibir comandos por parte del cliente. Cada hilo cliente puede realizar sólo una de éstas tareas, y en este caso nos comunicaremos con el hilo CM11A para ejecutar tareas sobre la red X10. La variable commandStackSize indica la capacidad total de la pila FIFO de entrada. Se recomienda dejar un valor bajo de cara a asegurar un funcionamiento fluido de la aplicación, puesto que la transmisión de datos es bastante rápida y en caso de que esta pila empezase a acumular comandos sería signo de un mal funcionamiento en el enlace con el dispositivo CM11A. En este ejemplo esta variable toma el valor 10.

A partir de éste punto, este programa escucha la entrada de datos a través del teclado y envía la información en forma de objeto Command al hilo CM11A a través de la pila. Podemos almacenar esta información dentro de un objeto String,

pasársela por parámetro al constructor de la clase Command y por último, mediante el método put, introducir el objeto Command así creado en la pila.

Como vemos, con tan sólo otras 2 líneas de código hemos leído un comando de la entrada estándar y añadido a la pila. De hecho, si obviamos la obtención del comando la operación es tan sencilla como crear un objeto Command a partir de la cadena de caracteres e introducirlo en la pila mediante el método put de ArrayBlockingQueue.

Para cerrar el hilo y abandonar la ejecución, basta con llamar al método shutdown del hilo CM11A. Este método se encargará de vaciar el array de datos pendientes de enviar y terminará la ejecución del hilo. Sin embargo, cabe notar que esta operación ignora el buffer de comandos pendientes y da precedencia a abandonar la ejecución lo antes posible.

Si por el contrario queremos un hilo monitor en local, la operación es igual de sencilla. Primero debemos darnos de alta como monitores en el objeto EventManager mediante el método addNetworkListener.

Esta pila, como ya hemos mencionado, es exclusiva del cliente, y en ella recibirá todos los eventos que ocurran en la red en forma de objetos Command.

Para procesar un evento y mostrar por pantalla el comando ejecutado basta con llamar al método take sobre la pila ArrayBlockingQueue, hacer un cast a objeto Command, y si deseamos imprimir por pantalla este comando por pantalla sólo tenemos que llamar al método toString de la clase Command.

Como vemos, una de las limitaciones de esta aproximación es que son necesarios dos hilos en el cliente, uno para la escucha de eventos y otro para el envío de comandos. Los servidores de sockets resuelven este problema, tal y como mostraremos en el siguiente ejemplo.

APLICACIÓN BÁSICA EN REMOTO

La aplicación básica de acceso remoto es aún más sencilla que la anterior, y a diferencia de la aproximación mediante el uso directo de las pilas FIFO en el que eran necesarios dos hilos de ejecución para actuar sobre la red X10 PLC a la vez que se monitorizaba, en este caso no es necesaria la creación de ningún hilo de ejecución puesto que la comunicación se hará vía sockets.

Como vimos en el capítulo anterior, una extensión del núcleo del API es el servidor de sockets X10SocketServer. En esta aplicación de ejemplo abriremos una comunicación con el servidor de sockets, que se supone en ejecución en el servidor domótico, a través del puerto 2611 para el envío de comandos y a través del puerto 2610 para la recepción de eventos en la red.

Si suponemos que la IP del servidor domótico es conocida, el envío de tres comandos básicos se haría como sigue:

- Se crea un nuevo objeto Socket usando la IP del servidor y el puerto de escucha (en este caso el 2611).

- Usamos el constructor de java.io.PrintWriter pasando por parámetro el objeto OutputStream devuelto por Socket.getOutputStream.

- A partir de ahí, podemos usar el objeto PrintWriter así creado para, mediante llamadas al método println, enviar comandos al servidor con el formato que ya hemos mencionado (listado de direcciones separadas por espacios, función y nivel si procede).

- Por ejemplo, si enviamos las cadenas de caracteres “A1 ON”, “A2 A3 OFF” y “A4 BRIGHT 70”, se encenderá el dispositivo A1, apagarán los dispositivos A2 y A3 y aumentará el brillo del dispositivo A4 hasta el 70%.

Como vemos, la comunicación mediante Strings es inmediatamente intuitiva, y al hacer uso de la clase Socket la comunicación se realiza mediante herramientas con las que el desarrollador estará fácilmente familiarizado.

Igualmente sencilla es la recepción de eventos, que se realiza de forma análoga al envío:

- Se crea un objeto java.io.BufferedReader a partir de un objeto java.io.InputStreamReader tomando como parámetro el objeto InputStream devuelto por Socket.getInputStream.

- A partir de ese punto, se pueden leer los comandos ejecutados en la red domótica en formato de cadena de caracteres mediante llamadas al método readLine sobre el objeto BufferedReader creado en el paso anterior. Es necesario, sin embargo, darse cuenta de que las llamadas a readLine() son bloqueantes (comunicación síncrona), y que si se desea realizar una lectura asíncrona es necesaria una aproximación diferente utilizando las herramientas que proporcionan java.net y java.io.

Al cerrar la conexión de los Sockets, los servidores de acceso y monitorización actuarán en consecuencia para evitar la carga innecesaria del sistema, en especial se dará de baja este cliente del listado presente en EventManager y se eliminará la pila FIFO asociada, aunque los comandos enviados antes del apagado seguirán siendo válidos y se ejecutarán correctamente.

Aquí podemos apreciar la verdadera simplicidad alcanzada por esta solución de comunicación mediante intercambio de cadenas de texto, que a diferencia de la anterior no obliga a que el cliente esté programado en Java, ni la importación del paquete protocolo_X10, ni siquiera a tener un conocimiento profundo sobre el funcionamiento interno del servicio, si no una idea general de los comandos aceptados y de los dispositivos presentes en la red.

Por supuesto soluciones más complejas son posibles en caso de ser necesarias, especialmente si queremos que el acceso a la red sea seguro y sus comunicaciones encriptadas. Sin embargo, el objetivo inicial de este proyecto era la simplicidad de acceso y trasladar, aún con limitaciones, la misma facilidad de instalación del hardware al despliegue y control de soluciones software.

APLICACIÓN CON INTERFAZ GRÁFICO DE USUARIO EN LOCAL

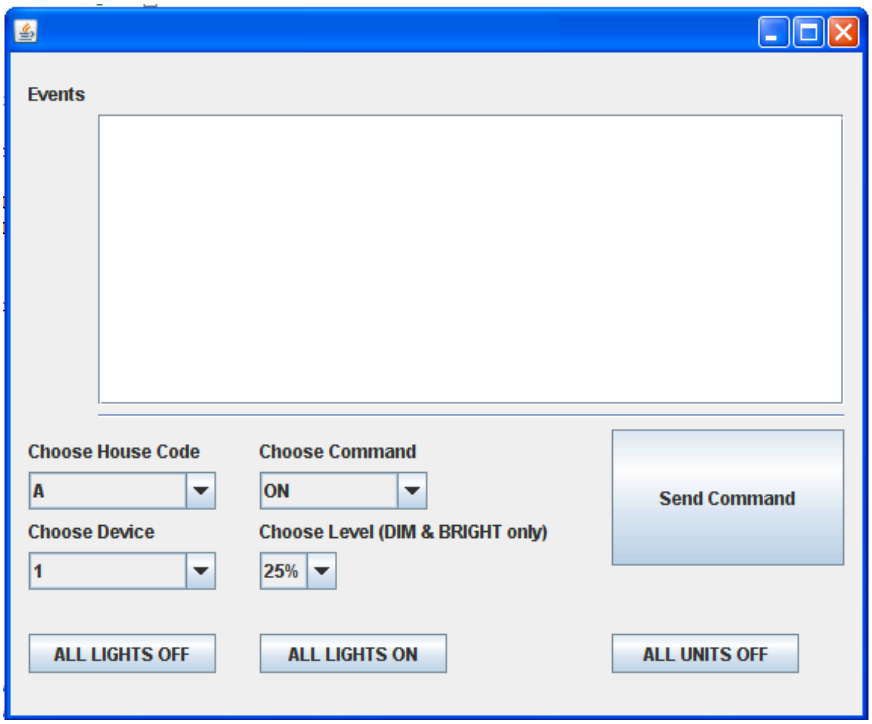

Figura 27 - Interfaz Gráfico de usuario para la aplicación de ejemplo de acceso local

Podemos ver una imagen de este interfaz gráfico de control en la figura superior. Aquí el objetivo era presentar la facilidad de despliegue de aplicaciones más complejas haciendo uso de la librería protocolo_X10.

El desarrollo de este ejemplo está basado en la ejemplo de aplicación básica en local, y hace uso del mismo esquema para lanzar y comunicarse con el hilo CM11A. Sin embargo, en este caso sí se pudo evitar el uso de dos hilos de ejecución diferentes de la forma que explicaremos a continuación.

Esta aplicación de ejemplo extiende de las clases Thread y javax.swing.JFrame para el diseño del interfaz. No es interesante describir aquí los detalles del diseño del interfaz, si no más bien discutir sobre las posibilidades que se abren al utilizar la clase JFrame.

Al pulsar sobre el botón de envío, el comando seleccionado será enviado a la red X10. El código de área, de dispositivo, de comando y el nivel pueden seleccionarse mediante los desplegables que se ven en la figura. Como se puede observar, sólo se permite dirigir el comando a un sólo dispositivo en vez de a varios, pero esta es una limitación propia del interfaz y no de la librería.

Además, los botones ALL LIGHTS OFF, ALL LIGHTS ON y ALL UNITS OFF actuarán sobre todos los dispositivos del código de área seleccionado con el resultado esperado.

La pulsación de cualquiera de estos botones dispara un evento que será tratado de forma interna en la clase X10Frame, e independientemente del método run implementado al heredar la clase de Thread. Gracias a esto se pueden enviar comandos a la vez que se monitoriza la red.

Sin embargo, esta solución no es tan sencilla como podría parecer en un principio, puesto que el panel principal muestra un mensaje con una representación en cadena de caracteres de los comandos enviados y recibidos. Una acción sobre este panel puede por lo tanto invocarse tanto a partir de un evento en el interfaz como de un evento en la red, y por esto los métodos que modifiquen directamente el contenido de este objeto TextArea deben ser sincronizados mediante el uso de la etiqueta synchronized, que evita que dos hilos diferentes llamen simultáneamente a métodos de la clase X10Frame así marcados.

Esta aplicación implementa además un pequeño extra, puesto que busca el puerto al que está conectado el dispositivo CM11A de forma automática. De nuevo, es necesario tomar ciertas precauciones al ejecutar este código de ejemplo puesto que podría provocar un funcionamiento errático de otros dispositivos conectados a los puertos serie y USB del servidor domótico.

Este es quizás el mejor ejemplo de lo sencillo que es el desarrollo de aplicaciones completas que actúan sobre la red X10. La complejidad de esta comunicación es mínima, en especial en comparación con el resto del desarrollo, como el interfaz gráfico. De hecho, esta aplicación de ejemplo demostró funcionar mejor que el software comercial, al demostrar una sorprendente mejor compatibilidad con el dispositivo CM11A y una ejecución de comandos mucho más rápida. Además, esta aplicación puede ejecutarse desde entornos GNU/Linux o derivados de UNIX siempre que estén configurados debidamente.

APLICACIÓN CON INTERFAZ GRÁFICO DE USUARIO EN REMOTO

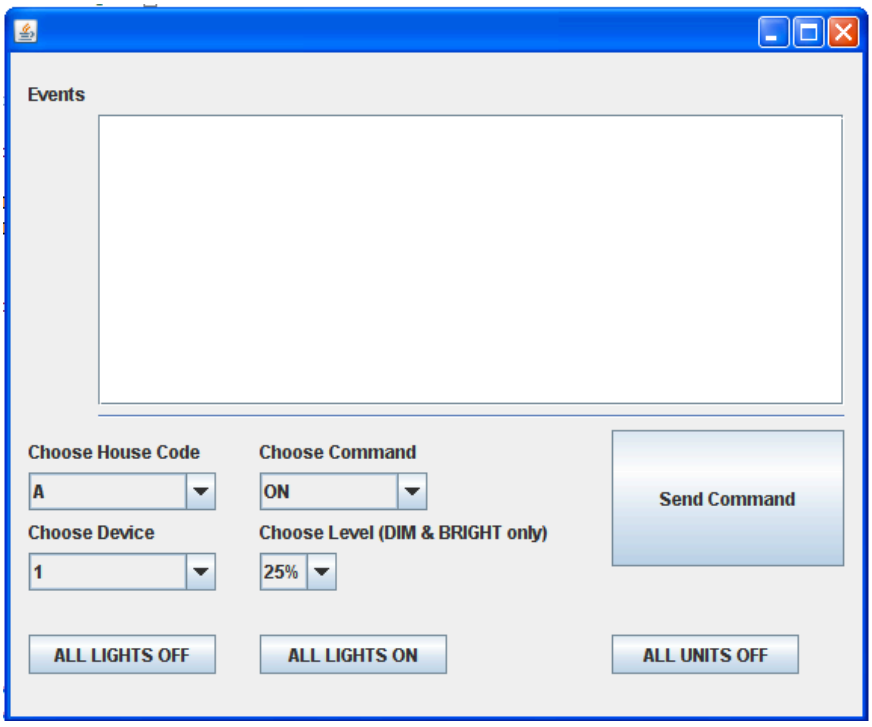

Figura 28 - Interfaz gráfico de usuario para la aplicación de acceso remoto

En este caso se hizo uso del mismo interfaz gráfico que en el ejemplo anterior para el desarrollo de un Applet que tuviese el mismo grado de control que la aplicación local. En este caso la comunicación está basada en el ejemplo de aplicación básica en remoto, y por ello su implementación resultó aún más sencilla que en el ejemplo anterior.

Al realizarse la comunicación mediante sockets y no mediante las pilas FIFO, este Applet no requiere de la instalación en el cliente de la librería protocolo_X10. Además, la comunicación mediante Strings simplifica de forma notable la implementación del envío y recepción de comandos.

En este caso, y por mantener la simplicidad en el despliegue de esta aplicación y por posibles incompatibilidades en el cliente no se hizo uso de métodos sincronizados y el panel principal muestra sólo los comandos ejecutados en la red X10.

Cabe notar, sin embargo, que debido a cómo está implementada la clase Applet ciertos clientes pueden levantar un mensaje de error al tratar de ejecutarlos por no tener los permisos de seguridad necesario y no estar firmada la aplicación de ejemplo.

CAPÍTULO 5 - CONCLUSIONES Y TRABAJO FUTURO

Como hemos visto en los dos primeros capítulos de este proyecto, X10 se presenta como una solución sencilla, de acceso inmediato y de enorme popularidad para instalaciones domóticas en hogares que no requieran de automatismos complejos.

El gran objetivo de este proyecto era el desarrollo de una librería de código abierto que permitiese el acceso a estas redes de la forma más simple posible, manteniendo el mismo esquema de simplicidad que ha hecho de X10 una solución tan popular entre los aficionados al bricolaje.

El acceso mediante sockets e intercambio de cadenas de caracteres es el esquema más sencillo que se nos ocurrió para garantizar la interacción con la red domótica por parte de la mayor variedad de dispositivos y lenguajes de programación posible. Incluso la integración directa mediante pilas FIFO de acceso síncrono es inmediatamente accesible a programadores JAVA, aunque limita el acceso a esta librería a desarrollos realizados en este lenguaje de programación.

Además, esta sencillez de diseño no sólo permite el acceso por parte de una amplia gama de plataformas, sino además la integración de redes X10 con otras redes domóticas como ZigBee. Una de las pruebas que se realizaron fue la integración de una pequeña red de dispositivos que se comunicaban mediante el estándar de radiofrecuencia 802.15.4 con la red X10, apareciendo como dispositivos X10 a todos los efectos y pudiendo ser controlados a través del mismo interfaz de control.

El diseño de la librería protocolo_X10 como servicio permite además el acceso de programas de terceros que, aunque no requieran de una integración completa con los servicios proporcionados por el API, sí necesiten de un acceso puntual a estos servicios y garantizan la compatibilidad con servicios de terceros que se puedan ejecutar en paralelo.

Sin embargo, la sencillez tiene un coste, y la integración de esta librería con sistemas más complejos (middleware como CORBA, esquemas RMI, SOAP, uso de Enterprise Java Beans, etcétera) no es óptima. Además, la transmisión de cadenas de texto por la red no es segura, y la integración de esta librería con esquemas de transmisión de datos seguros como SSL está por implementar.

TRABAJOS FUTUROS

1. Desarrollo de un esquema de acceso seguro a la red X10 mediante SSL, encriptación de mensajes u otros esquemas que se consideren apropiados. (Juan C. Cuevas)

2. Además, un ejercicio de integración con SOAP o RMI sería de mucho interés para la integración del sistema de control X10 con servicios web o móviles.

3. Integración de redes X10 con otras tecnologías domóticas como ZigBee u 802.15.4 está todavía por desarrollar, y creemos que aprovechando los servicios ofrecidos por la librería protocolo_X10 se podrían desarrollar otras librerías similares para ZigBee o KNX/EIB que permitiesen la integración transparente de estas redes dentro de un mismo hogar, en especial la integración de X10 con redes domóticas inalámbricas. (William Wallace)

4. Integración con el API de desarrollo de aplicaciones Bluetooth BLUC3M desarrollado por el mismo autor de este proyecto para la unificación de la comunicación con redes Bluetooth y X10 bajo GNU/Linux.

Hemos cumplido con los objetivos marcados al inicio de este proyecto, en especial en cuanto a sencillez de acceso y transparencia en el uso. Creemos que, aunque existen otras opciones de control de redes domóticas X10 tanto comerciales como de código abierto, ésta presenta el esquema más sencillo para otros desarrolladores y su facilidad de integración con otros sistemas es suficiente como para convertirla en un referente para el acceso rápido e indoloro a las capacidades básicas de las redes domóticas X10.

PROYECTOS SIMILARES

HEYU

Se trata de un programa para el control y monitorización de redes X10 a través de la línea de commandos desde sistemas operativos GNU/Linux, Unix y MAC OS. Fue la inspiración principal para este proyecto, en especial por su sencillez de uso, aunque por desgracia está limitado a solo algunos sistemas operativos y no facilita el control remoto.

http://www.heyu.org/

THE JAVA X10 PROJECT

Este proyecto de código abierto bajo licencia GNU GPL fue una gran fuente de información para el API protocolo_X10, siendo ambos API muy similares en estructura y objetivos. Sin embargo su desarrollo se paró en 2005, aunque en 2008 se integró con el proyecto OpenRemote. Las diferencias principales se encuentran en que, aunque este API permite el control de unidades CM17A de control por radio frecuencia, la comunicación con el hilo principal de comunicación con la red X10 está limitada a otras aplicaciones desarrolladas en Java mediante objetos propios al API y no estructuras estándar, no hace uso de las herramientas de la librería java.concurrent y algunos de los hilos caen en espera activa, no permite la monitorización de la red, la estructura del API es, en mi opinión, innecesariamente compleja y está claramente orientado a aplicaciones Java ejecutadas en el propio servidor domótico.

http://www.agaveblue.org/projects/x10/

OTROS PROYECTOS SIMILARES

Pueden encontrarse listados actualizados de proyectos similares de control de redes domóticas X10 en el sitio web Linux Home Automation para sistemas operativos GNU / Linux.

http://www.linuxha.com/athome/index.html#Software

BIBLIOGRAFÍA

(n.d.). From X10 Ltd. Homepage: http://www.smarthome.com/about_x10.html

(n.d.). From Java J2SE 1.6 API Specification: http://download.oracle.com/javase/6/docs/api/

(n.d.). From Java communications API: http://www.oracle.com/technetwork/java/index-jsp-141752.html

(n.d.). From Eclipse IDE: http://www.eclipse.org/

(n.d.). From Java Main Website: java.sun.com

William Wallace, J. I. *Adding functionality to X10 networks with 802.15.4.*

X10 Ltd. (n.d.). *X10 Transmission Theory.* From http://www.x10.com/support/technology1.htm

X10 Ltd. (n.d.). *Interface Communication Protocol.* From http://www.scribd.com/doc/11949900/X10-Interface-Communication-Protocol

X10 Ltd. (n.d.). *Standard and Extended X10 Code Protocol.* From ftp://ftp.x10.com/pub/manuals/xtdcode.pdf

Juan C. Cuevas, J. M. *El Protocolo x10: Una solución Antigua a Problemas actuales.* Universidad de Málaga, Dpto. Lenguajes y Ciencias de la Computación.

Neida Boscán, R. V. *Protocolos de control de dispositivos domóticos.* Universidad Rafael Belloso Chacin. Telematique.

Rye, D. (n.d.). *X10 history by one of its founders.* From http://www.hometoys.com/news_detail.php?section=view&id=15812638

ANEXO 1 - CONFIGURACIÓN DEL ENTORNO DE DESARROLLO

Describiremos la instalación bajo Windows XP Service Pack 2, aunque la configuración del entorno en otros sistema operativos, en especial UNIX, debería resultar inmediata haciendo uso de esta documentación.

1. Supondremos que se dispone de un ordenador personal con Windows XP Service Pack 2 instalado.

2. Podemos descargar la última versión del SDK de J2SE desde java.sun.com. Sin embargo, a fecha de este proyecto la adquisición de Sun por parte de Oracle ha cambiado gran parte de las URL de descarga. La actual es: http://www.oracle.com/technetwork/java/javase/downloads/index.html, pero puede cambiar.

3. La instalación de este JDK así como el resto de documentación puede encontrarse en http://www.oracle.com/technetwork/java/javase/documentation/index.html.

4. Una vez instalado el JDK, procederemos a la instalación de la librería externa javax.comm que se incluye junto a este proyecto en su distribución digital.

    a. Descomprimimos el archivo javacomm.rar.

    b. En la carpeta commapi descromprimida, buscamos el archivo win32comm.dll y lo movemos a la carpeta \jre\bin del directorio de instalación de Java (por defecto C:\Archivos de programa\Java\jdk1.6.x\, donde x es la versión del jdk descargada).

    c. Dentro de la misma carpeta commapi podemos encontrar el archivo comm.jar que tendremos que trasladar a la carpeta \jre\lib\ del directorio de instalación de Java.

d. Si se hace uso del entorno de desarrollo Eclipse, al crear un nuevo proyecto se debe añadir la librería externa comm.jar al proyecto en el que se vaya a hacer uso de ésta.

e. Si no se hace uso de un entorno como Eclipse, se debe enlazar el archivo comm.jar en el classpath del comando de compilación (javac –cp $JAVA_HOME/jre/lib/comm.jar archivo.java).

5. Para hacer uso de la librería protocolo_X10, se debe seguir el mismo procedimiento que en el caso de la librería javax.com e incluir el directorio donde se encuentren los archivos .class resultado de la compilación de la librería en el classpath de la aplicación que se desee desarrollar.

La conexión de los dispositivos X10 es inmediata, y se recomienda seguir el manual de instrucciones que debería acompañar a los dispositivos que adquiera para formar la red domótica.

La conexión con el dispositivo CM11A puede hacerse mediante USB o puerto serie RS-232.

ANEXO 2 - MANUAL DE USO

USO DE LA LIBRERÍA PROTOCOLO_X10

Para hacer uso de la librería protocolo_X10 desde una clase Java es necesario:

- Configurar el entorno de desarrollo tal y como se menciona en el Anexo 1 - Configuración del entorno de desarrollo.

- Importar la librería mediante import protocolo_X10.*;

- Compilar la clase indicando el path donde estén almacenados los binarios de protocolo X10 en el classpath mediante –cp PATH.

LANZAR EL HILO CM11A

Para crear y ejecutar un hilo de interacción con la red CM11A desde una aplicación Java es necesario:

- Importar la clase ArrayBlockingQueue mediante import java.util.concurrent.ArrayBlockingQueue.

- Importar la librería X10 como se indica en “Uso de la librería protocolo_x10”.

- Crear un objeto EventManager mediante el constructor sin parámetros.

- Crear un objeto ArrayBlockingQueue para el envío de objetos Command a la red domótica X10 mediante el constructor que toma como parámetro un entero indicando el tamaño de la pila. Se recomiendan tamaños pequeños de pila salvo que se espere el acceso simultáneo de gran cantidad de clientes, puesto que el llenado de la pila podría indicar un problema de conexión y de pilas de gran tamaño podrían acarrear desbordamientos de memoria en el servidor y que los clientes tarden más en reaccionar ante este fallo. Para el ejemplo del servidor de Sockets se utilizó un tamaño de pila de 10 elementos.

- Pasar ambos objetos al constructor de la clase CM11A, opcionalmente acompañados de la cadena de caracteres que representa el puerto serie al que está conectado el dispositivo CM11A si se conoce éste.

- Para terminar la ejecución del hilo basta con ejecutar CM11A.shutdown(), que terminará la transmisión del comando en curso y terminará inmediatamente después.

COMUNICACIÓN EN LOCAL MEDIANTE LAS PILAS FIFO

Para comunicarse con el hilo CM11A mediante las pilas FIFO es necesario tener en cuenta que tanto para el envío como la recepción de datos serán necesarios hilos dedicados al ser el acceso a estas pilas bloqueante.

- Importar la clase ArrayBlockingQueue mediante import java.util.concurrent.ArrayBlockingQueue.

- Importar la librería X10 como se indica en “Uso de la librería protocolo_x10”.

- Importar la librería io mediante import java.io.*

- La clase desde donde se ejecuten las llamadas a las pilas deben extender de java.io.Thread.

- Supuesto que el hilo CM11A se ha lanzado como se indica en el apartado "Lanzar el hilo CM11A", se pueden añadir objetos Command a la pila de ejecución mediante llamadas al método put(Object o) sobre el objeto ArrayBlockingQueue creado al lanzar el hilo, es importante notar que este método lanza una excepción de tipo InterruptedException en caso de fallo (normalmente que la pila esté llena). Estos objetos Command pueden crearse a partir de cualquiera de los constructores de la clase.

- Para recibir eventos de la red X10 en forma de objetos Command, es necesario darse de alta como monitor en el objeto EventManager mediante una llamada a su método addNetworkListener sin parámetros, que devolverá un objeto ArrayBlockingQueue desde el que se podrán leer los eventos ocurridos en la red mediante una llamada al método take(), que puede lanzar una excepción de tipo InterruptedException en caso de fallo.

COMUNICACIÓN CON LA RED X10 MEDIANTE LOS SERVIDORES DE SOCKETS

Para lanzar el servidor de Sockets X10SocketServer basta con ejecutar, desde la línea de comandos, en el directorio donde estén los binarios de la librería protocolo_X10, "java X10SocketServer" para lanzar los servidores de escucha de comandos y de monitorización. Por defecto la aplicación utilizará el puerto 2610 para aceptar conexiones de nuevos monitores y el puerto 2611 para aceptar conexiones de nuevos actuadores.

Para indicar los puertos de escucha basta con lanzar el servidor pasándole como argumentos el puerto de escucha de monitores y el de actuadores: "java X10SocketServer 2610 2611".

A partir de ahí, cualquier aplicación puede conectarse a estos puertos para el envío o recepción de comandos desde o hasta la red X10 en forma de cadenas de caracteres que sigan el formato:

[Código de área][Código de dispositivo] [Área][Dispositivo] … [Comando] [Nivel]*

Siendo nivel un parámetro opcional a incluir sólo en los casos de comandos Bright y Dim y que indica el porcentaje de brillo o de atenuación que recibirá el dispositivo, por ejemplo:

A1 A2 A7 BRIGHT 70

Aumentará el brillo de los elementos A1, A2 y A7 un 70%.

El código de área debe ser el mismo en todos los casos, de no serlo el comando actuará sólo sobre los dispositivos que tengan el mismo código de área que el primero listado. El comando

A1 B7 A15 OFF

Apagará sólo los dispositivos A1 y A15, manteniendo B7 su estado.

EJEMPLO DE CONEXIÓN CON EL PUERTO RS-232

Este ejemplo muestra el método setCommPort, que ilustra el uso de la librería javax.comm para establecer y configurar una conexión a través del puerto serie.

```
/** setCommPort sets the current RS.232 Communications Port.
 *
 * @param commPort Name of the Communications Port to use, for example "COM3"
under Windows.
 * @throws IOException Exception thrown when the port can't be successfully
configured.
 */
public void setCommPort (String commPort) throws IOException{

try{
        CommPortIdentifier cpi = CommPortIdentifier.getPortIdentifier(commPort);
        sp = (SerialPort) cpi.open("CM11AController", 10000);
        sp.setSerialPortParams(4800, SerialPort.DATABITS_8,
SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
        sp.addEventListener(this);
        sp.notifyOnDataAvailable(true);
        fromX10 = new DataInputStream(sp.getInputStream());
        toX10 = new DataOutputStream(sp.getOutputStream());
}
catch(NoSuchPortException nspe)
{
        nspe.printStackTrace();
        throw new IOException("No Such Port: " + nspe.getMessage());
}
catch(PortInUseException piue)
{
        throw new IOException("Port in use: " + piue.getMessage());
}
catch(UnsupportedCommOperationException ucoe)
{
        throw new IOException("Unsupported comm operation: " + ucoe.getMessage());
}
catch ( Exception tmle){
        throw new IOException("Too many listeners: "+ tmle.getMessage());
}
}
```

ANEXO 3 – PRESUPUESTO

El presupuesto se ha realizado a partir de los precios de las tiendas online appinformatica.es y electroebay.com. En España no existen muchos distribuidores de productos X10, al ser la domótica considera un hobby relacionado con el bricolaje y sólo realmente popular en Estados Unidos y Canadá, siendo estas dos tiendas online son los únicos distribuidores oficiales de Marmitek en España, el fabricante de los dispositivos que se han utilizado para el desarrollo de la librería protocolo_X10.

La Tabla 18 muestra el presupuesto de equipos domóticos e informáticos actualizado a precios actuales a fecha de presentación de este proyecto. Como equipo informático se ha incluido una referencia a un equipo con los requisitos mínimos para el desarrollo de esta librería, aunque otras opciones son posibles.

Se detallan a continuación las tareas necesarias para la realización del mismo, las horas necesarias para completar éstas de forma satisfactoria y el presupuesto en función de la remuneración media para un Ingeniero Superior de Telecomunicación en la Comunidad Autónoma según el informe PESIT VI de 2005 del Colegio Oficial de Ingenieros de Telecomunicación (COIT), que corresponde a 48 240 euros brutos anuales o de forma equivalente a una remuneración de 23,20 euros brutos por hora. La Figura 29 muestra el diagrama de red del proyecto, que se presenta en vez de el diagrama de Gantt por considerarse éste más claro. La Tabla 19 muestra el listado completo de tareas, la dependencia entre estas y el coste estimado en horas para completarlas de forma satisfactoria. La Tabla 20 muestra el presupuesto final teniendo en cuenta la estimación de 1376 horas totales y el coste por hora indicado antes.

Equipos domóticos X10 e informática			
Producto	**Precio Unitario (euros)**	**Unidades**	**Precio Total**
Interfaz CM11A por Marmitek	79.95	1	79.95 €
Interruptor Carril DIN AD10 por Marmitek	39.95	2	79.9 €
Atenuador de brillo LD11 por Marmitek	39.95	2	79.9 €
Micromódulo regulador LWM1[1]	85.95	1	89.95 €
COOLBOX Starter S140[2]	236.90	1	236.90 €
Marmitek FD Filtro / Acoplador de fases	49.95	1	49.95 €
Presupuesto Total			**616.55 €**

Tabla 18 – Presupuesto de equipos domóticos e informática

[1] Equivalente al micromódulo TMD4.
[2] Incluido como referencia de equipo mínimo recomendado.

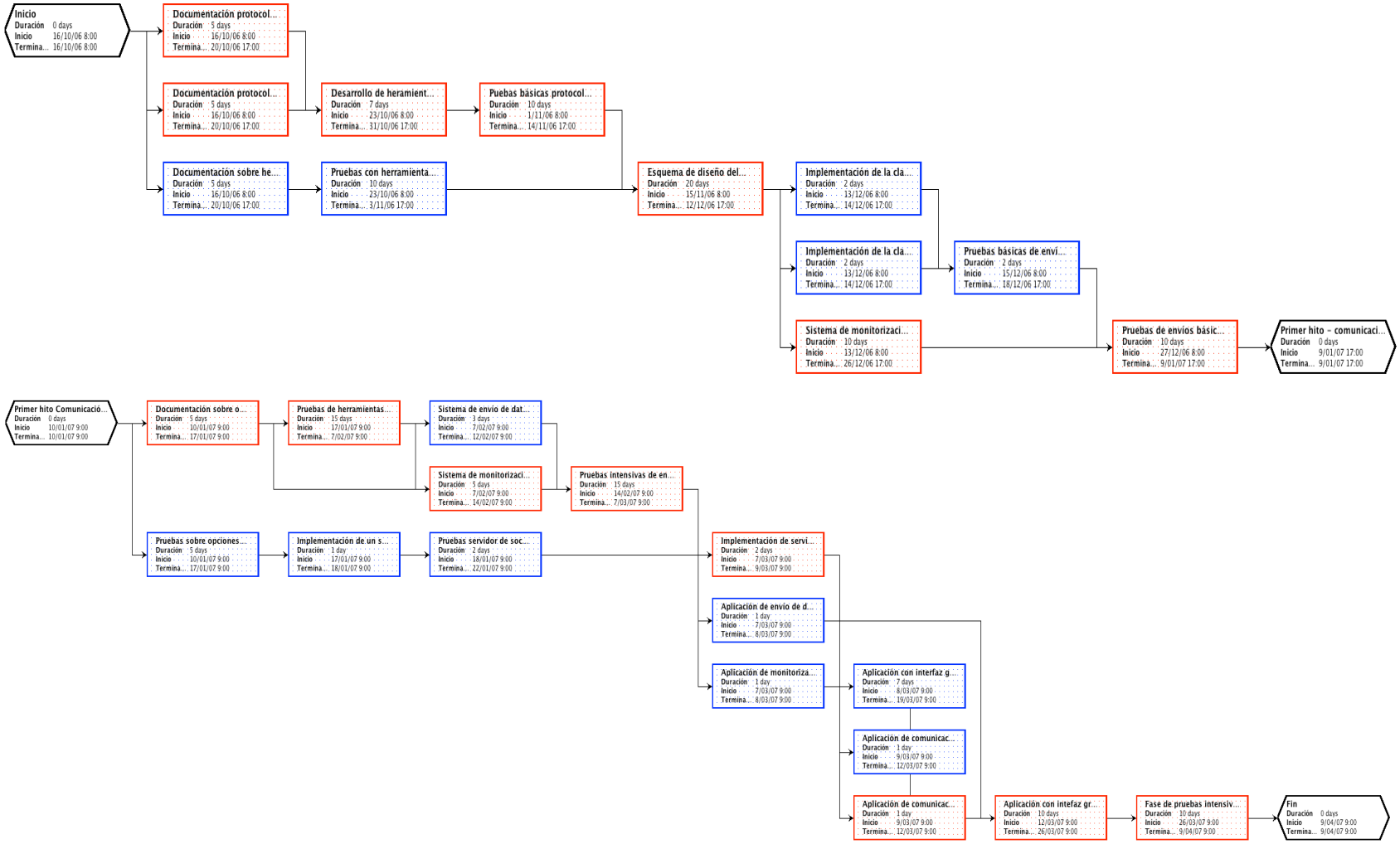
Inicio
Duración 0 days
Inicio 16/10/06 8:00
Termina... 16/10/06 8:00
Documentación protocol...
Duración 5 days
Inicio 16/10/06 8:00
Termina... 20/10/06 17:00
Documentación protocol...
Duración 5 days
Inicio 16/10/06 8:00
Termina... 20/10/06 17:00
Documentación sobre he...
Duración 5 days
Inicio 16/10/06 8:00
Termina... 20/10/06 17:00
Desarrollo de heramient...
Duración 7 days
Inicio 23/10/06 8:00
Termina... 31/10/06 17:00
Pruebas con herramienta...
Duración 10 days
Inicio 23/10/06 8:00
Termina... 3/11/06 17:00
Puebas básicas protocol...
Duración 10 days
Inicio 1/11/06 8:00
Termina... 14/11/06 17:00
Esquema de diseño del...
Duración 20 days
Inicio 15/11/06 8:00
Termina... 12/12/06 17:00
Implementación de la cla...
Duración 2 days
Inicio 13/12/06 8:00
Termina... 14/12/06 17:00
Implementación de la cla...
Duración 2 days
Inicio 13/12/06 8:00
Termina... 14/12/06 17:00
Sistema de monitorizaci...
Duración 10 days
Inicio 13/12/06 8:00
Termina... 26/12/06 17:00
Pruebas básicas de envi...
Duración 2 days
Inicio 15/12/06 8:00
Termina... 18/12/06 17:00
Pruebas de envíos básic...
Duración 10 days
Inicio 27/12/06 8:00
Termina... 9/01/07 17:00
Primer hito - comunicaci...
Duración 0 days
Inicio 9/01/07 17:00
Termina... 9/01/07 17:00
Primer hito Comunicació...
Duración 0 days
Inicio 10/01/07 9:00
Termina... 10/01/07 9:00
Documentación sobre o...
Duración 5 days
Inicio 10/01/07 9:00
Termina... 17/01/07 9:00
Pruebas sobre opciones...
Duración 5 days
Inicio 10/01/07 9:00
Termina... 17/01/07 9:00
Pruebas de herramientas...
Duración 15 days
Inicio 17/01/07 9:00
Termina... 7/02/07 9:00
Implementación de un s...
Duración 1 day
Inicio 17/01/07 9:00
Termina... 18/01/07 9:00
Sistema de envio de dat...
Duración 3 days
Inicio 7/02/07 9:00
Termina... 12/02/07 9:00
Sistema de monitorizaci...
Duración 5 days
Inicio 7/02/07 9:00
Termina... 14/02/07 9:00
Pruebas servidor de soc...
Duración 2 days
Inicio 18/01/07 9:00
Termina... 22/01/07 9:00
Pruebas intensivas de en...
Duración 15 days
Inicio 14/02/07 9:00
Termina... 7/03/07 9:00
Implementación de servi...
Duración 2 days
Inicio 7/03/07 9:00
Termina... 9/03/07 9:00
Aplicación de envio de d...
Duración 1 day
Inicio 7/03/07 9:00
Termina... 8/03/07 9:00
Aplicación de monitoriza...
Duración 1 day
Inicio 7/03/07 9:00
Termina... 8/03/07 9:00
Aplicación con interfaz g...
Duración 7 days
Inicio 8/03/07 9:00
Termina... 19/03/07 9:00
Aplicación de comunicac...
Duración 1 day
Inicio 9/03/07 9:00
Termina... 12/03/07 9:00
Aplicación de comunicac...
Duración 1 day
Inicio 9/03/07 9:00
Termina... 12/03/07 9:00
Aplicación con intefaz gr...
Duración 10 days
Inicio 12/03/07 9:00
Termina... 26/03/07 9:00
Fase de pruebas intensiv...
Duración 10 days
Inicio 26/03/07 9:00
Termina... 9/04/07 9:00
Fin
Duración 0 days
Inicio 9/04/07 9:00
Termina... 9/04/07 9:00

Figura 29 - Diagrama de red del proyecto

#	Descripción	Horas	Dependencias
1	Inicio	0	
2	Documentación protocolo X10	40	1
3	Documentación protocolo CM11A	40	1
4	Desarrollo de heramienta de comunicación por puerto serie, pruebas y depuración	56	2;3
5	Puebas básicas protocolo CM11A	80	4
6	Documentación sobre herramientas similares	40	1
7	Pruebas con herramientas similares	80	6
8	Esquema de diseño del API y diagrama de clases	160	5;7
9	Implementación de la clase Command	16	8
10	Implementación de la clase Unit	16	8
11	Pruebas básicas de envío de comandos a partir de objetos Unit y Command	16	9;10
12	Sistema de monitorización de eventos	80	8
13	Pruebas de envíos básicos y monitorización sin concurrencia	80	11;12
14	Documentación sobre opciones de sincronización	40	1
15	Pruebas de herramientas de concurrencia	120	14
16	Sistema de envío de datos con concurrencia	24	13;15
17	Sistema de monitorización de datos con concurrencia	40	13;15
18	Pruebas intensivas de envío y monitorización con concurrencia, depuración de código	120	16;17
19	Pruebas sobre opciones de comunicación remota	40	1
20	Implementación de un servidor de sockets de intercambio de cadenas de caracteres	8	19
21	Pruebas servidor de sockets con intercambio de objetos Command y Unit	16	9;10;20
22	Implementación de servidores de escucha y gestión	16	18;21
23	Aplicación de envío de datos en local mediante línea de comandos	8	18
24	Aplicación de monitorización de datos en local mediante línea de comandos	8	18
25	Aplicación con interfaz gráfico de usuario y comunicación local	56	23;24
26	Aplicación de comunicación remota mediante línea de comandos para envío de comandos	8	22
27	Aplicación de comunicación remota mediante línea de comandos para monitorización	8	22
28	Aplicación con intefaz gráfico de usuario y comunicación remota	80	24;27
29	Fase de pruebas intensivas	80	23;28
30	Fin	0	29

Tabla 19 – Listado de tareas

Elemento	Coste
Material informático y domótico	616.55 €
Recursos Humanos	31923,2 €
Presupuesto total	**32539,75 €**

Tabla 20 – Presupuesto final