UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de un controlador domótico programable basado en Arduino para un sistema X-10

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Ana María Arrufat Sánchez
Tutor: Vicente Luís Atienza Vanacloig

Curso 2015 - 2016

A mis padres

"La casa es una máquina para vivir"
Le Corbusier, 1921

Resumen

El avance de las Tecnologías de la Información y las Telecomunicaciones (TIC) y su penetración en prácticamente todos los ámbitos de nuestra vida ha hecho que en los últimos años se haya hablado con mucha frecuencia de *casas inteligentes*. Ahorro y eficiencia energética, entretenimiento y comodidad son solo, a muy grandes rasgos, algunas de las cosas que se pueden conseguir gracias a la domótica.

Sin embargo, las alternativas existentes actualmente en el mercado son caras de implantar, sobre todo en viviendas ya construidas. Con este trabajo se pretende ofrecer una alternativa más asequible que ofrezca estas características a los habitantes de una vivienda.

Para conseguirlo, se ha hecho uso de dos microcontroladores de código abierto: Arduino e Intel Galileo. Por una parte, Arduino se va a encargar de hacer de interfaz con los dispositivos que forman el sistema domótico (luces, electrodomésticos, etc.). Por la otra, Galileo va a albergar un pequeño servidor que será el verdadero encargado de gestionar el sistema, además de proporcionar una interfaz para interactuar con la vivienda accesible a través de una API REST.

Este sistema de automatización se completa con el desarrollo de una aplicación Android que proporciona una interfaz sencilla desde la que cualquier usuario, independientemente de su nivel de conocimiento en informática y electrónica, podrá gestionar y controlar todo lo que ocurra en su vivienda y que se comunicará con el servidor a través de su API.

Para realizar pruebas se ha construido un pequeño prototipo que cuenta con un sensor de presencia y permite controlar el comportamiento de una lámpara y una persiana. El resultado obtenido ha sido el de un sistema que realmente cumple con las expectativas iniciales: fácil de utilizar y económico.

Palabras clave: Domótica, X-10, Android, Arduino, Galileo

Abstract

The Information Technology and Telecommunications (ICT) advancement and its penetration into virtually every area of our lives has made in recent years it has been talked frequently about *smart homes*. Energy saving and efficiency, entertainment and comfort are just, very roughly, some of the things that can be achieved thanks to home automation.

However, alternatives currently available on the market are expensive to implement, especially in existing homes. This academic work aims to offer a more affordable alternative that offers these features to the inhabitants of a house.

To achieve this, I have made use of two open source microcontrollers: Arduino and Intel Galileo. On the one hand, Arduino is going to be responsible for interfacing

with devices that form the home automation system (lights, appliances, etc.). On the other hand, Galileo will hosts a small server that will be the real responsible for managing the system in addition to providing an interface to interact with the house reachable through a REST API.

This automation system is completed with the development of an Android application that provides a simple interface from which any user, regardless of their level of knowledge in information technology and electronics, can manage and control everything that happens in his home and that will communicate with the server through its API.

For testing, it has been built a small prototype that has a presence sensor and can control the behaviour of a lamp and a blind. The result has been a system that truly meets the initial expectations: easy to use and economic.

Palabras clave: Home automation, X-10, Android, Arduino, Galileo

Resum

L'avanç de les Tecnologies de la Informació i les Telecomunicacions (TIC) i la seua penetració en pràcticament tots el àmbits de la nostra vida ha fet que en els últims anys s'haja parlat amb molta freqüència de *cases intel-ligents*. Estalvi i eficiència energètica, entreteniment i comoditat son a només, a grans trets, algunes de les coses que es poden aconseguir gràcies a la domòtica. Però, les alternatives existents actualment son cares d'implantar, sobretot en habitatges ja construïts. Amb aquest treball es pretén oferir una alternativa més assequible que oferisca (Apunte: oferisca està aceptado en valenciano, però no en catalan, por eso corrige) aquestes característiques als habitants d'un habitatge.

Per aconseguir-ho, s'ha fet us de dos microcontroladors de codi obert: Arduino i Intel Galileo. D'una banda, Arduino es va a encarregar de fer d'interfície amb els dispositius que formen el sistema domòtic (llums, electrodomèstics, etc.). Per l'altra banda, Galileo va a albergar un xicotet servidor que serà el vertader encarregat de gestionar el sistema a més de proporcionar una interfície per a interactuar amb l'habitatge accessible a través d'una API REST.

Aquest sistema d'automatització es completa amb el desenvolupament d'una aplicació Android que proporciona una interfície senzilla des de la qual qualsevol usuari, independentment del seu coneixement en informàtica i electrònica, podrà gestionar i controlar tot el que ocórrega en el seu habitatge i que es comunicarà amb el servidor a través de la seua API.

Per a realitzar probes s'ha construït un xicotet prototip que compta amb un sensor de presència i permet controlar el comportament d'una llum i d'una persiana. El resultat obtingut ha sigut el d'un sistema que realment compleix amb les expectatives inicials: fàcil d'utilitzar i econòmic.

Paraules clau: domòtica, X-10, Android, Arduino, Galileo

Tabla de contenidos

Lista de figuras

Lista de tablas

Lista de siglas y acrónimos

API	*Application Programming Interface*
DIY	*Do It Yourself*
E/S	Entrada/Salida
IDE	*Integrated Development Environment*
IoT	*Internet of Thins*
JSON	*JavaScript Object Notation*
NTP	*Network Time Protocol*
OCU	Organización de Consumidores y Usuarios
PLC	*Power Line Communications*
REST	*Representational State Transfer*
RTC	*Real Time Clock*
TIC	Tecnologías de la Información y las Comunicaciones
UGI	*User Graphical Interface*
UART	*Universal Asynchronous Receiver-Transmitter*
URI	*Uniform Resource Identifier*
XML	*Extensible Markup Language*

1. Introducción

1.1. Justificación del proyecto

El vertiginoso avance tecnológico experimentado en los últimos años ha revolucionado la vida contemporánea en todos sus aspectos. Centrándonos en el ámbito de las tecnologías de la información y la comunicación, la aparición de lo que se conoce como «el Internet de las cosas» (en inglés, *Internet of things*, o IoT) ha cambiado completamente nuestra manera de interactuar con el mundo que nos rodea.

Este concepto, que nació en el Instituto de Tecnología de Massachusetts (MIT), hace referencia a las relaciones establecidas entre los objetos y las personas a través de Internet. Una forma de interconexión que permite ofrecer ingentes cantidades de datos en tiempo real y que supone una revolución a la hora de digitalizar el mundo físico que nos rodea.

En esta línea, en los últimos años han ido apareciendo multitud de dispositivos y servicios que aportan no solo confort, conectividad y mejores oportunidades de ocio a la vida de las personas, sino también seguridad, accesibilidad y ahorro energético. El futuro apunta hacia un paradigma llamado «inteligencia ambiental», que consiste en dotar al hogar de un nivel de conectividad e inteligencia tal que se adapte automáticamente a las necesidades de sus ocupantes.

La domótica permite dar respuesta a los requerimientos que plantean estos cambios sociales y las nuevas tendencias de nuestra forma de vida, facilitando el diseño de casas y hogares más personalizados, multifuncionales y flexibles.

Sin embargo, las soluciones existentes actualmente en el mercado para poder disfrutar de las ventajas que ofrece un sistema de automatización de este tipo son caras de implantar, sobre todo en viviendas ya construidas. Por ello, sería interesante contar con alguna alternativa que ofrezca estas características a los habitantes de una vivienda a un precio más asequible.

Además, estos sistemas suelen ser sistemas cerrados y muy poco flexibles. Si, por ejemplo, quisieramos ahorrar dinero en sensores y en su lugar consultar los datos ambientales en Internet nos resultaría del todo imposible. Por esta razón resulta de gran interés desarrollar un sistema totalmente abierto, ampliable y personalizable.

A todo esto se une la rápida extensión del uso de los *teléfonos inteligentes* y la ampliación de su funcionalidad que hacen muy interesante su uso como dispositivo de control y comunicación con el sistema domótico.

1.2. Motivación personal

La motivación inicial fue la idea de trabajar con Arduino, un microcontrolador de código abierto basado en hardware y software flexibles y fáciles de usar. Desde el primer momento en que me puse a buscar tema para mi Trabajo de Fin de Grado quería encontrar algo en esta línea, pues es un tema que me resulta de gran interés y sobre el que llevo trabajando desde hace un tiempo.

Además del interés que supone trabajar con una plataforma de tan amplia difusión, me parece una manera muy entretenida de aprender electrónica y programación de microcontroladores y en el grado no se ha tenido ninguna asignatura relacionada con ello.

Por otro lado, la idea de utilizar este microcontrolador para crear algo tan en boga hoy en día como lo son las casas inteligentes me atrajo en el momento. A esto se sumó la necesidad de aprender a programar en Android para poder desarrollar la aplicación que controla todo el sistema, puesto que ha sido para mí una asignatura pendiente desde hace un par de años.

1.3. Objetivos

El objetivo del proyecto será **desarrollar un sistema de automatización para el hogar programable** basado en el protocolo X-10 que permita implementar una alternativa económica a las opciones existentes actualmente en el mercado. Para ello se va a desarrollar un software que actúe de intermediario entre el usuario y los dispositivos que formen parte del sistema domótico y permita la gestión, control y programación del mismo.

Mediante una placa Intel Galileo [14] se va a desarrollar un controlador domótico que podrá recibir órdenes de programación y control a través de una conexión Ethernet y enlazará con los actuadores X-10 haciendo uso de un dispositivo de interfaz XM10.

El interés de hacer uso de una placa Intel Galileo deriva del hecho de que es una plataforma compatible con la arquitectura Arduino que ofrece mejores prestaciones en cuanto a velocidad de proceso, capacidad de memoria, conectividad, etc. características que resultarán muy convenientes para este trabajo. Además, se dispone actualmente de algunas unidades de forma gratuita gracias a un programa de colaboración con las universidades desarrollado por el fabricante.

Se quiere lograr que dicho sistema sea «amigable», es decir, de uso fácil e intuitivo por parte de un usuario que no tenga amplios conocimientos sobre electrónica ni informática (aunque si serán necesarios unos conocimientos mínimos). Para conseguirlo se ha optado por desarrollar una aplicación para dispositivos Android que permita al usuario gestionar fácilmente todo el sistema.

Para desarrollar este software de automatización, se va a partir de un proyecto de código abierto ya existente [17] para Arduino que se replanteará para adaptarlo a una plataforma Intel Galileo Gen2 y para permitir la programación de escenarios desde el propio dispositivo móvil. Se pretende que estos escenarios estén formados por una serie de acciones que deben llevarse a cabo cuando se cumplan unas determinadas condiciones. Por ejemplo, podría programarse un escenario que se encargara de encender la calefacción si la temperatura desciende de 15 grados, u otro que se encargara de encender el sistema de riego del jardín todos los días a las 20:00.

Por último, para lograr un precio asequible, se pretende reducir el número de dispositivos a comprar. Con este fín se ha optado por aprovechar la conectividad a Internet de la placa Galileo y sustituir los sensores de temperatura, humedad, etc,

por consultas a algún servicio web que proporcione datos meteorológicos en tiempo real.

1.4. Estructura de la memoria

La presente memoria se divide en diez secciones. A continuación, paso a describir brevemente qué se va a tratar en cada una de ellas.

La primera sección está compuesta por este apartado, junto con los tres anteriores, formando la **Introducción**, donde se describe cuál ha sido la motivación para realizar el proyecto y se explica brevemente cual es el objetivo y los temas que van a ser tratados a lo largo de la memoria.

La segunda sección, **Estado del arte**, se explica en qué consiste la domótica, qué beneficios aporta a una vivienda y se hace un breve análisis sobre su situación actual y qué tendencias se esperan para los próximos años.

En la sección tres, **Tecnologías empleadas**, se describen las diferentes herramientas empleadas en el desarrollo del proyecto y sus principales características.

En la sección cuatro, **El protocolo de comunicaciónes**, se describen los protocolos de comunicaciones más utilizados en el mundo de la domótica en la actualidad, justificando la elección del protocolo X-10 y describiendo en más profundidad el funcionamiento y características de este último.

En la sección cinco, **Análisis del sistema**, se describen los requisistos funcionales del sistema, para lo cual se han identificado y descrito en detalle los diferentes casos de uso.

En la sección seis, **Plan de trabajo, problemas y soluciones**, se detallan los pasos que se han ido siguiendo a lo largo de la realización del proyecto, así como los problemas encontrados y qué soluciones se han implantado para solventar estos problemas.

En la sección siete, **Diseño hardware**, se describe cuál va a ser el diseño del hardware que formará parte del sistema de automatización, explicando brevemente su funcionamiento.

En la sección ocho, **Diseño software**, se presenta el diseño de la lógica de la aplicación, el diseño de la persistencia y el diseño de la interfaz gráfica de usuario o GUI (del inglés *Graphical User Interface*), lo que incluye el diseño de la aplicación Android que se va a utilizar para gestionar y controlar todo el sistema.

En la sección nueve, **Implementación física**, se describe cual ha sido el prototipo utilizado para realizar el proyecto, incluyendo algunas imágenes que ilustran la descripción. También se presenta el desglose de cuál ha sido el coste total de dicho prototipo.

En la sección diez se exponen las **Conclusiones** a las que se ha llegado tras la realización del proyecto, así como una valoración personal acerca de los objetivos conseguidos y el trabajo que se podría seguir desarrollando en el futuro.

Por último se incluye una sección de **Anexos** entre los que se encuentra un manual del usuario, la documentación de la Interfaz de Programación de Aplicaciones (API) desarrollada y una guía inicial de puesta en marcha.

2. Estado del arte

2.1. ¿Qué es la domótica?

El término domótica viene de la unión de la palabras *domus*, que significa casa en latín, y el sufijo *tica*, de automática, palabra en griego que significa «que funciona por sí sola». La Asociación Española de Domótica e Inmótica[1] (CEDOM) define la domótica como el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta seguridad y confort, además de comunicación entre el usuario y el sistema. Otros la definen como la integración de la tecnología en el diseño inteligente de un espacio.

En lo que todos coinciden es que se necesitan elementos de automatización y al menos cierto nivel de intercomunicación entre los diferentes componentes de la casa para poder llamársele domótico. Es decir, se trata de aplicar la tecnología de la automatización y el control al hogar.

Convertir una casa normal en un hogar domótico cuesta entre 1.500 y 3.000 euros, lo que equivale a entre el 1,5 % y el 2 % del coste de las instalaciones de una vivienda, una cantidad que se amortiza en cuatro años gracias al ahorro energético que permiten estos equipamientos [19]. El coste final depende de muchos aspectos. Instalaciones hay de muchos tipos, podría decirse que tantas como usuarios, y cada una requiere sus características específicas en cuanto a funcionalidad.

Domintell. empresa especializada en el sector de la automatización, presenta el siguiente cuadro de precios de refencia en su página web:

	En Euros		
Tipo de vivienda	**Equipamiento estandar**	**Equipamiento lujoso**	**Equipamiento premium**
Apartamento 2 habitaciones	2.200	5.900	8.800
Unifamiliar 3 habitaciones	4.200	8.900	12.700
Chalet 4 habitaciones	9.000	12.600	18.700

Figura 1: Coste de una instalación domótica Domintell

[1] La Inmótica es equivalente a la domótica, pero aplicada a edificios no destinados a vivienda, como hoteles, centros comerciales, escuelas, universidades, hospitales y todos los edificios terciarios.

2.2. ¿Qué ofrece la domótica?

Según la CEDOM los servicios que ofrece la domótica pueden agruparse en cinco categorías [9]:

1. **Programación y ahorro energético**: gestiona inteligentemente la iluminación, climatización, agua caliente, el riego, los electrodomésticos, etc., aprovechando mejor los recursos naturales, utilizando las tarifas horarias de menor coste, y reduciendo así, la factura energética.

2. **Confort**: a través de la gestión de dispositivos y actividades domésticas. La domótica permite abrir, cerrar, apagar, encender, regular, etc. los electrodomésticos, la climatización, ventilación, iluminación natural y artificial, persianas, toldos, puertas, riego, suministro de agua, gas, electriciad y muchas cosas más que frecen un hogar más confortable a sus inquilinos.

3. **Seguridad**: vigilancia automática de personas, animales y bienes, así como de incidencias y averías. Mediante controles de intrusión, cierre automático de todas las aberturas, simulación dinámica de presencia, cámaras de vigilancia, y a través de alarmas técnicas que permiten detectar incendios, fugas de gas, inundaciones de agua, fallos del suministro eléctrico, etc.

4. **Comunicaciones**: mediante el control y supervisión remotos de la vivienda a través de su teléfono, ordenador personal, etc. que permite la recepción de avisos de anomalías e información del funcionamiento de equipos e instalaciones.

5. **Accesibilidad**: facilita el manejo de los elementos del hogar a las personas con discapacidad de la forma que más se ajuste a sus necesidades, además de ofrecer servicios de teleasistencia para aquellos que lo necesiten.

2.3. Situación actual

Mantener la casa segura es la principal razón por la que la mayoría de usuarios instala tecnología domótica en sus hogares, según ha revelado un informe realizado por la empresa D-Link. Los resultados de la investigación realizada en 20 países europeos a través de más de 8.500 encuestados muestran que las cinco razones principales para comprar e instalar un dispositivo inteligente en casa son:

1. La seguridad (44 % de los encuestados).

2. Automatizar la vivieda (36 % de los encuestados).

3. Controlar remotamente electrodomésticos y dispositivos (26 %).

4. Ahorrar en el consumo eléctrico (22 % de los encuestados).

5. El entretenimiento (15 % de los encuestados).

En España, el desplome de la construcción producido tras el estallido de la burbuja inmobiliaria en 2008, caída que se mantiene hasta el año 2013, está directamente ligado al estancamiento que ha sufrido la domótica. Durante esos años el mercado sufrió un ajuste muy severo cuyo resultado fue la desaparición de muchas empresas, especialmente las que habían apostado por la Domótica y la Inmótica en grandes promociones de viviendas y en edificios.

Sin embargo, según un estudio realizado por CEDOM [10] el sector de la domótica ha dejado de caer y comienza a impulsarse su implantación.

2.4. Tendencias futuras

Año 2016: Potenciar el ahorro y la eficiencia energética

Existen multitud de encuestas y estudios realizados sobre el sector de la domótica que estudian en qué situación se encuenta el sector y qué evolución se espera del mismo en los próximos años, y todos parecen estar de acuerdo en que a partir del año 2016 la domótica va a experimentar un gran crecimiento.

Una de las principales razones que motivan este pensamiento es que la ciudadanía está cada vez más concienciada con la eficiencia y el ahorro energético. El consumo energético de las viviendas supone un gasto importante en la economía familiar, además del impacto negativo sobre el medio ambiente, debido a la generación de residuos asociados a dicho consumo.

Según la Organización de Consumidores y Usuarios (OCU), en el año 2012 una casa española gastaba en energía una media de 990 euros en consumo energético anual. También muestra, que una vivienda española consumía anualmente unos 9.922 kilovatios por hora (kWh), lo que equivale a 0,85 toneladas de petróleo al año. Y cada vez se consume más energía. La tendencia mundial es a duplicar y triplicar dicho consumo.

La domótica ofrece un sistema de gestión que permite optimizar el uso de la energía, por ejemplo, controlando las persianas para ahorrar en calefacción en invierno y aire acondicionado en verano, apagando las luces en habitaciones en las que no haya gente, etc. Por tanto, actualmente la demanda del mercado es hacia sistemas con soluciones enfocadas al ahorro y a la mejora de la eficiencia energética.

Otro aspecto importante en el que la domótica y la automatización experimentará mayores crecimientos es el relacionado con la seguridad y la vigilancia de viviendas. El informe *DIY Home Automation Market by Offerings, by Technology, and by Geography* [2] prevé que el crecimiento de la domótica orientada a particulares en todo el mundo, de cara al 2016 y años siguientes, va a estar impulsada en gran medida por el segmento de la seguridad y el control de accesos.

A esta previsión se une la consultora ABI Research. En uno de sus estudios cifró en 1,7 millones de euros el montante de negocio en ventas de los sistemas y equipos

[2]Reconocido informe que analiza la tendencia del mercado del «hazlo tu mismo» o DIY (del inglés, *Do It Your Self*)

domóticos de seguridad y monitorización a finales del año 2015. Pues bien, en el 2016 se espera que esas cifras lleguen a los 11,3 millones de euros.

Año 2018: Fuerte incremento de las tecnologías inalámbricas

Las previsones señalan que a partir del año 2018 las tecnologías inalámbricas van a tomar un papel importante en el crecimiento de los sistemas domóticos. Cada vez son más los dispositivos domóticos inalámbricos que aparecen en el mercado y más los hogares conectados a internet. Según ABI Research, en 2018 está previsto que haya 500 millones de dispositivos domóticos inalámbricos en nuestros hogares (frente a los 17 millones de dispositivos que había en 2013).

La gran ventaja de los dispositivos inalámbricos es que no se precisan costosas obras y la instalación suele ser bastante sencilla, además de fácil de manejar, presentándose como una opción muy atractiva para los usuarios. Si la posibilidad de realizar auténticas casas inteligentes con dispositivos mayoritáriamente inalámbricos toma forma real en los próximos años, el sector dará un salto muy importante.

Año 2020: La era de la información

Según la empresa americana Gartner, líder mundial en investigación de tecnologías de la información, en 2020 habrá en el mundo aproximadamente veintiseis mil millones de dispositivos con un sistema de adaptación al internet de las cosas. ABI Research, por otro lado, asegura que para el mismo año existirán 30 mil millones de dispositivos inalámbricos conectados a Internet.

Los hogares crearán información de prácticamente todos los dispositivos, facilitando una correcta gestión por parte de los usuarios e información cualitativa a los fabricantes para seguir ofreciendo las mejores soluciones. Este incremento en la generación de información vendrá acompañado de mejoras en los procesos de seguridad, sector que invertirá grandes cantidades de recursos.

3. Tecnologías empleadas

3.1. Arduino Uno

Arduino es una placa programable con entradas y salidas digitales y analógicas, cuyo bajo coste la hace ideal para iniciarse en automatización o realizar pequeños proyectos domésticos en electrónica y robótica. Esto significa que disponemos de un pequeño «autómata», capaz de recibir información del entorno (sensores) y realizar acciones (actuadores, motores, etc), según un programa que introducimos con un ordenador, y que puede ejecutar de forma autónoma.

Arduino es en realidad tres cosas:

1. **Una placa de *hardware* libre**, consistente en una placa de circuito impreso con un microcontrolador reprogramable, usualmente Atmel AVR, y puertos digitales y analógicos de entrada/salida, los cuales pueden conectarse a placas de expansión (llamadas *shields*) que amplían las características de funcionamiento de la placa Arduino. Asimismo posee un puerto de conexión USB desde donde se puede alimentar la placa y establecer comunicación serie con un ordenador.

2. **Un *software* gratis, libre y multiplataforma** que permite escribir, compilar y guardar en la memoria del microcontrolador de la placa Arduino el conjunto de intrucciones que deseamos que este empiece a ejecutar.

3. **Un lenguaje de programación libre**: tanto el entorno de desarrollo (o IDE, del inglés Integrated Development Kit) como el lenguaje de programación de Arduino están inspirados en otro entorno y lenguaje libre existente: Processing[3].

Existen multitud de modelos Arduino disponibles. La placa Arduino UNO es el modelo más estándar y la que se ha empleado durante el desarrollo de este proyecto. En la siguiente tabla se presentan los modelos de Arduino más habituales, con sus características más importantes y un precio de referencia:

	Uno Rev3	Leonardo	Mega Rev3	Mini 05
E/S digitales	16	20	54	14
Entradas analógicas	6	12	16	6
Salidas analógicas	6	7	14	8
Número de puertos serie	1	1	4	1
Memoria flash	32 kb	32 kb	256 kb	32kb
Precion en €	20	16	35	14

Tabla 1: Comparativa de placas Arduino

[3]Lenguaje de programación descendiente de Java orientado a diseñadores y específicamente diseñado para el desarrollo de arte gráfico, animaciones y aplicaciones gráficas de todo tipo.

3.1.1. Características principales

Arduino se diferencia de otros microcontroladores existentes en el mercado por tener las siguientes características:

- **Es libre y extensible**. Cualquiera que quiera ampliar o mejorar tanto el diseño hardware de las placas como el entorno de desarrollo o el propio lenguaje de programación puede hacerlo sin problema. Gracias a esta característica existe un amplio abanico de placas no oficiales y librerías software que aumentan la oferta y las posibilidades de creación del programador.

- **Existe un activa comunidad de usuarios** que continuamente comparten ideas y resuelven dudas a otros usuarios.

- **Es un entorno multiplataforma**, se puede ejecutar en Windows, Linux y Macintosh.

- **Las placas Arduino son muy económicas**, como se ha visto en la tabla 1. Además, al ser *hardware* libre si alguien lo desea puede descargar el diagrama del circuito, comprar todos los componentes y crear su propia placa, con lo que el precio total sería mucho menor. Esto la convierte en una plataforma excelente para práctica y aprendizaje, pues cualquier usuario inexperto puede utilizarla sin temor a romperla.

- **Se programa mediante cable USB**, no a través de puerto serie. Esto es my útil ya que la mayoría de los equipos modernos no disponen de puertos serie.

- Tanto el entorno de desarrollo como el lenguaje de programación son **fáciles de aprender y utilizar**, a la vez que flexibles y completos para que usuarios avanzados puedan explotar todas las posibilidades que ofrece el *hardware*.

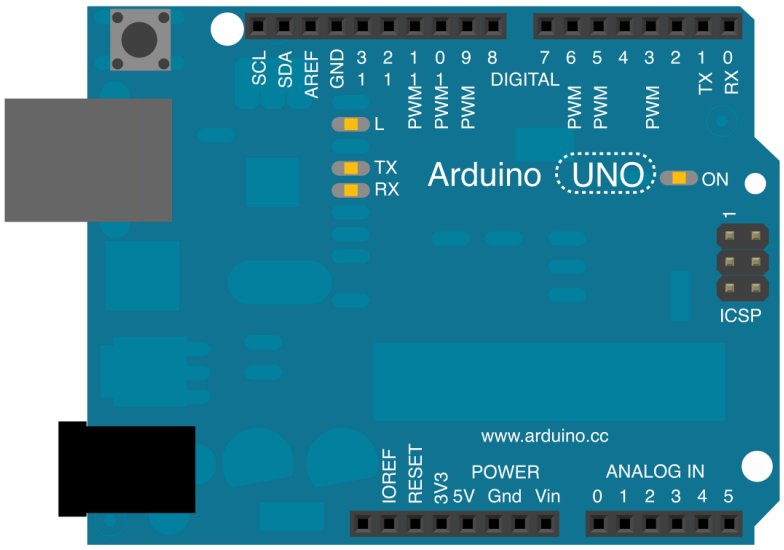


Figura 2: Esquema de la placa Arduino Uno Rev3

3.1.2. El hardware de Arduino Uno

La placa Arduino está formada por:

- **14 pines de entrada/salida digial (pines 0 a 13):** pueden ser pines de entrada o salida, lo que se especifica mediante el *sketch* creado en el IDE.
- **6 pines de entrada analógica (pines de 0 a 5):** estos pines dedicados aceptan valores analógicos (es decir, lecturas de voltaje desde un sensor) y los convierten en un número comprendido entre 0 y 1023.
- **6 pines de salida analógica (pines 3, 5, 6, 9, 10 y 11):** son los marcados con la etiqueta «PWM». Estos son, en realidad, seis pines digitales que se han reprogramado para salida analógica usando el *sketch* creado en el IDE.

Pines de alimentación

- **3.3V**: proporciona una tensión de 3,3 voltios y una intensidad máxima de 50 mA, generada por el chip FTDI integrado.
- **5V**: proporciona una tensión de 5 voltios y una intensidad máxima de 300 mA, la utilizada para alimentar el microcontrolador y otros componentes de la placa. La tensión puede venir del pin VIN, a través de un regulador integrado en la placa, o ser suministrada por el USB u otra fuente de alimentación regulada de 5V.
- **GND**: es la toma de tierra, o nivel 0 voltios de referencia.
- **VIN** (a veces etiquetado como «9V»): proporciona la tensión máxima con la que está alimentado Arduino cuando se usa una fuente de alimentación externa (en oposición a los 5V de la conexión USB u otra fuente de alimentación regualda). A través de este pin se puede suministrar tensión a la placa o si el suministro de tensión proviene del conector jack, acceder a él.

3.1.3. Un programa Arduino

Un programa Arduino, comúnmente llamado *sketch*, necesita de dos bloques para su compilación:

- **El bloque *setup***: la parte de código incluida en la función `void setup(){}` se ejecuta una sola vez cuando comienza el programa, es decir, cada vez que se presiona el botón *reset* o se enciende la placa. Es el método encargado de la inicialización del programa.

  Se suele emplear para determinar si un determinado pin es entrada o salida, establecer su valor inicial, inicializar el puerto serie, etc. Aún cuando no sea necesario escribir nada en él, es necesario que aparezca y añadir las llaves de apertura y cierre `{}`.

- **El bloque *loop***: se ejecuta justo después del bloque *setup* y, como su propio nombre indica, la función `void loop(){}` se ejecuta de forma ininterrumpida, una y otra vez. Con este bucle se logra que el programa responda ante los distintos eventos que se produzcan.

A continuación se incluye un ejemplo de *sketch* que, repetidamente, enciende un led durante un segundo y luego lo apaga durante otro segundo:

```
// LED conectado al pin digital 13
int LED = 13;

void setup() {
  // inicializa el pin digital 13 como salida
  pinMode(LED, OUTPUT);
}

void loop() {
  // enciende el LED (HIGH es el nivel de voltaje)
  digitalWrite(LED, HIGH);
  // espera un segundo
  delay(1000);
  // apaga el LED haciendo que la tensión sea baja
  digitalWrite(LED, LOW);
  // espera un segundo
  delay(1000);
}
```

3.2. Intel Galileo

La Intel Galileo es el fruto de la colaboración entre la Fundación Arduino e Intel. Basada en la arquitectura x86 de Intel tiene una potencia similar a aquellos primeros procesadores Pentium de mitad de los años noventa. Es el primer paso de la nueva familia Intel Spark, una gama de placas de desarrollo compatibles con Arduino basadas en arquitecturas Intel, y resulta ideal para proyectos de domótica, *wearables* y el Internet de las Cosas.

Es una placa de desarrollo de hardware libre. Se basada en el procesador Quark SoC X1000 de 32 bits de Intel con una velocidad de 400 megahercios (MHz). Está diseñada para ser compatible con el IDE de Arduino y con las Arduino *Shields*. Su *hardware* incluye los mismos pines que un Arduino Uno Rev 3, además de un conector Ethernet, un zócalo para tarjetas microSD, puerto USB, puerto serie RS-232, un puerto mini PCI Express (mPCIE) y 8 megabytes de memoria flash.

La diferencia entre la Galileo y una placa Arduino estándar la marca el hecho de poder combinar la estructura de hardware y software de Arduino con el sistema operativo Linux. Gracias a esto, podemos controlar hardware como sensores o motores con otros lenguajes de programación como Python o NodeJS, conectarlos

a Internet, crear un servidor o tener acceso a fecha y hora real, entre otras muchas posibilidades de computación comunes en una plataforma con la arquitectura x86. Por lo tanto, la Intel Galileo está enfocada sobre todo al desarrollo de proyectos sobre el **Internet de las Cosas**.

3.2.1. El hardware de Intel Galileo

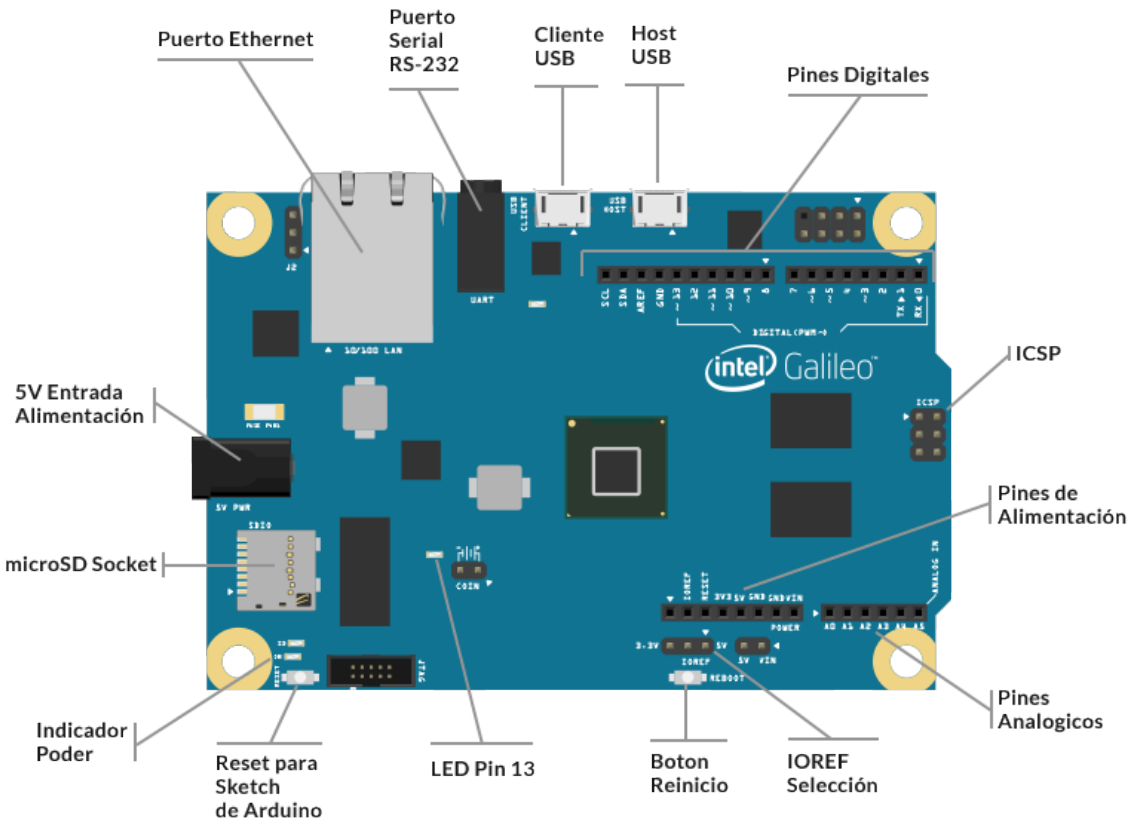


Figura 3: Esquema de la placa Intel Galileo Gen2

Algunas de las características más interesantes de esta placa son:

- **Intel Quark SoC X1000**: el cerebro de Galileo es este procesador Intel Pentium de 32 bits que puede funcionar a velocidades de hasta 400 MHz, y cuenta con 512 KB de SRAM incorporada.
- **Arduino**: Intel Galileo es la primera placa basada en Arduino desarrollada sobre una arquitectura Intel y compatible con todas los *shields* de Arduino. La placa incluye el mismo pinout que un Arduino Uno Rev 3:
  - 14 pines de entrada/salida digitales (6 de estos con salida PWM)
  - 6 pines entradas analógicas (A0 – A5)

  - 8 pines de potencia (5V, 3V, GND, reset, reboot)
  - SPI: 10(SS), 11(MOSI), 12(MISO), 13(SCK).
  - I2C: A4 o pin SDA y A5 o pin SCL. Soportan la comunicación I2C (TWI).
  - ICSP Header: para conectar Arduino Shields

- **Ethernet**: conector Ethernet de 10/100. La Galileo es capaz de conectarse a Internet a través del protocolo de configuración dinámica de host (o DHCP, del inglés *Dynamic Host Configuration Protocol*).

- Tiene un **IDE conocido**: desde el propio IDE de Arduino se pueden programar sketches para esta placa. Además el IDE permite llamadas de firmware de Linux a la programación de esquema de Arduino a través de la orden `system(command)`, donde `command` puede ser cualquier comando Linux. Por ejemplo: `system("ls -l /home > /dev/ttyGS0")` lista los ficheros de la carpeta home y redirige la salida al puerto serie.

- **Compatibilidad con las librerías Ethernet**: utilizar el puerto Ethernet de la Galileo es tan simple como utilizar la librería Ethernet de Arduino.

- **Reloj de Tiempo Real** (RTC, *Real Time Clock*): la mayoría de las tarjetas de Linux requieren una conexión internet para obtener la fecha y la hora actuales. Pero el reloj en tiempo real de Galileo puede seguir midiendo el tiempo aún cuando la tarjeta esté desconectada, solo tienes que conectar una batería de 3V a la tarjeta cuando la Galileo no tenga alimentación.

- **Trabaja con tarjetas PCI Express Mini**: debajo de la tarjeta cuenta con una ranura de expansión mini-PCI express de tamaño completo. Esto significa que es posible conectar tarjetas WiFi, Bluetooth, GSM, etc, para ampliar la conectividad de Galileo.

- **Cliente USB**: para programar y cargar *sketches* Arduino.

- **Puerto USB 2.0**: que permite utilizar la librería USB de Arduino y convertir a la Galileo en un ratón o teclado. Con un hub USB se pueden conectar hasta un máximo de 128 dispositivos en este puerto.

- **Soporte MicroSD**: zócalo para tarjetas microSD para poder usarlas con la librería SD de Arduino. También sirve para cargar el Linux SO a la Galileo.

- **Comunicación serie UART**[4]: disponible en el pin digital 0 (pin RX, recepción) y 1 (pin TX, transmisión).

- **Conectividad serie**: aparte de los pines 0 y 1 Galileo tiene un puerto serie separado que permite conectarse a la línea de comandos de Linux desde el ordenador. Esto se puede hacer desde el jack de audio contiguo al puerto de Ethernet (puerto serial RS-232).

- **Almacenamiento**:

[4]Transmisor-Receptor Asíncrono Universal: es el dispositivo que controla los puertos y dispositivos serie. Se encuentra integrado en la placa base o en la tarjeta adaptadora del dispositivo.

  - 8 MByte de memoria SPI Flash que almacena el firmware, de los cuales entre 256 kB y 512 kB se utilizan para almacenar el sketch Arduino.
  - 256 MBytes de DRAM.
  - 8 KBytes de memoria EEPROM, compatible con la librería EEPROM de Arduino.
  - MicroSD hasta un máximo de 32 GB.
  - Almacenamiento USB a través del puerto USB.

- **Alimentación**: la Galileo se alimenta por el conector jack 2,1mm o por los pins Vin y GND. Es importante destacar que a diferencia de la mayoría de placas Arduino, la Galileo se alimenta solo a 5V y conectar más voltaje podría dañar la placa.
- **Linux en Galileo**: una versión muy ligera de Linux está cargada en la memoria flash de 8 MegaBytes (MB).

3.3. Linux e Intel Galileo

El sistema Linux es muy flexible y puede ser adaptado para propósitos específicos. Hay un gran número de versiones personalizadas de Linux hechas para las tarjetas Intel Galileo, algunas creadas por los propios usuarios. A continuación se presentan algunas de las principales distribuciones:

- **Imagen SPI**: es la versión preinstalada que se actualiza con el IDE Arduino y que no necesita una tarjeta SD para arrancar. Esta *mini* distribución, tiene funcionalidad limitada. Por ejemplo, no tiene los controladores para los módulos WiFi y no cuenta con la característica de persistencia en los sketch Arduino cuando se reinicia la tarjeta. Este sistema inicia apenas arranca la tarjeta.
- **Imagen «SD-Card»**: esta versión requiere una tarjeta SD y proporciona características que no tiene la imagen SPI., como los controladores para los módulos WiFi además de soporte para: OpenCV, ALSA y Node.js por nombrar solo algunos ejemplos.
- **Imagen «IoT Devkit»**: esta versión tiene características mas interesantes. Por ejemplo: es mas fácil realizar conexiones de red, tiene mas herramientas para desarrollar y soporta OpenCV para python. Su instalación resulta un poco mas compleja.
- **Debian e imagenes personalizadas**: estas versiones son generadas por los usuarios y cada una tiene sus ventajas y métodos de instalación.

3.4. Software de Intel

Además de poder trabajar con el IDE de Arduino para crear sketches, Intel ofrece un entorno de desarrollo que junto con la distribución «IoT Devkit» proporciona las herramientas necesarias para poder desarrollar proyectos que utilicen otras tecnologías como NodeJs o Python. Se trata del IDE *Intel XDK IoT Edition*.

Este entorno de desarrollo puede ser usado tanto para crear aplicaciones web como aplicaciones para dispositivos móviles en forma nativa y proporciona plantillas de código para crear aplicaciones que interactúan con sensores, aceleradores, cámaras y los componentes propios de cada dispositivo.

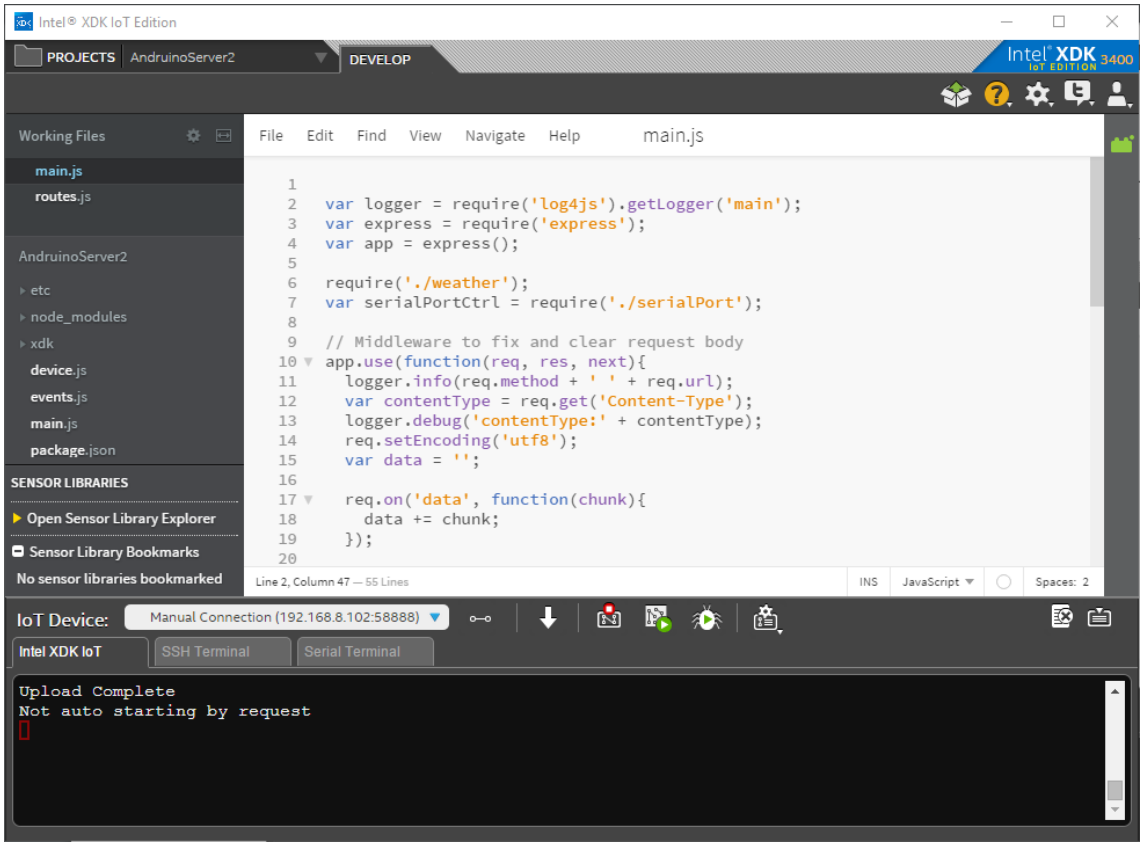


Figura 4: Intel XDK

3.5. NodeJS

NodeJS es un intérprete de Javascript del lado del servidor, cuya meta es permitir a un programador construir aplicaciones altamente escalables y escribir código que maneje decenas de miles de conexiones simultáneas en una sóla máquina física. Uno de los puntos fuertes que ha contribuido al crecimiento de su popularidad reside en que JavaScript es un lenguaje omnipresente, conocido por millones de desarrolladores que no tendrán que aprender un nuevo lenguaje para empezar a construir aplicaciones usando NodeJS.

En NodeJS el código se organiza por medio de módulos, concepto que se asemeja a los paquetes o librerías de otros lenguajes como Java. El gestor de paquetes de node, o npm (siglas del inglés Node Package Manager), se diferencia de otros gestores en que, por defecto, instala los módulos localmente en los proyectos. Para poder utilizarlo se debe hacer desde la línea de comandos de NodeJS con la orden: `npm install modulo`, donde `modulo` es el nombre que recibe el paquete que se quiere instalar.

Por ejemplo:

```
npm install async
```

instalará el paquete async dentro del proyecto, en la carpeta «node_modules», y a partir de ese momento estará disponible y se podrá incluir con la sentencia «`require`»:

```
var require = require("async");
```

Construido sobre el intérprete de JavaScript V8 creado por Google, NodeJS resulta una alternativa muy interesante para programar el IoT: dirigido por eventos asíncronos, con un modelo de entrada y salida no bloqueantes, escalable, ligero, con buen rendimiento y una comunidad de usuarios tremendamente amplia y activa detrás de él resulta excelente para desarollar aplicaciones de tiempo real y/o que manejen grandes volúmenes de información. Poco más se puede pedir para ponerse a trabajar con IoT.

La efectividad, interactividad y seguridad en las comunicaciones son de vital importancia en el mundo del Internet de las cosas. Por esta razón las APIs se adaptan muy bien al modelo del IoT. El mayor y más aceptado uso para NodeJs es la creación de APIs. Por ejemplo, lo primero que desarrolló la empresa LinkedIn usando esta tecnología fue su API para móviles.

Existen multitud de frameworks diseñados para crear APIs con NodeJS, aunque no es imprescindible utilizarlos y con solo un par de líneas de código se puede crear una API para empezar a desarrollar en NodeJS.

También es importante señalar que la comunidad de usuarios de NodeJS tiene una gran afinidad con las tecnologías del IoT, y que los primeros usuarios del IoT tienen tendencia a utilizar NodeJs en sus productos y experimentos. De hecho, el repositorio de módulos de Node es un indicador de la asociación entre estas comunidades. Incluye más de 80 paquetes para el controlador Arduino, más de 15 para Bluetooth de bajo consumo y múltiples paquetes para los dispositivos *wearables* Pebble and Fitbit.

3.6. Express y los servicios REST

La Transferencia de Estado Representacional o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la *World Wide Web* que considera los servicios web como recursos que «verdaderamente están en la web» y con cuyas capacidades se interactúa por medio del protocolo HTTP.

Sobre esos recursos podemos realizar acciones, generalmente diferenciadas a través de los distintos verbos HTTP: GET, PUT, POST y DELETE. Por lo tanto, REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

Express es un framework que se ejecuta por encima de Node y que permite crear servidores web y recibir peticiones HTTP de una manera sencilla, lo que facilita la creación de APIs REST de forma rápida.

En express una ruta se define siguiendo el siguiente patrón:

```
app.method(path, handler)
```

donde:

- `app` es una instancia de express.
- `method` es un método de solicitud HTTP.
- `path` es una vía de acceso a un recurso en el servidor (URI).
- `handler` es la función que se ejecuta cuando un cliente accede a la ruta.

Un ejemplo muy sencillo de lo que podría ser una API REST escrita con Node y el framework Express es el siguiente:

```
var express = require("express");
var app = express();

app.get("/", function(req, res){
  res.send("Recibida una petición GET!");
});

app.post("/", function (req, res) {
  res.send("Recibida una petición POST!");
});

app.put("/", function (req, res) {
  res.send("Recibida una petición PUT!");
});

app.delete("/", function (req, res) {
  res.send("Recibida una petición DELETE!");
});

app.listen(3000, function () {
  console.log("Servidor escuchado en el puerto 3000");
});
```

3.7. JSON

Según *Ecma International* [11] (organización basada en membresías de estándares para la comunicación y la información) la Notación de Objetos JavaScript o JSON (acrónimo del inglés *JavaScript Object Notation*), es un formato de texto

que facilita el intercambio de datos estructurados entre todos los lenguajes de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

Aunque JSON nació para llevar la notación nativa usada para declarar objetos en JavaScript a otros lenguajes, también se le ha conseguido dar usos tan alejados de su origen como el de almacenar datos en bases de datos. MongoDB es el ejemplo perfecto. MongoDB en lugar de disponer de las típicas tablas, dispone de colecciones, que son ni más ni menos que objetos JSON que podemos definir a nuestro antojo y de forma totalmente independiente a como lo hacemos en cualquier otra colección

Las ventajas de JSON sobre XML como formato de intercambio de datos son que ocupa menos espacio, es más ligero y es mucho más sencillo escribir un analizador sintáctico. En JavaScript, un texto JSON se puede analizar fácilmente usando la función `eval()`, lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

A continuación se muestra un ejemplo de uso de JSON en JavaScript:

```
// Ejemplo de texto JSON
var json_string = '{
  "nombre":"Santiago Sánchez",
  "edad":27,
  "nacionalidad":"Español",
  "altura":"172 cm",
  "peso":75
  }';

// JSON.parse() analiza una cadena de texto como JSON
// Retorna el objeto que se corresponde con el texto JSON entregado.
var object = JSON.parse(json_string);

//
JSON.parse('{}');              // {}
JSON.parse('true');            // true
JSON.parse('"foo"');           // "foo"
JSON.parse('[1, 5, "false"]'); // [1, 5, "false"]
JSON.parse('null');            // null
```

Los tipos de datos disponibles con JSON son:

- Números: se permiten números negativos y opcionalmente pueden contener parte fraccional separada por puntos. Por ejemplo: 123.456
- Cadenas: representan secuencias de cero o más caracteres. Se ponen entre doble comilla y se permiten cadenas de escape. Por ejemplo: `"Hola"`
- Booleanos: representan valores booleanos y pueden tener dos valores: `true` y `false`

- null: representa el valor nulo.
- Vectores: un vector representa una lista ordenada de cero o más valores que pueden ser de cualquier tipo. los valores se separan por comas y el vector se mete entre corchetes. Por ejemplo: `[1,2,3]`
- Objetos: son colecciones no ordenadas de pares de la forma `<nombre>:<valor>` separados por comas y puesto s entre llaves. El nombre tiene que ser una cadena. El valor puede ser de cualquier tipo. Un ejemplo podría ser:

```
{
  "id": 1,
  "name": "Leanne Graham",
  "email": "Sincere@april.biz",
  "address": {
    "street": "Kulas Light",
    "city": "Gwenborough",
    "zipcode": "92998-3874",
    "geo": {
      "latitude": -37.3159,
      "longitude": 81.1496
    }
  },
  "phone": 615879586,
  "adult": true,
  "employees": [
    {
        "name": "Ervin Howell",
        "email": "shanna@melissa.tv"
    },
    {
        "name": "Clementine Bauch",
        "email": "nathan@yesenia.net"
    },
  ]
}
```

3.8. Android

Android es un sistema operativo desarrollado inicialmente por Android Inc., empresa que Google respaldó económicamente y acabó comprando en 2005. Está basado en Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma. Características que lo convierten en uno de los sistemas operativos más utilizados en todo el mundo.

Según datos de la empresa Kantar (empresa líder mundial en estudios de mercado, investigación y análisis) en Europa Android ocupa el 92,8 % de la cuota de

mercado del sector, seguido del iOS de Apple Inc. con el 6,9 % y windows phone con apenas el 0,2 % [6].

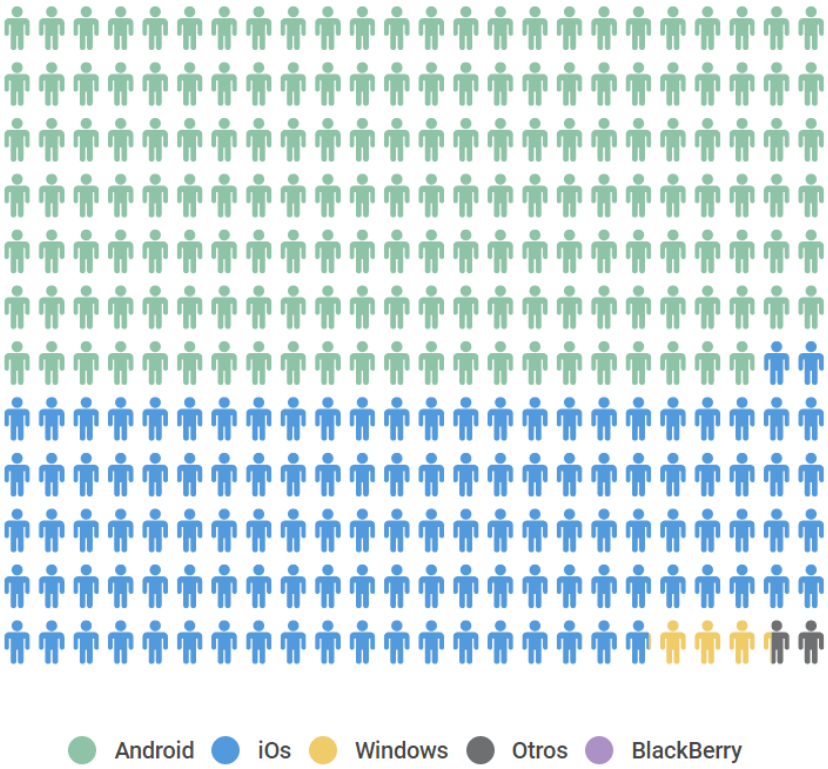


Figura 5: Cuota de mercado europea de *smartphones* en base a su sistema operativo

La diferencia de Android con respecto a todos los demás sistemas operativos es que es un software de código abierto, no cerrado, característica irrelevante para el usuario pero de vital importancia de cara al mercado. Software de código abierto es aquel cuyo código fuente y otros derechos que normalmente son exclusivos para quienes poseen los derechos de autor, son publicados bajo una licencia de software compatible con la Definición de Código Abierto[5] (*Open Source Definition*) o forman parte del dominio público. Esto permite a los usuarios utilizar, cambiar o mejorar el software y redistribuirlo, ya sea en su forma modificada o en su forma original. Frecuentemente se desarrolla de manera colaborativa y los resultados se publican en internet.

Esta caracteristica es de especial interés para los desarrolladores, quienes pueden experimentar y probar, mientras que cada fabricante puede introducir sus particularidades. Por esta razón, no es exactamente igual Android en el teléfono Nexus, por ejemplo, que en un Samsung. Ambos utilizan Android pero los usuarios de un Galaxy tienen ciertas características particulares que la compañía ha decidido añadir con el objetivo de diferenciarse de su competencia. Por tanto, no es de extrañar que en torno al 80 por ciento de los dispositivos móviles que existen en el mundo funcionen con Android.

[5]https://opensource.org/osd

3.8.1. Las ventajas de Android

En *El gran libro de Android* [13] se presentan las siguientes características que distinguen este sistema operativo de entre todos los existentes para teléfonos inteligentes:

- **Plataforma realmente abierta**: puede ser usada y personalizada sin pagar derechos de autor.
- **Adaptable a cualquier tipo de hardware**: Android no ha sido diseñado exclusivamente para su uso en teléfonos y tabletas. Hoy en día se pueden encontrar relojes, gafas, cámaras, televisiones, sistemas para automóviles, electrodomésticos y una gran variedad de sistemas empotrados que se basan en este sistema operativo, lo cual tiene sus evidentes ventajas, pero también va a suponer un esfuerzo adicional para el programador. La aplicación ha de funcionar correctamente en dispositivos con una gran variedad de tipos de entrada, pantallas, memoria, etc.
- **Portabilidad asegurada**: las aplicaciones finales son desarrolladas en Java lo que nos asegura que podrán ser ejecutadas en cualquier tipo de CPU. Esto se consigue gracias al conecpto de máquina virtual.
- **Arquitectura basada en componentes inspirados en Internet**: por ejemplo, el diseño de la interfaz de usuario se hace en XML, lo que permite que una misma aplicación se ejecute en un reloj de pantalla reducida o en un televisor.
- **Filosofía de dispositivo siempre conectado a Internet**: muchas aplicaciones solo funcionan si disponemos de una conexión permanente a Internet.
- **Gran cantidad de servicios incorporados**: como localización basada tanto en sistema de posicionamiento global (GPS) como en redes, bases de datos con SQL, reconocimiento y síntesis de la voz, navegador, etc.
- **Aceptable nivel de seguridad**: los programas se encuentran aislados unos de otros gracias al concepto de «ejecución dentro de una caja», que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de actuación (localización, acceso a Internet, lectura de la agenda de contactos, etc.).
- **Optimizado para baja potencia y poca memoria**: en el diseño de Android se ha tenido en cuenta el *hardware* específico de los dispositivos móviles.
- **Alta calidad de gráficos y sonido**: gráficos vectoriales suavizados, animaciones, gráficos 3D basados en OpenGL. Incorpora los codecs estándares más comunes de audio y vídeo, incluyendo H.264 (AVC), MP3, AAC,

4. El protocolo de comunicaciones

Para que dos dispositivos se comuniquen no solo es necesario interconectarlos a través de un medio de comunicación, sino que también es necesario que hablen el mismo idioma, es decir, que utilicen el mismo protocolo de comunicación. El protocolo define el formato del mensaje y el lenguaje común a todos los dispositivos para que se puedan comunicar entre ellos.

4.1. Principales estándares en domótica

Desde los inicios de la domótica ha habido una carrera constante por parte de los fabricantes y agrupaciones de empresas del sector por establecer estándares de fabricación. En la actualidad los que han logrado imponerse a nivel mundial son: LonWorks de LonMark Association, KNX de Konnex Association y X-10.

LONWorks

El protocolo LONWorks, también conocido como LONTalk o Estándar de Control de Red ANSI/EIA 709.1, es un protocolo que proporciona un conjunto de servicios de comunicación que permiten al programa de aplicación de un dispositivo enviar y recibir mensajes de otros dispositivos sobre una red de control sin necesidad de conocer la topología de la red ni los nombres, direcciones o funciones de otros dispositivos. El estándar ha sido ratificado por el Instituto Nacional Estadounidense de Estándares[6] (ANSI) como oficial en Octubre de 1999.

El principal propósito del sistema LONWorks es crear de forma sencilla, eficiente y con bajo coste, sistemas abiertos de control. Los objetivos de la empresa Echelon eran:

1. Proporcionar las herramientas necesarias para que cada fabricante pueda desarrollar su propio producto
2. Asegurar la compatibilidad e interoperabilidad de productos desarrollados por distintos fabricantes.
3. Utilizar diversos medios de transmisión. Lonworks no está sujeto a buses físicos. En este sentido se podría comparar con TCP/IP, ya que sus mensajes pueden transmitirse por cualquier soporte conocido (RF, fibra, TCP/IP, RS-232, RS-485. RS-422, el cableado eléctrico, etc.). Para integrar dispositivos en una red Lonworks basta con usar el transceptor adecuado.

[6]Organización sin ánimo de lucro que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos. ANSI es miembro de la Organización Internacional para la Estandarización (ISO).

El estándar LONWork consiste en un conjunto de dispositivos inteligentes (también llamados nodos) que se conectan mediante uno o más medios físicos y que se comunican utilizando un protocolo común. Por inteligente se entiende que cada nodo es autónomo y proactivo, de forma que puede ser programado para enviar mensajes a cualquier otro nodo como resultado de cumplirse ciertas condiciones o para llevar a cabo ciertas acciones en respuesta a los mensajes recibidos.

El funcionamiento completo de la red surge de las distintas interconexiones entre cada uno de los nodos. Mientras que la función desarrollada por uno de los nodos puede ser muy simple, la interacción entre todos puede dar lugar a implementar aplicaciones complejas. Uno de los beneficios inmediatos de LON es que un pequeño número de nodos pueden realizar un gran número de distintas funciones dependiendo de cómo estén interconectados.

En el envío de mensajes entre nodos existe una jerarquía de direccionamiento que incluye dirección de dominio, subred y nodo. Cada nodo está conectado a un canal. Un dominio es una colección lógica de nodos que pertenecen a uno o más canales. Una subred es una colección lógica de hasta 127 nodos dentro de un dominio, pudiéndose definir hasta 255 subredes dentro de un único dominio. Resulta fácil entrever que la capacidad de crecimiento de las redes LON es casi ilimitado.

La empresa Echelon proporciona un paquete software que facilita la tarea de programar los diferentes nodos que conforman la red LON. *LonMaker* es la herramienta de diseño, instalación y mantenimiento de la red mientras que *LonBuilder Developer's Workbench* es la que suministra la información de configuración de la misma.

Sin embargo, una gran desventaja de este protocolo es que se requiere experiencia y conocimiento en la materia para llevar a cabo la instalación y programación del sistema. Aunque Echelon ofrece un software muy robusto su utilización no resulta sencilla para un usuario inexperto.

KNX

KNX es una tecnología que apareció a principios de los 90 de la mano de tres protocolos domóticos existentes: Batibus, EIB y EHS, que se unieron en 1997 en un único estándar internacional al que bautizaron con el nombre de KNX. Su objetivo era crear un único estándar europeo para la automatización de viviendas y oficinas capaz de competir en calidad, prestaciones y precio con otros sistemas como LonWorks, CEBus o el estándar americano de convergencia SCP. La tecnología KNX está respaldada por la KNX *Association*, un grupo de compañías líderes en el mercado y activas en muchas áreas de aplicación relativas al control de casas y edificios.

En el sistema KNX la transmisión de las señales se hace a través de un bus de datos al que están conectados todos los dispositivos. De esta forma, es posible que cualquier componente dé órdenes a cualquier otro, independientemente de la distancia entre ellos y su ubicación. Para interconectar los dispositivos del bus en cada línea se permite cualquier tipo de topología: árbol, estrella, bus o anillo. Una

línea puede tener conectados un máximo de 64 dispositivos.

Una de sus mayores ventajas consiste en la libertad de elección en el amplio número de fabricantes KNX. Actualmente, la KNX *Association* tiene más de 170 empresas en 29 paises, que ofrecen más de 7.000 grupos de productos certificados KNX en sus catálogos, englobando más del 80 % de los dispositivos vendidos en Europa para el control de casas y edificios.

Otra gran ventaja de este protocolo es que posee una única herramienta que es independiente de la aplicación y del fabricante llamada *«Engineering Tool Software»* (ETS) que permite el diseño, la implementación y la configuración de una instalación que posea productos certificados KNX.

Sin embargo, este sistema posee dos grandes desventajas. La primera es la dificultad del manejo de la ETS. Aunque es una herramiento muy robusta no resulta sencilla de manejar, no se pueden simular situaciones pre-programadas y no está orientada a objetos con los que se pueda interactuar.

La segunda gran desventaja es el coste de los productos KNX que, por lo general, suelen ser caros en comparación con otros productos de domótica. Además, a parte de las molestias derivadas de la necesidad de realizar obras en la vivienda, la necesidad de instalar nuevo cableado encarece el precio de la instalación.

KNX resulta de especial interés en viviendas de nueva construcción, ya que el coste que supone el lanzar un cableado específico es sobrepasado con creces por las ventajas que ofrece el tener un bus dedicado.

En edificios ya construidos no presenta buenas prestaciones estéticas, pues necesita de un cableado extra que, si se oculta, supone un incremento sustancial en el coste (bastante más que si los cableados, eléctrico y de bus se trazaran a la vez). Si se opta por la utilización de dispositivos de radiofrecuencia estos, evidentemente, son de un coste mayor que los aparatos normales.

X-10

X-10 es un protocolo de comunicación que permite controlar aparatos eléctricos a través de la instalación de una red eléctrica. El estándar surgió hace veinte años como parte de los experimentos realizados por la empresa Picosystem y lleva más de quince funcionando a nivel comercial.

Sus principales ventajas son:

- Protocolo altamente extendido: X-10 es un protocolo que está muy presente en el mercado mundial, sobre todo en Norteamérica y Europa.
- Simple de instalar y configurar: se conectan los módulos en los aparatos a controlar y se les asignan direcciones y a continuación se les pueden enviar órdenes básicas a través de un ordenador, mando a distancia o cualquier otro dispositivo de control remoto compatible con X-10.
- Económico: dentro de los sistemas domóticos es el que mejor precio presenta

debido al hecho de ser un sistema sencillo y preparado para instalación no profesional.

Sin embargo, también presenta considerables desventajas:

- Falta de versatilidad: el sistema X-10 no dispone de funciones lógicas programables que permitan realizar funciones complejas. La mayoría son funciones de control del tipo encener/apagar o regular la intensidad.
- Fiabilidad del sistema: la propia génesis del sistema que comparte la comunicación y la alimentación de los aparatos en la misma onda, tiene como principal consecuencia que la calidad de la señal dependa siempre de la calidad de la señal de red que llega a las viviendas, comprometiendo la fiabilidad del sistema.

Tras analizar brevemente los principales protocolos de comunicacion empleados en domótica, se llega a la conclusión de que si lo que queremos es domotizar un edificio que ya está construido, lo más cómodo sería decantarse por el protocolo X-10, ya que no requiere una instalación de cable adicional, mientras que si quisiéramos instalar el protocolo KNX o LONWorks, deberíamos hacer obra en la vivienda o dejar los cables al descubierto (lo que no quedaría muy estético).

Además, se ha optado por esta alternativa por que se adapta perfectamente a los requerimientos explicados en la sección 1.3: es económico y sencillo de utilizar y configurar. La falta de versatibilidad se va a cubrir con el desarrollo de un software que permita la definición de escenarios.

4.2. El protocolo X-10

La tecnología X-10, basada en corrientes portadoras, fue desarrollada entre 1976 y 1978 por Pico Electronics, en Glenrother, Escocia. X-10 surgió de una familia de chips denominada los proyectos X (o series X). El nombre X-10 se debe al hecho de que era el décimo proyecto de la compañía. Esta empresa comenzó a desarrollar este proyecto con la idea de obtener un circuito que pudiera ser insertado en un sistema mayor y controlado remotamente.

Su principal objetivo era transmitir datos por las líneas de baja tensión (220 Voltios o 110 Voltios) a muy baja velocidad (60 bytes por segundo en EEUU y 50 bytes por segundo en Europa) y muy bajo coste. Este es el punto clave de esta tecnología y su gran ventaja en comparación con otros protocolos de domótica. Al utilizar la red eléctrica existente en las casas la instalación resulta sencilla, cómoda y económica ya que no se precisa realizar ninguna obra ni cableado adicional.

Sin embargo, paradójicamente, este también es su gran inconveniente debido a que el sistema es muy sensible a los ruidos eléctricos lo que compromete su fiabilidad. Pese a ello, su bajo coste, facilidad de uso y variedad de dispositivos han hecho de X-10 el protocolo de domótica más extendido en el mundo.

Sin embargo, esta misma característica es la que provoca que el sistema sea muy sensible a los ruidos eléctricos, lo cual, como se ha indicado en la sección 4.1, compromete la fiabilidad del sistema. Pese a ello, su bajo coste, facilidad de uso y variedad de dispositivos han hecho de X-10 el protocolo de domótica más extendido en el mundo.

El protocolo X-10 no es propietario, cualquier fabricante puede producir dispositivos X-10 y ofrecerlos en su catálogo, sin embargo está obligado a usar los circuitos del fabricante escocés. Aunque, al contrario de lo que sucede con la firma Echelon y su Neuron Chip que implementa LonWorks, los circuitos integrados que implementan el X-10 tienen un royalty muy bajo (casi simbólico).

Se trata de un sistema modular y fácilmente ampliable. Es el propio usuario el que, en función de sus necesidades, irá determinando qué elementos del hogar necesita controlar pudiéndolos adquirir de manera progresiva. Utiliza una arquitectura descentralizada, que no requiere ningún elemento central para funcionar. Un sistema X-10 puede ser un único conjunto de dispositivos que están controlados directamente por el usuario.

X-10 implementó un sistema simple de direccionamiento que utiliza 16 códigos de casa (usando las letras de la A a la P) y 16 códigos de dispositivo (1 a 16), para poder emitir sin ambigüedad 256 (16x16) dispositivos. Esto no implica que solo puedan ser 256 los dispositivos utilizados en una red X-10, ya que puede haber varios módulos que utilicen la misma dirección. Es ese caso, la señal enviada a esa dirección será leída y ejecutada por todos los módulos configurados en esa dirección.

4.3. Teoría de la transmisión de las señales X-10

La transmisión de la información se realiza modulando pulsos de 120 kilohercios (kHz) que se superponen a la señal de la red. Esta transmisión está sincronizada con los pasos por cero de la corriente eléctrica con un retraso máximo de 200 microsegundos. Un uno binario está representado como un pulso de 120 kHz durante un milisegundo, y un cero como la ausencia de ese pulso.

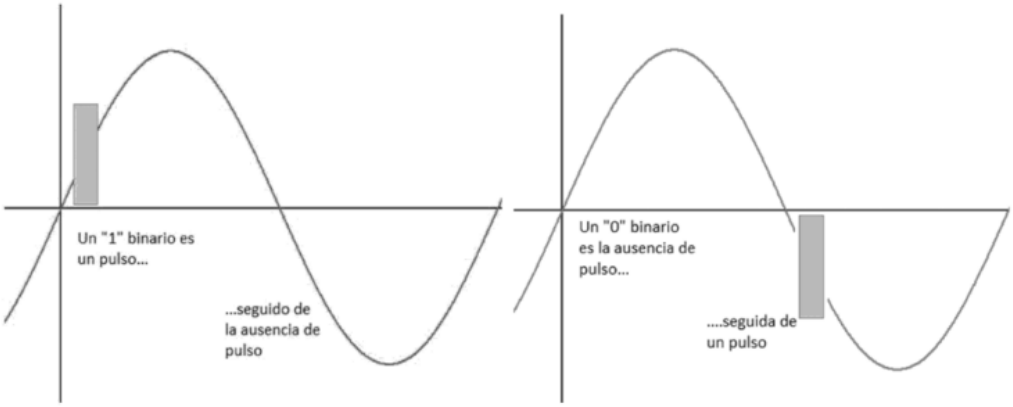


Figura 6: Representación de un 1 y un 0 binarios

El hecho de que los transmisores estén sincronizados con el paso por cero de

la tensión de red tiene dos objetivos: el primero, sincronizar a los transmisores y receptores, ya que la única conexión física que existe entre ellos es la línea de red; el segundo, reducir todo lo posible el ruido presente en la línea eléctrica ya que el nivel mínimo de interferencias producidas por otros equipos electrónicos se produce cuando al señal de red pasa por cero.

Precisamente, por ser los medios de distribución de energía eléctrica tan ruidosos, se llegó al acuerdo de que el envío de un bit nunca debe hacerse de forma aislada, sino que siempre debe enviarse junto con su complementario. El objetivo es reducir al mínimo la probabilidad de que el ruido eléctrico se pueda confundir con una señal válida. Sin embargo, tiene la desventaja de la reducción de la tasa de transmisión que así se limita a unos escasos 50 puntos básicos (el bit se envía por cada ciclo de la red eléctrica).

La transmisión completa de un código X-10 necesita 11 ciclos de corriente. Los dos primeros ciclos son para el código de inicio de mensaje (start code), 1110. Cabe señalar que esta secuencia es exactamente la indicada y no se aplica la regla de que cada bit debe ir acompañado por su complementario. Los cuatro siguientes ciclos corresponden al código de casa, y los cinco siguientes al código de unidad o de función.

A fin de distinguir este último campo se envía un último bit (y su complementario correspondiente), que identifica si el campo se refiere al número de un dispositivo (el último bit se corresponde con el 0 lógico) o al código de la función (el último bit se corresponde con el 1 lógico).

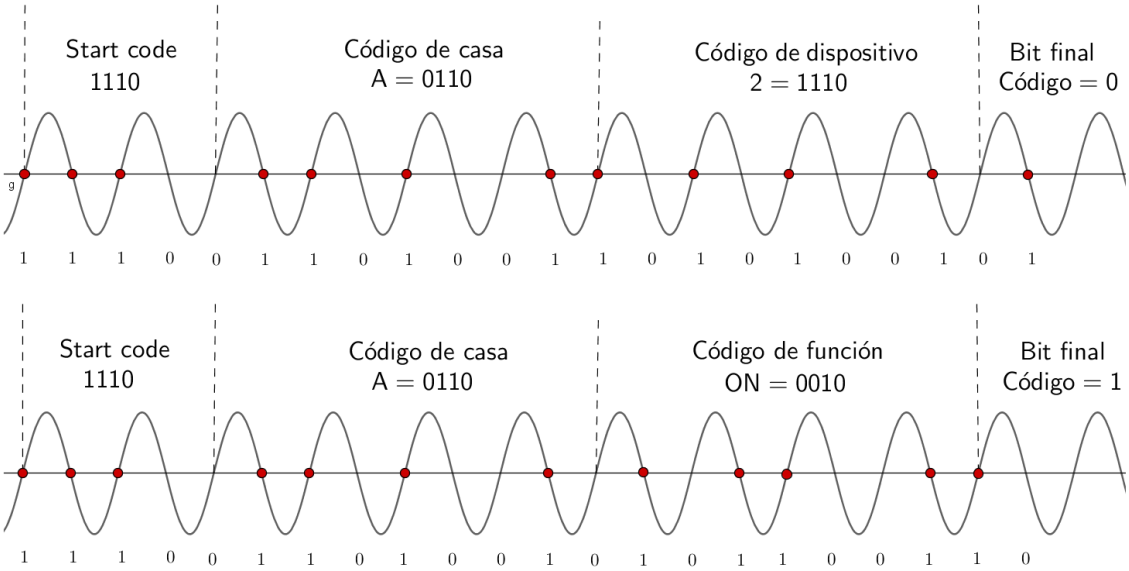


Figura 7: Onda de la señal eléctrica

Este bloque completo es transmitido dos veces, la primera para indicar el dispositivo y la segunda la función que se debe ejecutar, separadas cada una por tres ciclos de corriente. Los comandos *Dim* y *Bright* son excepciones a esta regla y deben ser continuamente transmitidos sin ninguna pausa de ciclo entre ellos.

Un código (o comando) X-10, por lo general, incluye dos acciones: activar un

dispositivo en particular (mensaje que indica el código del dispositivo), y luego enviar la función a ejecutar (mensaje con el código de la función). Después de activar un dispositivo este se mantendrá activo hasta que se indique lo contrario.

4.4. Lista de comandos X-10

La tabla 2 describe todos los comandos soportados por el protocolo X-10. Los seis primeros son los comandos básicos utilizados con mayor frecuencia. Cabe señalar que el uso de las funciones *Bright* y *Dim* no se limitan al ajuste de la intensidad de la luz sino que también pueden ser utilizados para otras funciones, tales como el balanceo de las persianas o el control de la calefacción.

Las funciones *Hail Request* y *Hail Acknowledge* se utilizan para determinar si es posible comunicarse con una casa cercana. Si se produce esta situación, es necesario utilizar un código diferente o un filtro de casa, que impida que las señales X-10 circulen a otras casas. Las funciones *Extended Code* y *Extended Data* se introducen en el X-10 con el fin de enviar más comandos o datos adicionales.

Código	Función	Descripción
0 0 0 0	*All Units Off*	Apaga todos los dispositivos con el código de la casa indicado en el mensaje
0 0 0 1	*All Lights On*	Activa todos los dispositivos de iluminación (con capacidad para controlar el brillo)
0 0 1 0	On	Enciende el dispositivo
0 0 1 1	Off	Apaga el dispositivo
0 1 0 0	Dim	Atenuar la intensidad
0 1 0 1	*Bright*	Aumenta la intensidad
0 1 1 0	*All Lights Off*	Apagar todas las luces
0 1 1 1	*Extended Code*	Código extendido
1 0 0 0	*Hail Request*	Pide respuesta a los dispositivos con el código de la casa indicado en el mensaje
1 0 0 1	*Hail Acknowledge*	Respuesta a la orden anterior
1 0 1 X	Pré-Set Dim	Permite la selección de dos niveles predefinidos de intensidad de la luz
1 1 0 0	*Extended Data Transfer*	Datos adicionales (seguido por 8 bytes)
1 1 0 1	*Satus is On*	Respuesta a la solicitud *Status Request*, indicando que el dispositivo está encendido
1 1 1 0	*Status is Off*	Respuesta a la solicitud *Status Request*, indicando que el dispositivo está apagado
1 1 1 1	*Status Request*	Solicitud sobre el estado del dispositivo

Tabla 2: Comandos X-10

4.5. Módulos X-10

En el libro *Domótica para ingenieros* [22] los módulos X-10 se clasifican en cinco grupos según la función que cumplen dentro del sistema domótico.

Controladores

Son capaces de emitir órdenes X-10 sobre el cableado eléctrico. Los más sencillos son simples mandos a distancia que se conectan directamente a la red eléctrica o que usan radio frecuencia. Los más sofisticados pueden ser conectados a un ordenador personal (PC) y programados con el *software* adecuado.

Módulos de aplicación

También llamados módulos actuadores. Son dispositivos conectados a la red eléctrica y al elemento que queremos controlar. Son capaces de leer el código X-10 presente en la red eléctrica, comprobar que va dirigido a él y ejecutar la orden correspondiente.

Poseen dos pequeños conmutadores giratorios que permiten asignarles una dirección de las 256 posibles. En uno de ellos se selecciona el código de la casa y en el otro el código del dispositivo.

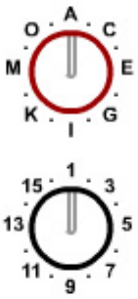


Figura 8: Asignación de dirección a un dispositivo

Dentro de los módulos actuadores se pueden distinguir dos tipos de dispositivos:

1. Módulos de aparato: responden a los comandos *On, Off* y *All Units Off*.
2. Módulos de lámpara (o *dimmers*): cuya función es regular la luz en intensidad. Responden a los comandos anteriores y además a *Bright, Dim, All Lights On* y *All Lights Off*.

Sensores

Recogen información de diferentes magnitudes físicas y, en caso de que estas sobrepasen cierto umbral, envían órdenes al módulo que se le haya programado.

Transceptores

Reciben una señal de radio frecuencia que es convertida en una señal eléctrica y se envía luego a través del cableado.

Repetidores y filtros

La misión de estos dispositivos es reducir los problemas derivados del uso de la red eléctrica como medio de transmisión. Los repetidores se encargan de regenerar y retransmitir las señales X-10 que reciben, por ello se ubican en aquellos puntos de la red en los que la señal tiene poca calidad. Por el contrario, los filtros tienen la funcionalidad opuesta, evitar la propagación de las señales X-10 a lo largo de la red eléctrica, lo cual es conveniente para evitar que haya interferencias con instalaciones X-10 vecinas.

Módulos sistema

Dispositivos que garantizan el correcto funcionamiento de la instalación (filtros, acopladores, equipos de medida).

5. Análisis del sistema

5.1. Definición de conceptos

Antes de definir la lógica y el diseño que va a tener la aplicación encargada de controlar y gestionar el sistema domótico hay que definir claramente algunos conceptos que se han ido mencionando a lo largo del proyecto: los conceptos de **dispositivo**, **sensor**, **escenario** y **evento**.

5.1.1. Definición de dispositivo

Un dispositivo está formado por un conjunto de hardware y software. La parte hardware la forma el módulo X-10 presente físicamente en el sistema (un módulo de lámpara, un sensor de movimiento, un módulo de persiana, etc.) mientras que la parte software la forman el conjunto de propiedades que el usuario defina sobre dicho módulo. Estas propiedades pueden ser:

- Una descripción o nombre del módulo.
- El tipo (módulos de aparato o de lámpara).
- En qué estado se encuentra (encendido, apagado, nivel de intensidad).

A lo largo del proyecto, cuando se habla de actualizar el estado de un dispositivo o de actualizar el valor de su intensidad se quiere hacer referencia a un hecho que tiene dos implicaciones directas:

1. El módulo físico presente en la instalación domótica se enciende o apaga en función del nuevo estado o, si es el caso, actualiza su intensidad.
2. El sistema actualiza el nuevo estado o brillo para dicho dispositivo en su almacenamiento persistente.

5.1.2. Definición de sensor

Se distinguen dos tipos de sensores:

- **Sensores físicos:** son dispositivos de tipo sensor, es decir, los módulos X-10 de tipo sensor que haya instalados físicamente en la vivienda junto con el conjunto de propiedades que el usuario defina sobre dicho módulo (y que son las mismas que para el caso de los dispositivos).
- **Sensores virtuales:** son sensores cuya presencia se simula consultando a algún servicio web de meteorología los valores que deberían ofrecer. Por ejemplo, se podría tener un sensor virtual llamado «Temperatura» o «Lluvia» que se encargara de averiguar estos datos para la zona geográfica en la que se encuentra la instalación.

5.1.3. Definición de escenario y evento

Un escenario no es más que un conjunto de dispositivos y un estado que los mismos deben adoptar cuando dicho escenario se active.

Por ejemplo, podríamos crear un escenario llamado «Modo película» que al activarse bajara las persianas del salón al 20 % y encendiera el DVD y la televisón. O podríamos crear otro llamado «Invierno» que se encargara de encender la calefacción.

5.1.4. Definición de evento

Al introducir el concepto de escenario, aparece la necesidad de poder introducir condiciones en el sistema, que al cumplirse realicen acciones como activar un escenario. Resultaría de interés que, por ejemplo, se pudiera indicar que de nueve de la noche a nueve de la mañana, si la temperatura es inferior a 18 grados, se active el escenario «Invierno» encargado de activar la calefación. Esto es lo que se ha llamado *evento*.

Un evento no es más que un conjunto de condiciones que deben cumplirse y las acciones que deben llevarse a cabo en caso de que aquellas se cumplan. Estas condiciones pueden ser tanto ambientales como de tiempo.

5.2. Casos de uso

Como se ha explicado en la sección 1.3 el objetivo del proyecto será desarrollar un sistema de automatización para el hogar que permita la gestión, control y programación del mismo. Se pretende que dicho sistema sea abierto, ampliable y personalizable, para lo cual se debe permitir la definición de escenarios y eventos de manera sencilla desde el teléfono móvil.

Por último, el sistema hará uso de una serie de sensores virtuales que se encargarán de proporcionar datos climáticos consultando a algún servicio disponible en la web.

Mediante el diagrama de casos de uso se van a mostrar de una forma gráfica los requisitos funcionales de todo el sistema y cómo el usuario de la aplicación interactúa con el mismo.

5.2.1. Casos de uso en relación con los dispositivos

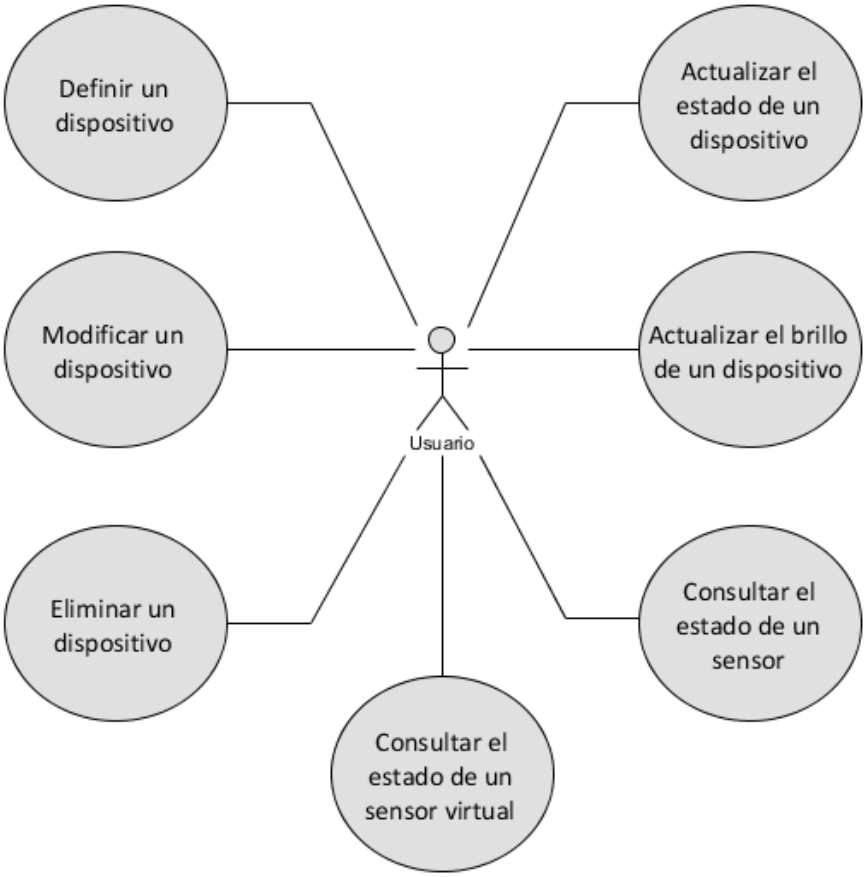


Figura 9: Casos de uso relacionados con el dispositivo

Caso de uso	Definir un dispositivo
Resumen	El actor debe indicar el código de casa, el código de unidad, el tipo de dispositivo y una breve descripción. El sistema comprueba que no exista un dispositivo con los mismos códigos de casa y unidad y en ese caso lo crea.
Actores	Usuario
Precondición	-
Postcondición	El dispositivo se guarda de manera persistente en el sistema
Propósito	Crear un nuevo dispositivo

Tabla 3: Definir un dispositivo

Caso de uso	Modificar un dispositivo
Resumen	El actor puede modificar tanto la descripción como el tipo de dispositivo.
Actores	Usuario
Precondición	El dispositivo debe haber sido creado con anterioridad
Postcondición	El dispositivo se almacena de manera persistente en el sistema
Propósito	Modificar un dispositivo existente en el sistema

Tabla 4: Modificar un dispositivo

Caso de uso	Eliminar un dispositivo
Resumen	Permite al usuario eliminar dispositivos almacenadas en el sistema
Actores	Usuario
Precondición	El actor debe haber seleccionado uno de los dispositivos almacenados en el sistema
Postcondición	El dispositivo se elimina totalmente del sistema
Propósito	Eliminar un dispositivo

Tabla 5: Eliminar un dispositivo

Caso de uso	Cambiar el estado de un dispositivo
Resumen	El actor selecciona un dispositivo e indica que quiere cambiar su estado (encendido/apagado). El sistema almacena el nuevo estado y el módulo en cuestión actualiza su estado.
Actores	Usuario
Precondición	El dispositivo debe haber sido creado con anterioridad
Postcondición	El estado del dispositivo ha cambiado
Propósito	Cambiar el estado de un dispositivo

Tabla 6: Cambiar el estado de un dispositivo

Caso de uso	Cambiar el brillo de un dispositivo
Resumen	El actor selecciona un dispositivo e indica el nuevo valor para el brillo. El sistema almacena el valor y el módulo en cuestión actualiza su brillo.
Actores	Usuario
Precondición	El dispositivo debe haber sido creada con anterioridad
Postcondición	El brillo del dispositivo ha cambiado
Propósito	Cambiar el brillo de un dispositivo

Tabla 7: Cambiar el brillo de un dispositivo

Caso de uso	Consultar el estado de un sensor
Resumen	El actor selecciona un dispositivo de tipo sensor. El sistema localiza el dispositivo seleccionado y muestra sus datos en pantalla.
Actores	Usuario
Precondición	El sensor (dispositivo de tipo sensor) debe haber sido creado con anterioridad
Postcondición	-
Propósito	Consultar el estado de un sensor

Tabla 8: Consultar el estado de un sensor

Caso de uso	Consultar el estado de un sensor virtual
Resumen	El actor selecciona un dispositivo de tipo sensor virtual. El sistema realiza la petición necesaria al servicio web elegido y muestra los datos en pantalla.
Actores	Usuario
Precondición	-
Postcondición	-
Propósito	Consultar el estado de un sensor virtual

Tabla 9: Consultar el estado de un sensor virtual

5.2.2. Casos de uso en relación con los escenarios

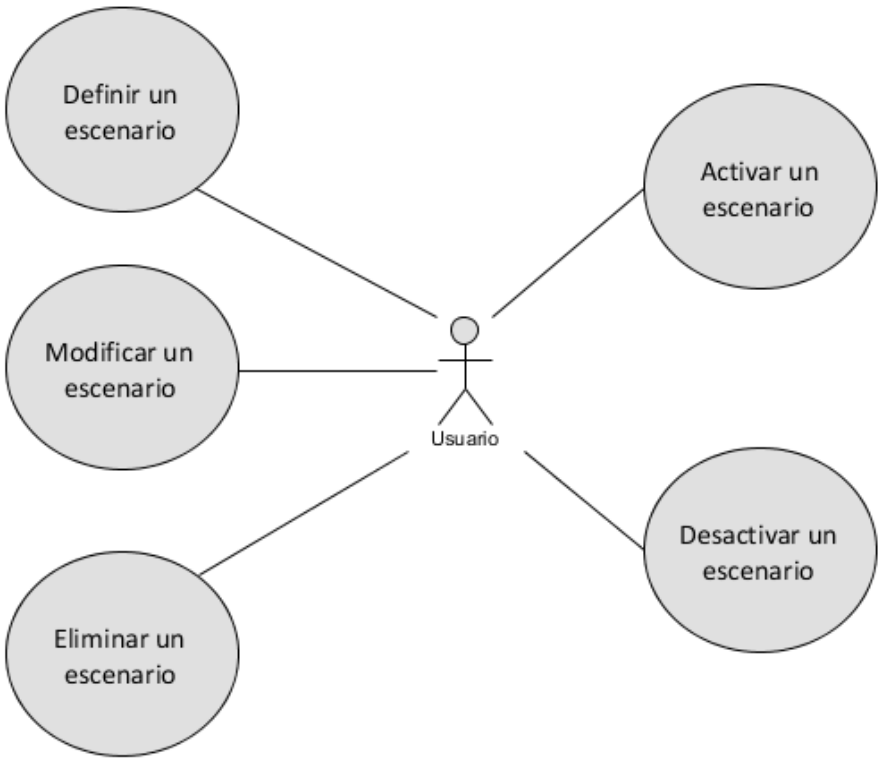


Figura 10: Casos de uso relacionados con el escenario

Caso de uso	Definir un escenario
Resumen	El actor debe indicar una breve descripción del mismo y una lista de dispositivos y los estados que deben adoptar dichos dispositivos cuando se active el escenario. A continuación el sistema crea el nuevo escenario.
Actores	Usuario
Precondición	-
Postcondición	El escenario se almacena de manera persistente en el sistema
Propósito	Definir un escenario

Tabla 10: Definir un escenario

Caso de uso	Modificar un escenario
Resumen	El actor puede modificar tanto la descripción del escenario como la lista de dispositivos y los estados que deben adoptar dichos dispositivos cuando se active el escenario. A continuación el sistema actualiza el escenario.
Actores	Usuario
Precondición	-
Postcondición	El escenario se actualiza de manera persistente en el sistema
Propósito	Modificar un escenario

Tabla 11: Modificar un escenario

Caso de uso	Eliminar un escenario
Resumen	Permite al usuario eliminar un escenario almacenado en el sistema.
Actores	Usuario
Precondición	El actor debe haber seleccionado uno de los escenarios almacenados en el sistema
Postcondición	El escenario se actualiza de manera persistente en el sistema
Propósito	Eliminar un escenario

Tabla 12: Eliminar un escenario

Caso de uso	Activar un escenario
Resumen	Permite al usuario activar un escenario almacenado en el sistema. El sistema localiza el escenario y actualiza el estado de cada uno de los dispositivos del mismo.
Actores	Usuario
Precondición	El actor debe haber seleccionado uno de los escenarios almacenados en el sistema
Postcondición	El escenario se actualiza de manera persistente en el sistema
Propósito	Activar un escenario

Tabla 13: Activar un escenario

5.2.3. Casos de uso en relación con los eventos

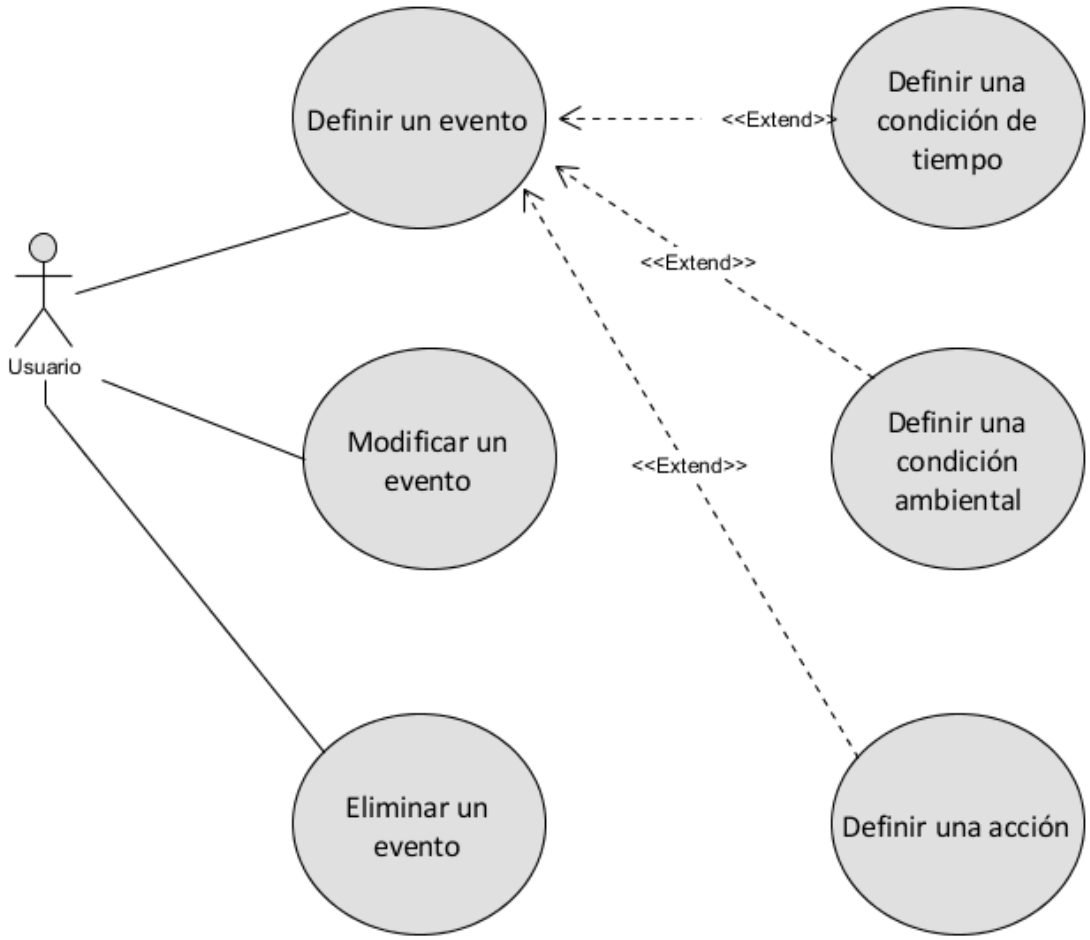


Figura 11: Casos de uso relacionados con el evento

Caso de uso	Definir un evento
Resumen	El actor debe indicar una lista de condiciones y una lista de las acciones a realizar cuando dichas condiciones se cumplan. Las condiciones pueden ser ambientales o temporales. Se debe definir al menos una condición y una acción para poder crear un evento. A continuación el sistema crea el nuevo evento.
Actores	Usuario
Precondición	-
Postcondición	El evento se almacena de manera persistente en el sistema
Propósito	Definir un evento

Tabla 14: Definir un evento

Caso de uso	Modificar un escenario
Resumen	El actor puede modificar tanto la lista de condiciones como la lista de acciones. A continuación el sistema actualiza el evento.
Actores	Usuario
Precondición	-
Postcondición	El evento se actualiza de manera persistente en el sistema
Propósito	Modificar un evento

Tabla 15: Modificar un evento

Caso de uso	Eliminar un evento
Resumen	Permite al usuario eliminar un evento almacenado en el sistema.
Actores	Usuario
Precondición	El actor debe haber seleccionado uno de los evento almacenados en el sistema
Postcondición	El evento es eliminado del sistema
Propósito	Eliminar un escenario

Tabla 16: Eliminar un evento

6. Plan de trabajo. Problemas y soluciones.

Como se ha dicho en la sección 1.3, para desarrollar el software de automatización se va a partir de un proyecto de código abierto ya existente para la placa Arduino que se replanteará para adaptarlo a la plataforma Intel Galileo.

Se trata de una librería desarrollada por Thomas Mittet que, como él mismo explica en su blog [17], proporciona «una solución independiente de la plataforma que facilita la comunicación con el hardware X-10 y soporta el envío y recepción de mensajes X-10 tanto estándar como extendidos a través de la red eléctrica y la recepción a través de Ethernet, radiofrecuencia e infrarrojos». Esta librería proporciona un conjunto muy completo de funciones que permiten al desarrollador mandar órdenes X-10 en el formato correcto y de una forma sencilla.

En realidad, esta formada por tres módulos más pequeños:

- La librería **X10ex** que proporcina soporte para mandar y recibir comandos X-10 a través del puerto serie.
- La librería **X10ir** que proporciona soporte para la recepción de comandos X-10 por infrarrojos.
- La librería **X10rf** que proporciona soporte para la recepción de comandos X-10 por radiofrecuencia.

Junto con la librería, Thomas Mittet incluye tres programas de ejemplo:

- **X10_Simple**: permite probar la librería X10ex.
- **X10_Remote**: permite probar las librerías X10rf y X10ir.
- **X10_Ethernet**: integra y permite probar toda la funcionalidad además de proporcionar soporte para servicios REST y JSON.

A continuación se detalla el plan de trabajo propuesto inicialmente y que ha guiado el desarrollo de este trabajo. Este planteamiento derivaba de la existencia de ciertas incertidumbres, como la de si sería factible portar un código desarrollado para un controlador Arduino a la plataforma Intel Galileo disponible. Aunque la plataforma era supuestamente compatible en cuanto a herramientas de desarrollo y conectividad con el entorno Arduino, poseía una arquitectura notablemete diferente. Durante el desarrollo del trabajo, pronto se confirmó que estas dudas estaban plenamente justificadas.

Paso 1: descargar la librería X-10 y poner en marcha el ejemplo X10_Simple en un Arduino estándar.

Paso 2: una vez conseguido, portarlo a Galileo.

Paso 3: si se consigue realizar el paso 2, poner en marcha el ejemplo X10_Ethernet y mejorar el control a través de Ethernet con Galileo. Si no se consigue, usar un módulo Ethernet para Arduino y hacer toda la programación sobre este microcontrolador.

Paso 4: buscar algún servicio web que proporcione datos meteorológicos en tiempo real e integrar su uso en el proyecto.

Paso 5: incluir la programación de escenarios.

Paso 6: desarrollar una aplicación en Android capaz de interactuar con el microcontrolador utilizado.

6.1. Problemas encontrados

La mayor parte de problemas que he tenido que enfrentar a lo largo de la realización del proyecto han estado realacionados con la portabilidad de los *sketches* de Arduino a Galileo. Ya que Galileo nació como fruto de la colaboración entre Intel y la fundación Arduino con el objetivo de ser compatible con las placas y librerías de Arduino, en primera instancia se puede pensar que usar los programas de Arduino en Galileo no debe ser una tarea difícil. Sin embargo, la realidad es diferente.

Lo primero de lo que se dará cuenta alguien que intente ponerse manos a la obra será con que no todas las librerías de Arduino han sido portadas correctamente a la arquitectura de Galileo. Un ejemplo muy evidente puede encontrarse en la clase `String`. Al usar objetos de la clase `String` en un *sketch* es probable que el compilador muestre errores del tipo «función no declarada» o «variable no declarada».

A continuación se presentan tres grandes problemas que se han encontrado a la hora de portar los programas de la librería X-10 de Arduino a Galileo.

6.1.1. Primer problema: x86 no es AVR

Lo primero que debemos tener en cuenta es que, por desgracia, la arquitectura x86 es muy diferente de la arquitectura AVR. Por tanto, los registros, los espacios de memoria, etc. son completamente diferentes. Aunque parezca obvio es algo que se debe tener muy presente a la hora de portar código de Arduino a Galileo.

Tras cargar el código del programa X10_Simple en el IDE de Arduino, seleccionar la placa Intel Galileo Gen2 y compilar, lo primero que nos muestra el compilador son algunos errores ligados a las diferencias de hardware entre Arduino y Galileo:

```
Arduino:1.6.7 (Windows 10), Placa:"Intel® Galileo Gen2"

ATENCIÓN: la librería X10 pretende ejecutarse sobre arquitectura(s)
[avr] y puede ser incompatible con tu actual tarjeta la cual corre
sobre arquitectura(s) [i586].
```

```
In file included from
E:\Arduino\libraries\X10\examples\X10_Simple\X10_Simple.ino:20:0:

E:\Arduino\libraries\X10\src/X10ex.h:24:24: fatal error:
avr/eeprom.h: No such file or directory

compilation terminated.
exit status 1
Error de compilación
```

En la librería escrita por Thomas Mittet tanto la detección del paso por cero de la señal de red como la recepción de datos procedentes de la misma se realizan mediante interrupciones. Estas están programadas a muy bajo nivel, haciendo uso directo de los registros propios del microcontrolador Atmel AVR. Para realizar esta programación a tan bajo nivel, Arduino proporciona la librería `<avr/pgmspace.h>`.

Sin embargo, Galileo no tiene la libreria `<avr/pgmspace.h>`, lo que es lógico, ya que no posee esa arquitectura. Esto supone un gran problema ya que la gran mayoría de librerías de terceros usan este archivo y sus definiciones, en especial para definir variables directamente en la memoria flash, debido a la capacidad limitada de memoria RAM (2 KB) de Arduino.

Por tanto, la solución que parece obvia, llegados a este punto, pasa por reescribir las partes de la librería X-10 que sean incompatibles con la arquitectura de Intel, accediendo a los registros equivalentes de su microcontrolador.

6.1.2. Segundo problema: el sistema operativo y el hardware

Una vez adoptada la decisión de reescribir la librería para adaptarla a la nueva arquitectura nos topamos con el siguiente problema: el acceso al hardware. Tras mucho investigar se llega a la conclusión de que los *sketches* en Galileo no pueden acceder directamente a los registros de su microcontrolador. Y es obvio que no puedan hacerlo.

Galileo usa un sistema x86 con un sistema operativo Linux completo. Esto significa que Intel Galileo ejecuta los *sketches* de Arduino como procesos de usuario Linux y estos no tienen permitido el acceso directo al hardware. En su lugar, el acceso debería hacerse mediante servicios apropiados de Linux o funciones de más alto nivel de las librerías de Arduino.

Por tanto, la solución no es tan trivial como realizar una tarea de búsqueda y reemplazo.

6.1.3. Tercer problema: el sistema operativo y las interrupciones

El tercer y último problema encontrado está relacionado con el anterior. El dispositivo encargado de inyectar las órdenes X-10 en la red eléctrica es el PLC (*Power*

Line Communications) Marmitek XM10, un dispositivo que utiliza el cableado de la vivienda para transmitir datos además de la corriente eléctrica. Las especificaciones técnicas del PLC, detalladas en su manual, indican que una vez detectado el paso por cero los datos deben enviarse antes de que hayan pasado 50 microsegundos [16].

La librería X-10 resuelve este problema con interrupciones. De esta manera, cada vez que se recibe la señal de activación del paso por cero, se lanza una interupción que se encarga de maneajar todo lo necesario para enviar los datos pertinentes al PLC.

Sin embargo, al ejecutar Galileo los *sketches* como procesos de usuario, todas las llamadas al kernel, interrupciones del sistema operativo, hilos, etc. tienen una prioridad más alta, lo que significa que pueden interrumpir la ejecución del *sketch* en cualquier momento, comprometiendo la velocidad de ejecución. En Arduino, al no usarse sistema operativo, los *sketches* se ejecutan a la mayor velocidad posible sin interrupciones de ningún tipo. Es decir, el comportamiento en tiempo real es «verdadero» tiempo real.

Consultando en la comunidad de usuarios de Intel las experiencias y datos obtenidos por otros desarrolladores, nos encontramos con que empezar a manejar una interrupción de este tipo puede costar entre 8 y 11 milisegundos. Por tanto, no se cumplen los requisitos del PLC.

Por esta razón, aunque Intel Galileo es una máquina muy rápida, sobretodo en comparación con las placas Arduino, utilizarla para ejecutar *sketches* que requieran respuestas en tiempo real a los pines de entrada y salida puede no ser una gran idea.

6.2. Soluciones propuestas

6.2.1. Primera solución propuesta: Arduino

Ante todas las dificultades encontradas para poder usar la placa Galileo, se tomó la decisión de seguir desarrollando el proyecto sobre la placa Arduino, dejando de lado a Galileo. La idea era la siguiente: partiendo del programa de ejemplo X10_Ethernet, incorporar la nueva funcionalidad. Para ello era necesario:

1. Usar un módulo Ethernet para poder crear un pequeño servidor que atendiera las peticiones de la aplicación Android.
2. Utilizar una tarjeta micro-SD para almacenar todos los datos que va a manejar la aplicación (escenarios y otra información necesaria para hacer funcionar el sistema).
3. Usar un reloj de tiempo real o RTC (*Real Time Clock*) para poder mantener la fecha y la hora aunque se produzca algún fallo como, por ejemplo, un corte en la corriente eléctrica.
4. Obtener datos ambientales de algún servicio web.

Al compilarlo el compilador nos da el siguiente mensaje:

```
El Sketch usa 26.136 bytes (81%) del espacio de almacenamiento de
programa. El máximo es 32.256 bytes.

Las variables Globales usan 1.232 bytes (60%) de la memoria
dinámica, dejando 816 bytes para las variables locales. El máximo es
2.048 bytes.
```

Y simplemente añadiendo la librería SD (sin más código):

```
El Sketch usa 30.130 bytes (93%) del espacio de almacenamiento de
programa. El máximo es 32.256 bytes.

Las variables Globales usan 1.847 bytes (90%) de la memoria
dinámica, dejando 201 bytes para las variables locales. El máximo es
2.048 bytes.

Poca memoria disponible, se pueden producir problemas de
estabilidad.
```

Ya nos podemos imaginar qué ocurre si, además, incluimos la librería encargada de manejar el RTC. Pero incluso aunque se prescindiera del reloj RTC y se optara por usar algún servicio web (de manera que cada vez que se reiniciara la placa se pidiera la fecha y hora actuales) los requerimiento de memoria del resto del proyecto seguirían siendo demasiado grandes para lo que Arduino puede ofrecer.

Todo el código necesario para analizar los ficheros de la SD y poder extraer los datos de cada escenario, desbordaría de inmediato tanto la memoria flash como la dinámica.

En definitiva, Arduino es una buena opción para realizar proyectos pequeños, sin grandes necesidades de memoria, pero si lo que se persigue es desarrollar un proyecto más complejo y completo no resulta ser la alternativa más adecuada.

6.2.2. La solución definitiva: lo mejor de ambos mundos

Llegados a este punto, y tras mucho investigar, probar y volver a investigar, se llega a la conclusión de que cada placa tiene algo que aportar al proyecto. Por una parte, Arduino es la mejor alternativa para manejar las interrupciones. Por la otra, Galileo ofrece la capacidad y recursos necesarios para gestionar y realizar todo «el trabajo pesado».

Así que, ¿por qué no aprevechar los puntos fuertes de ambas placas? Por separado, cada una cojea por uno u otro lado, mientras que juntas proporcionan todo lo necesario.

Además, nos encontramos desarrollando un proyecto que pertenece al mundo del Internet de las Cosas, y como se ha explicado en la sección 3.2, la imagen Linux IoTDevkit de Intel proporciona un montón de características que hacen de Galileo una alternativa muy adecuada para desarrollar software relacionado con el IoT:

1. No necesita módulo Ethernet adicional (lo lleva integrado).
2. No habrá que preocuparse por mantener el reloj en hora. El sistema operativo Linux se encarga él solo, mediante NTP (*Network Time Protocol*), de mantener la fecha y hora sincronizadas.
3. No habrá que preocuparse por los problemas de espacio, pues ofrece 256 MBytes de memoria dinámica y 8 KBytes de memoria EEPROM, más que suficiente para realizar este proyecto.
4. Además ofrece la posibilidad de trabajar con NodeJS, que como se ha visto en la sección 3.5 resulta una alternativa muy interesante para programa el IoT.

7. Diseño del hardware

Como ya se ha introducido en la sección 6.1.3, el encargado de inyectar las órdenes X-10 en la red eléctrica será el PLC. Para realizar esta tarea el PLC separa la información digital de la señal eléctrica de forma similar a como las líneas ADSL separan la señal de voz de la de datos, es decir, con un filtrado en frecuencia.

Por una parte tenemos la corriente eléctrica que viaja a baja frecuencia (entre 50 Hz y 60 Hz) y a relativamente alto voltaje (entre los 110 y los 220 voltios). Por otra, tenemos la señal de datos que se encuentra en una frecuencia superior (decenas de MHz) y con un voltaje muy inferior, que es separada mediante un filtro incluido en los adaptadores PLC.

Para comunicarnos con el PLC utilizaremos la placa Arduino. Esta se encargará de recibir comandos X-10 desde la placa Galileo (en un formato legible por los humanos), transformarlos en órdenes X-10 con el formato descrito en la sección 4.3 y transmitírselos en el momento adecuado al PLC. Por tanto, la interfaz de red del sistema de automatización estará formada por la unión de ambos dipositivos: el PLC y Arduino.

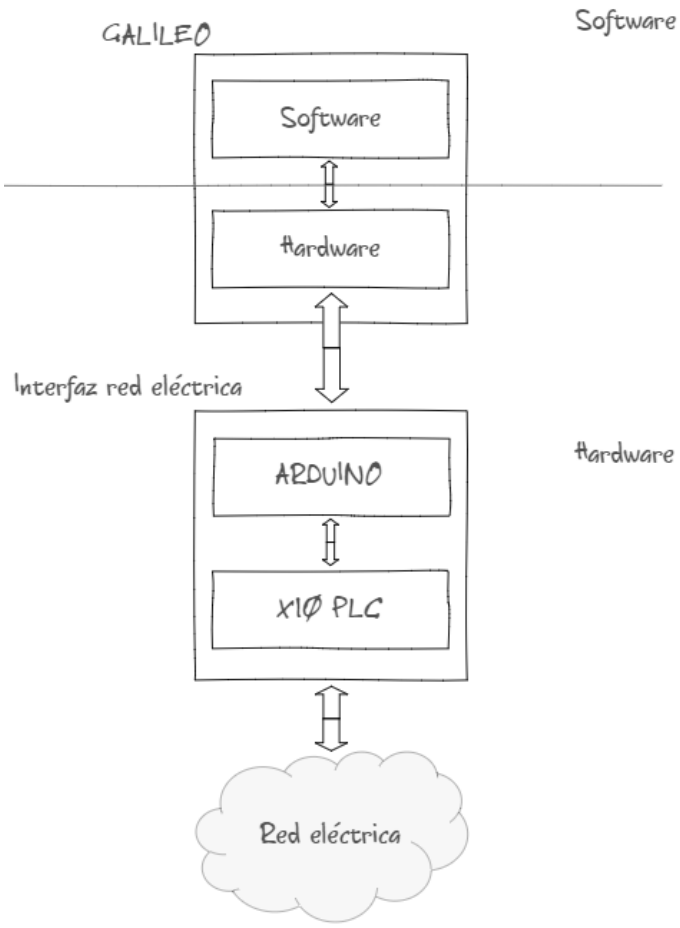


Figura 12: Arquitectura hardware del sistema domótico

Cuando el PLC detecta que se está produciendo un paso por cero activa una señal que es recibida por Arduino en el pin 4 a través del hilo etiquetado como «Zero

Cross» en la figura 14, provocando una interrupción. Si en ese momento Arduino tiene datos que proporcionar, los envía a través del pin 5 por el hilo etiquetado como «transmisión». El PLC los recibe y los inyecta en la red eléctrica.

Del mismo modo, cuando el PLC recibe información procedente de la red eléctrica, la recoge y la manda por el hilo etiquetado como «recepción». La librería X-10 se encarga de permanecer a la escucha de los datos que puedan llegar por este hilo y recogerlos a través del pin 6. De esta manera, Arduino puede recibir las órdenes que otros módulos X-10 mandan a través de la red eléctrica.

En la siguiente imagen se muestra el aspecto que tiene el PLC que se va a utilizar: En este caso concreto, el cable rojo es el de toma de tierra (GND), el amarillo el que

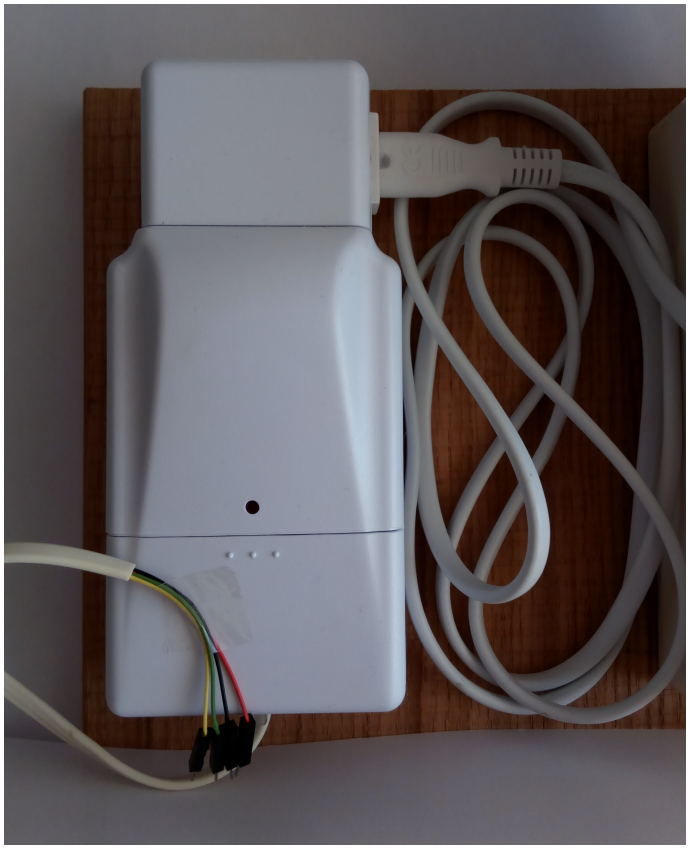


Figura 13: PLC Marmitek XM10

se encarga de transmitir la señal de activación del paso por cero, el verde el que se encarga de transmitir los datos procedentes de la red eléctrica y el amarillo el que se encarga de recibir los datos procedentes de la placa Arduino.

7.1. Conexiones

A continuación se presenta un esquema con las conexiones que deben realizarse para configurar correctamente el sistema.

Arduino y Galileo se comunicarán entre sí a través de sus respectivos puertos serie. Los hilos amarillo y verde de la figura 14 son los encargados de establecer esta

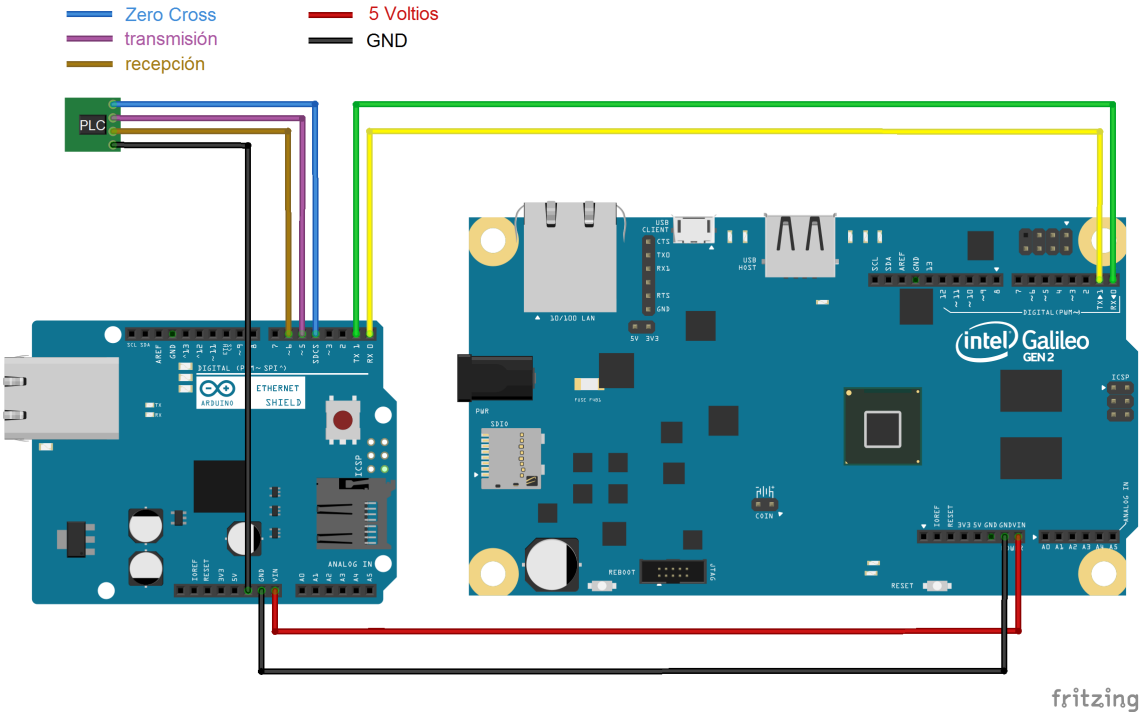


Figura 14: Conexiones

conexión bidireccional. Cuando la Galileo necesite emitir algún comando X-10 se lo mandará a Arduino a través de su puerto serie y este se encargará de mandárselo al PLC como se ha descrito en el párrafo anterior. Por su parte, Arduino se encargará de retransmitir a Galileo los datos que le lleguen desde el PLC.

Para alimentar el circuito se va a utilizar una fuente de alimentación de 12V conectada a la placa Intel Galileo, que será la encargada de proporcionar alimentación a Arduino a través de su pin VIn.

8. Diseño del software

8.1. Diseño de la lógica de la aplicación

Para diseñar la lógica que va a tener el sistema de automatización se deben tener claros los conceptos de **dispositivo**, **sensor**, **escenario** y **evento** definidos en la sección 5.1.

8.1.1. Diagrama de clases

Un diagrama de clases representa los objetos fundamentales del sistema, es decir, los que acaba percibiendo el usuario y con los que espera interactuar para tratar de obtener el resultado deseado. Reflejan la estructura estática del sistema.

El siguiente diagrama describe la estructura de nuestro sistema domótico. Este estará formado por las clases `Dispositivo`, `Sensor_virtual`, `Escenario`, `Evento`, `Condicion_tiempo` y `Condicion_ambiental`.

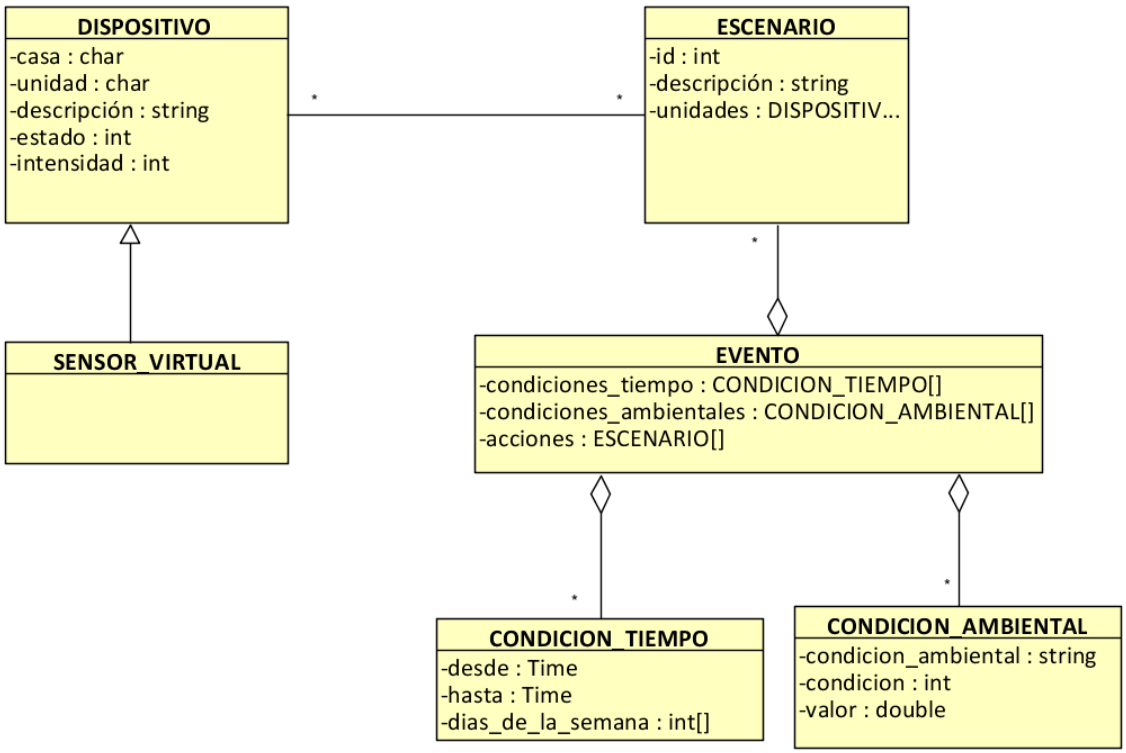


Figura 15: Diagrama de clases

La clase `Evento` se emplea para modelar situaciones que queremos identificar para, en consecuencia, realizar una o varias acciones. Está formada por una serie de condiciones y las acciones que deben llevarse a cabo cuando dichas condiciones se cumplen. Estas condiciones pueden ser de dos tipos: ambientales o temporales.

La clase `Condicion_ambiental` se emplea para representar una condición ambiental. Está formada por tres atributos:

- `condicion_ambiental`: que puede tomar los valores temperatura, humedad, presión, viento, lluvia o nieve.
- `condicion`: que puede tomar los valores menor que, menor o igual que, igual que, mayor o igual que o mayor que.
- `valor`: valor sobre el cual se evalúa la condición.

Un ejemplo de condición ambiental podría ser: temperatura mayor que 25.

La clase `Condicion_temporal` se emplea para representar una condición de tiempo. Está formada por tres atributos:

- `desde`: representa la hora a partir de la cual la condición se evaluará a `cierto`.
- `hasta`: representa la hora a partir de la cual la condición se evaluará de nuevo a `falso`.
- `dias_de_la_semana`: es una lista de los días de la semana en los que debe activarse el escenario.

Un ejemplo de condición temporal podría ser: desde 22:00 hasta 07:00.

8.1.2. Consultando la información ambiental: OpenWeatherMap

El encargado de proporcionar información a los «sensores virtuales» va a ser OpenWeatherMap[7], un servicio web que ofrece datos meteorológicos en tiempo real desarrollado por un equipo especialista en Tecnologías de la Información (IT) con más de diez años de experiencia en el desarrollo de soluciones Big Data y telecomunicaciones.

Su objetivo estuvo inspirado por el trabajo de otros servicios web como OpenStreetMap[8] o Wikipedia[9], que hacen disponible para todo el mundo el acceso gratuito a la información.

OpenWeatherMap suministra alrededor de mil millones de previsiones por día, proporcionando datos meteorológicos a mas de 200.000 ciudades y cualquier ubicación geográfica. Proporciona una amplia gama de datos meteorológicos incluyendo clima en tiempo real (gracias a las más de 40.000 estaciones meteorológicas que tiene a su disposición), pronósticos, precipitaciones, viento, nubes, gran variedad de mapas y análisis.

Lo más importante es que ofrece una API simple y fácil de utilizar que proporciona sus datos en los formatos de uso más extendido: XML, JSON y HTML. Para poder usarla basta con registrarse en su página web, obtener una clave de API y empezar a realizar llamadas a su servidor. Con una licencia gratuita se pueden hacer

[7] http://openweathermap.org
[8] www.openstreetmap.org
[9] www.wikipedia.org

hasta sesenta peticiones cada hora, más que suficiente para cubrir la necesidad del proyecto.

Si, por ejemplo, se quieren obtener los datos meteorológicos actuales para la ciudad de Valencia, basta con realizar la siguiente llamada desde cualquier aplicación o navegador web:

`api.openweathermap.org/data/2.5/weather?units=metric&method=json&appid=API_key&q=Valencia,ES`

donde `API_key` debe ser sustituido por la clave API obtenida durante el registro. En el momento en el que se escribían estas líneas la respuesta obtenida para esta llamada fue:


```
{
  "coord": {
    "lon": -0.38,
    "lat": 39.47
  },
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01n"
    }
  ],
  "base": "cmc stations",
  "main": {
    "temp": 25.36,
    "pressure": 1072,
    "humidity": 55,
    "temp_min": 25.2,
    "temp_max": 25.56
  },
  "wind": {
    "speed": 3.76,
    "deg": 108.506
  },
  "rain": {},
  "snow": {},
  "clouds": {
    "all": 0
  },
  "dt": 1471549220,
  "sys": {
    "type": 3,
    "id": 255375,
    "message": 0.0105,
```

```
    "country": "ES",
    "sunrise": 1471497475,
    "sunset": 1471546294
  },
  "id": 2509954,
  "name": "Valencia",
  "cod": 200
}
```

De toda esta información, la que se ha considerado de interés para este proyecto es la siguiente: temperatura, presión, humedad, viento, lluvia y nieve.

8.2. Diseño de la persistencia

Para la persistencia de la información se ha optado por emplear ficheros JSON que se almacenarán en el sistema de archivos de Linux de la placa Intel Galileo. Habrá tres ficheros: `units.json`, `scenarios.json` y `events.json`

8.2.1. El fichero units.json

Contendrá una lista de dispositivos. Cada dispositivo estará representada por un objeto JSON con el siguiente formato:

- `url`: dirección relativa del dispositivo dentro del sistema domótico, formada por el código de casa y el código de unidad. Por ejemplo: '/A/2/'
- `type`: tipo de módulo X-10:
  0 - Desconocido
  1 - Dispositivo
  2 - Regulador
  3 - Sensor
- `name`: nombre o breve descripción del módulo X-10
- `state`: estado del dispositivo X-10
  0 - Desconocido
  2 - Encendido
  3 - Apagado
- `brightness`: intensidad del dispositivo X-10 en caso de que se trate de un dispositivo de tipo regulador.

8.2.2. El fichero scenarios.json

Contendrá una lista de escenarios. Cada escenario estará representado por un objeto JSON con el siguiente formato:

- `id`: identificador único del escenario
- `enabled`: booleano que indica el estado actual del escenario
- `description`: breve descripción del escenario
- `units`: lista que contiene los dispositivos que forman parte del escenario
  - `home`: código de casa
  - `unit`: código de unidad
  - `state`: estado
  - `brightness`: intensidad en caso de que se trate de un dispositivo de tipo regulador

8.2.3. El fichero events.json

Contendrá una lista de eventos. Cada escenario estará representado por un objeto JSON con el siguiente formato:

- `id`: identificador único del evento
- `weatherConditions`: lista de condiciones ambientales
  - `weather`: puede ser temperatura, humedad, presión, viento, lluvia o nieve
  - `condicion`: representada por un entero indica cual debe ser la condición que ha de cumplirse (mayor que, menor que, etc)
  - `value`: valor sobre el que se establece la condición
- `timeConditions`: lista de condiciones temporales
  - `from`: hora de inicio
  - `to`: hora de fín
  - `daysOfWeek`: lista de los días de la semana en los que debe activarse el escenario

- `actions`: lista de los escenarios que deberán ejecutarse cuando se cumplan todas las condiciones

8.3. Diseño de la comunicación cliente - servidor

Para poder comunicarse de manera eficiente con cualquier cliente, el servidor del sistema de automatización va a proporcionar una API REST que permitirá acceder a los recursos del sistema, modificarlos, eliminarlos o crear nuevos. Estos recursos serán: los **módulos** X-10, los **escenarios** y los **eventos**.

Por ejemplo, para consultar la información de un determinado módulo podría realizarse la siguiente petición:

```
GET /modules/:house/:unit
```

donde `house` es un código de casa y `unit` un código de unidad.

La documentación completa de la API puede consultarse en el anexo I (sección 10.1).

8.4. Diseño de la interfaz de usuario: la aplicación *MyDuino*

Como se ha dicho en la sección 1.3, para la interfaz de usuario se ha optado por desarrollar una aplicación Android. Existen multitud de aplicaciones para teléfonos inteligentes diseñadas con el objetivo de facilitar la interacción entre el usuario y los sistemas de automatización presentes en los hogares. Por ello, antes de realizar el diseño que tendrá la aplicación se ha procedido a realizar un breve análisis sobre algunas de las aplicaciones de domótica ya existentes.

8.4.1. Aplicaciones para teléfonos inteligentes en el ámbito de la domótica

Algunas de las aplicaciones más descargadas y con mejor valoración por parte de los usuarios son:

Houseinhand KNX

Es una aplicación para dispositivos iOS[10] de Apple o Android que permite controlar una vivienda de una forma rápida e intuitiva. Permite controlar dispositivos KNX (luces, persianas, climatización...), audiovisuales (televisión, dispositivos de audio, dvd...), videoporteros y cámaras IP (Axis y Mobotix) desde cualquier lugar,

[10] *Iphone Operative System*, en español Sistema Operativo de Iphone

en tiempo real y sin necesidad de hardware adicional. Houseinhand KNX conecta directamente a cualquier pasarela o router KNX IP.

Houseinhand también permite crear escenas personalizadas de manera muy intuitiva. Por ejemplo, la escena «película» que ajustará la luz al 30 % y la temperatura a 25°C. Basta con elegir para cada estancia de la casa y cada dispositivo de esa estancia (que se quiera incluir en el escenario) cuál debe ser el estado que debe adoptar en caso de activarse dicha escena.

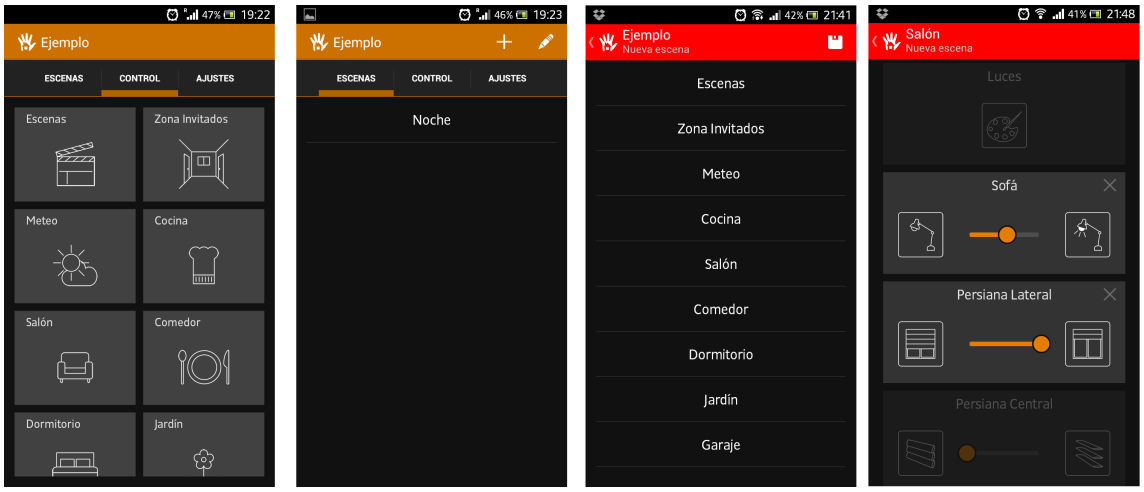


Figura 16: Capturas de pantalla de la aplicación Houseinhand

See-home KNX

See-Home es una aplicación de Schneider Electric para dispositivos con sistema operativo Android e iOS que permite controlar y supervisar, desde cualquier lugar y en tiempo real, cualquier instalación domótica con sistema KNX.

Gracias al software de configuración See-Home Builder es posible adaptar la aplicación a la instalación deseada, en la que podras crear diferentes tipos de dispositivos o controles para supervisar y/o controlar la instalación KNX.

See-home permite:

1. El control de la iluminación
2. Visualizar tantas cámaras IP como haya presentes en la instalación.
3. El control de persianas o toldos mediante botones de subida / bajada o mediante «slider»
4. Visualizar el estado de las alarmas, las cuales se muestan resaltadas en rojo una vez disparadas. Una vez que el defecto ha desaparecido vuelven a su estado normal.
5. Mostrar valores de tipo meteorología con sus unidades de medida correspondientes.

Figura 17: Capturas de pantalla de la aplicación See-home

Philips Hue

La empresa Philips es una de las pioneras en ofrecer productos para el hogar automatizado y conectado y en su catálogo ofrece un amplio abanico de posibilidades, incluyendo esta aplicación que permite controlar remotamente los productos de iluminación Hue que se tenga en casa desde cualquier teléfono Android o iOS.

Igual que ocurría con las aplicaciones anteriores, Philips Hue permite definir escenarios, permitiendo definir configuraciones de iluminación. Pero todavía se puede hacer mucho más con Philips Hue. Gracias a IFTTT se pueden seleccionar «disparadores» que a su vez activan otras opciones.

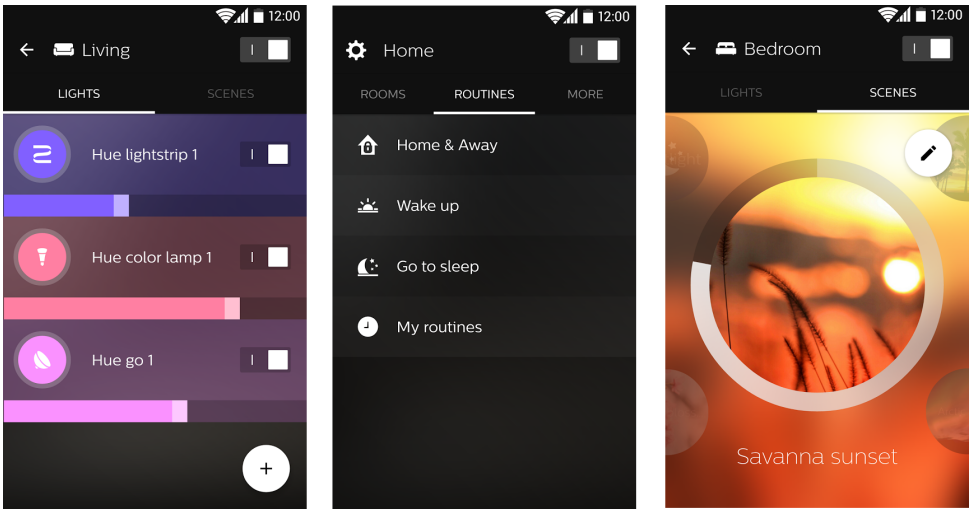


Figura 18: Capturas de pantalla de la aplicación Philips Hue

IFTTT es un servicio web gratuito que permite a los usuarios crear cadenas de sentencias condicionales simples, llamadas «recetas», que se activan en base a los cambios producidos en otros servicios web, como Gmail, Facebook, Instagram o Pinterest, entre otros. Su propio nombre explica en qué consiste su funcionamiento: *If This Then that* (si pasa esto, entonces ocurre esto otro).

Dentro del canal de Hue, en IFTTT, se pueden encontrar ejemplos como los siguientes:

- Cambia de color por SMS: la tonalidad de tu iluminación se cambia a otra distinta y aleatoria si envías un SMS.
- Iluminar al anochecer: programa el encendido de tus luces Hue cuando llegan las 6 de la noche.
- Cambia según el tiempo: si llueve, haz que la luz de tu hogar tome tonos azulados.
- Notificaciones por email: si alguien determinado te envía un correo, haz que tus lámparas parpadeen para avisarte.
- Fotos en Facebook: si te etiquetan en una foto de Facebook, las luces parpadearán.
- Cambio de luces = gol: cada vez que el Manchester United marca un gol, la luz pasa a ser roja.

A continuación se muestra una captura obtenida de la página web oficial de IFTTT, `https://ifttt.com/p/hue/shared`:

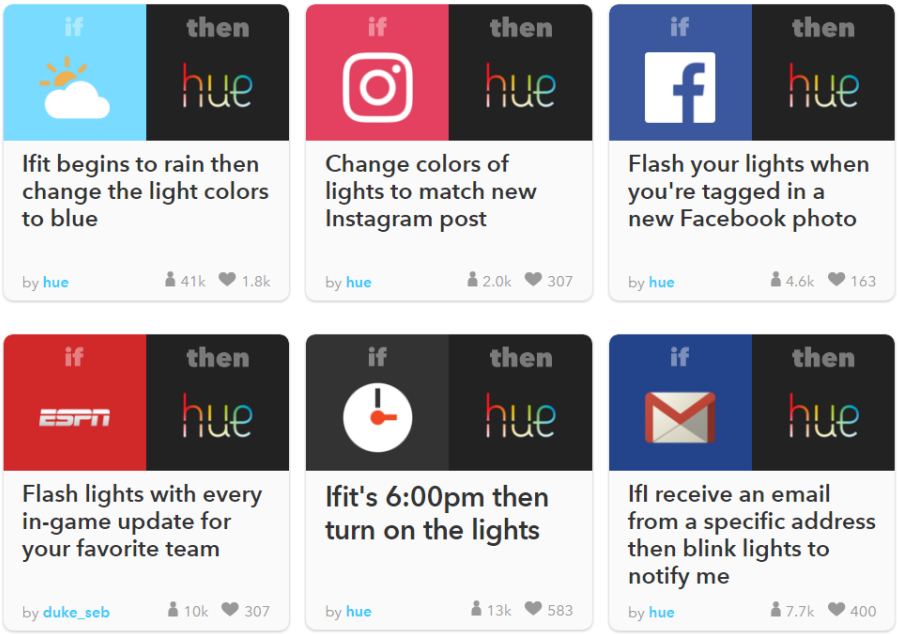


Figura 19: Recetas Philips Hue populares

Existen muchas más aplicaciones enfocadas a la domótica, y aunque cada una tiene sus peculiaridades, la mayoría comparte los siguientes aspectos:

- Una interfaz de usuario sencilla e intuitiva, que permita su uso a cualquier tipo de usuario.
- La posibilidad de definir escenarios. Aunque existen ligeras diferencias en cuanto a lo que se considera «escenario», la gran mayoría de aplicaciones incluye esta posibilidad.
- Mostrar de manera visual y fácilmente accesible información sobre el estado de todos los dispositivos que forman el sistema domótico.

8.4.2. La aplicación *MyDuino*

Para realizar el diseño de la interfaz de usuario se han tenido en cuenta los casos de uso descritos en la sección 5, cubriendo todos los requisitos funcionales.

Lista de dispositivos

En esta pantalla se muestra una lista con todos los dispositivos presentes en el sistema X-10 que sean de tipo dispositivo, regulador o desconocido. Cada elemento de la lista contiene una imagen que indica, en función del color, si la unidad está encendida, apagada, o se desconoce su estado.

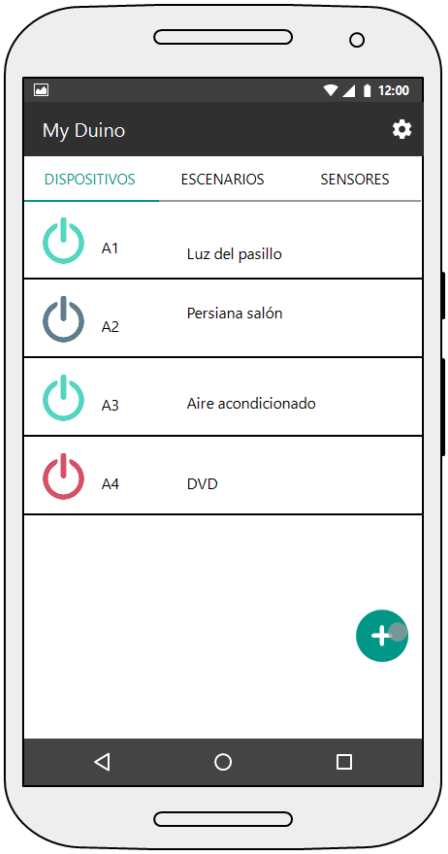


Figura 20: Pantalla lista de dispositivos

Crear un nuevo dispositivo

Al presionar sobre el botón con el icono + que aparece en la pantalla «Lista de dispositivos» se abre la pantalla «Nuevo dispositivo». En ella el usuario puede dar valor a los diferentes atributos que definen un dispositivo.

Si se presiona sobre el icono de guardar de la barra de navegación, el dispositivo se almacenará en el sistema y volverá a aparecer la lista de dispositivos (con el nuevo dispositivo incluido). Si se presionara el icono de cancelar se descartarían los cambios y se volvería a la pantalla «Lista de dispositivos».

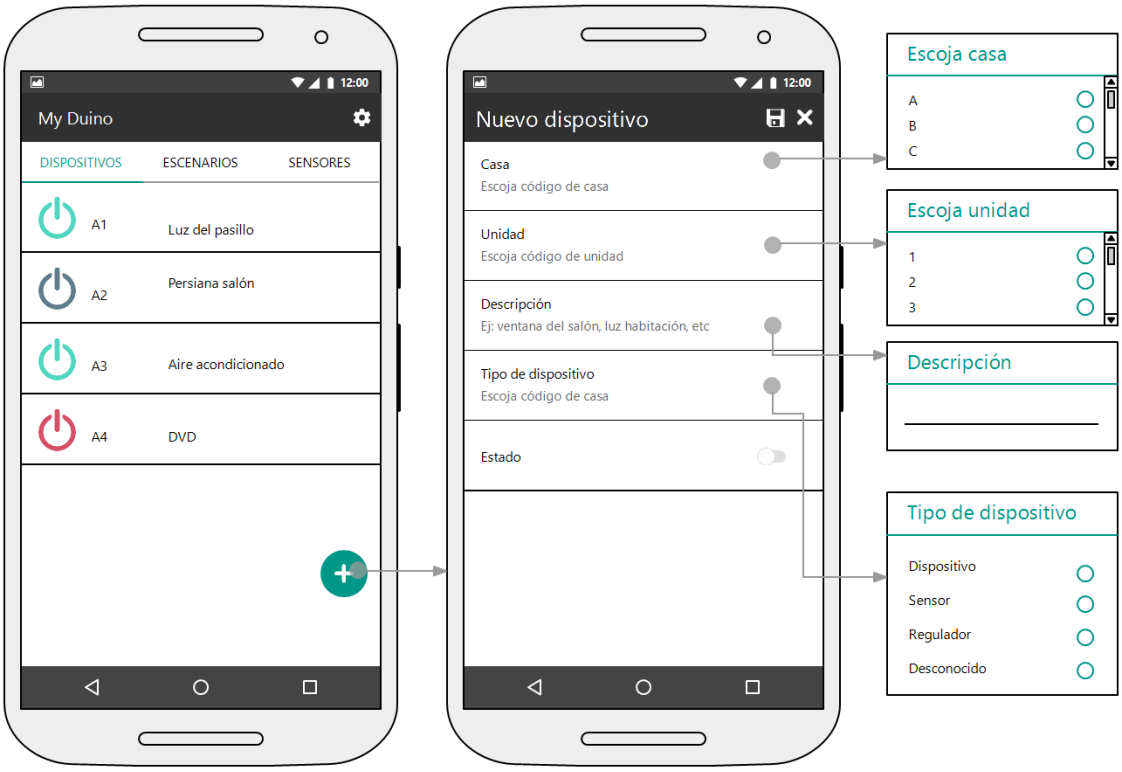


Figura 21: Pantalla crear dispositivo

Editar, encender o apagar, reducir o aumentar la intensidad y borrar un dispositivo

Al presionar sobre cualquier elemento de la lista de dispositivos aparece la pantalla «Detalle dispositivo» desde donde se puede consultar toda la información de dicho dispositivo.

Desde esta misma pantalla también podemos:

1. Indicar que queremos apagar o encender el dispositivo
2. Aumentar o disminuir su intensidad (si es de tipo regulador)
3. Eliminarlo del sistema

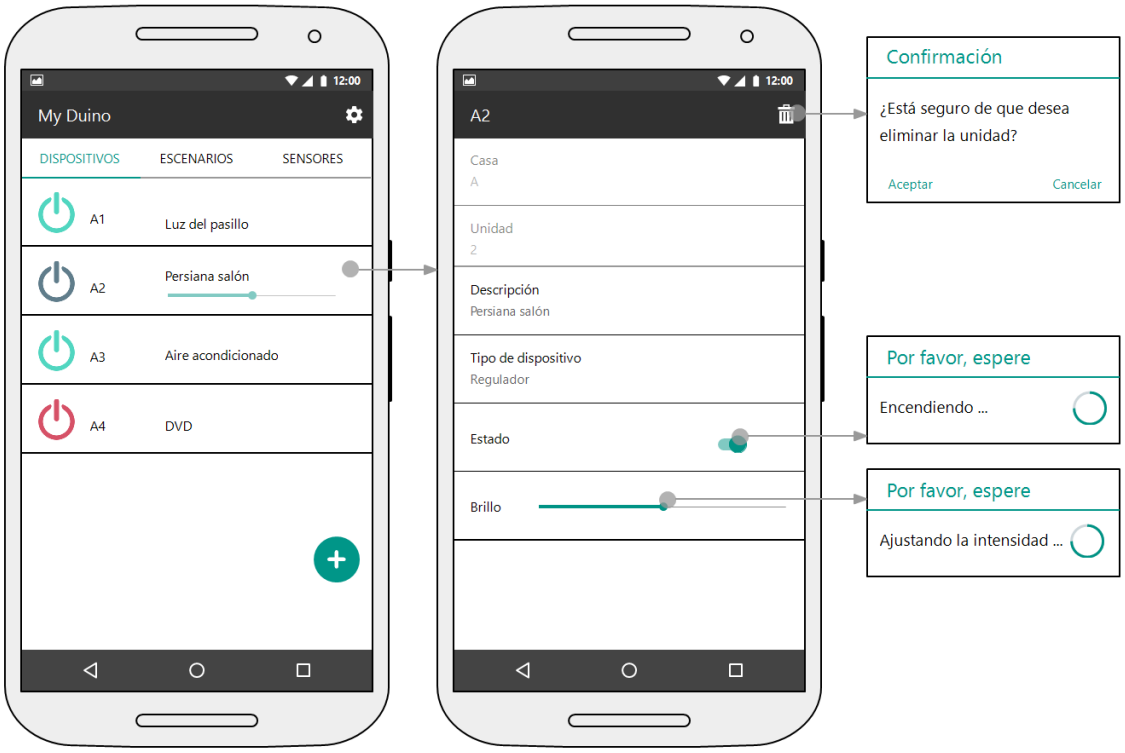


Figura 22: Pantalla editar dispositivo

Lista de escenarios

En esta pantalla se muestra una lista con todos los escenarios que han sido definidos por el usuario. Cada elemento de la lista contiene un *switch*, que indica si el escenario se encuentra en ese momento activo o no y, a su vez, permite activarlo o desactivarlo.

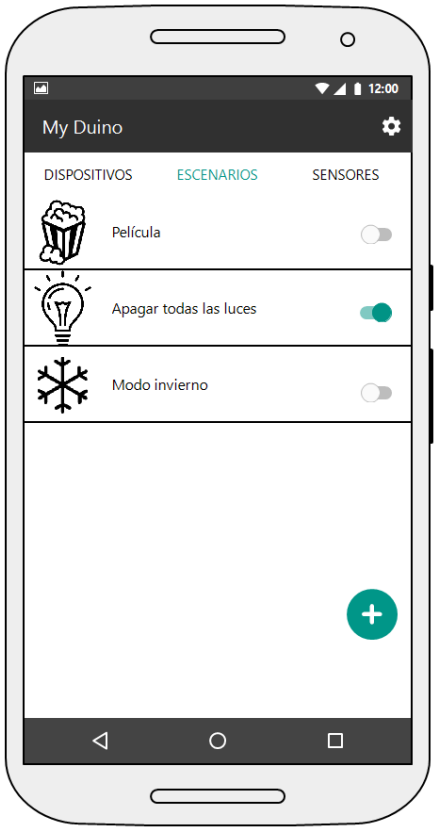


Figura 23: Pantalla lista de escenarios

Crear un escenario

Al presionar sobre el botón con el icono + que aparece en la pantalla «Lista de escenarios» se abre la pantalla «Nuevo escenario». En ella el usuario puede dar valor a los diferentes atributos que definen un escenario.

Si se presiona sobre el icono de guardar de la barra de navegación, la unidad se almacenará en el sistema y volverá a aparecer la lista de unidades (con la nueva unidad incluida). Si se presionara el icono de cancelar se descartarían los cambios y se volvería a la pantalla «Lista de unidades».

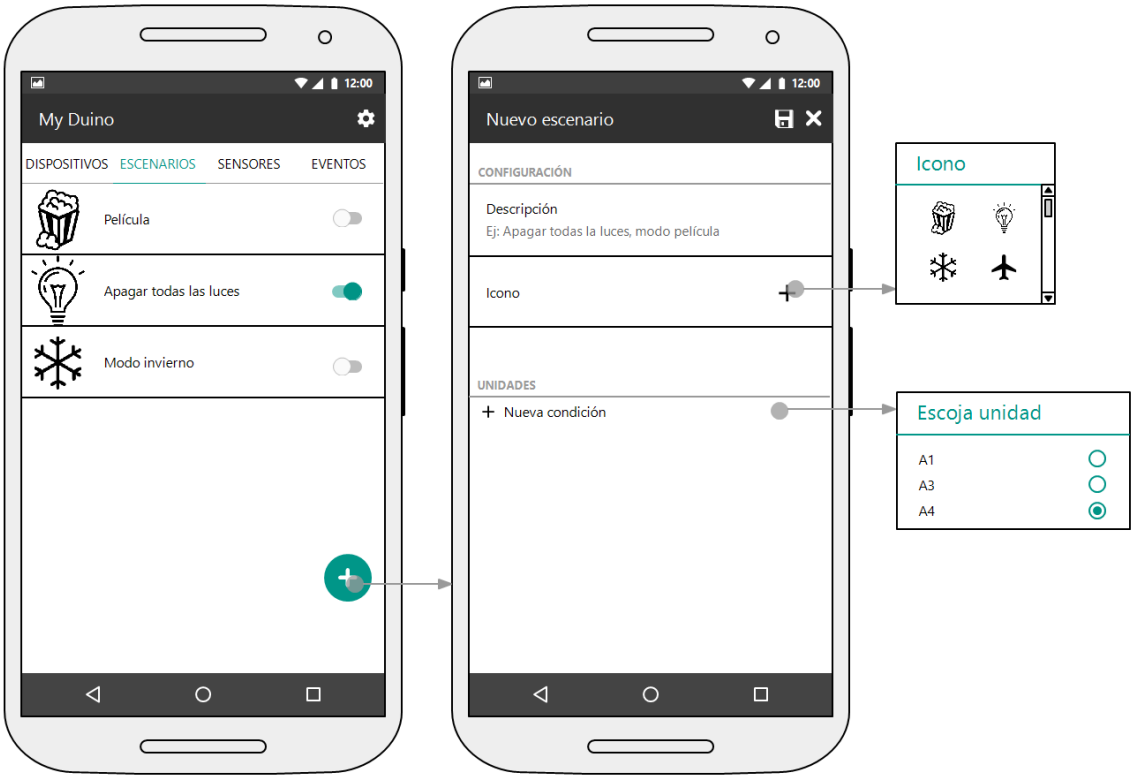


Figura 24: Pantalla crear escenario

Editar un escenario

Al presionar sobre cualquier elemento de la lista de escenarios aparece la pantalla «Detalle escenario» desde donde se puede consultar toda la información asociada a dicho escenario.

Desde esta misma pantalla también podemos eliminar el escenario del sistema presionando el icono de la papelera que aparece en la barra de navegación.

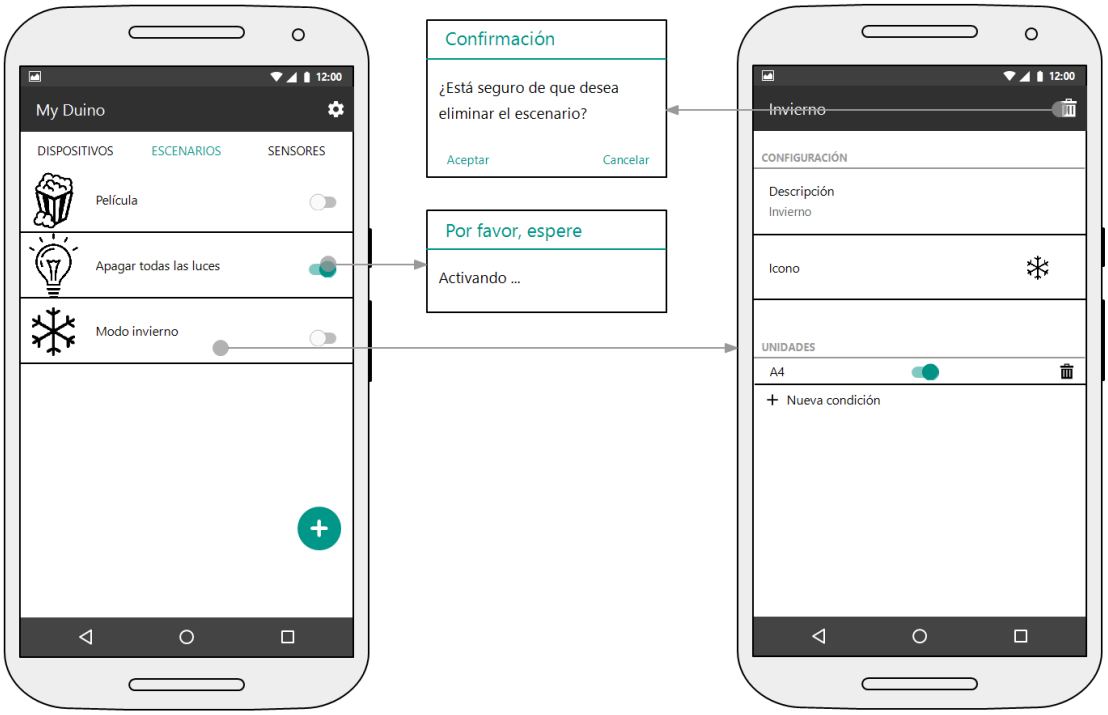


Figura 25: Pantalla editar escenario

Lista de eventos

En esta pantalla se muestra una lista con todos los eventos que han sido definidos por el usuario.

Figura 26: Pantalla lista de eventos

Crear un evento

Al presionar sobre el botón con el icono + que aparece en la pantalla «Lista de eventos» se abre la pantalla «Nuevo evento». En ella el usuario puede dar valor a los diferentes atributos que definen un evento.

Si se presiona sobre el icono de guardar de la barra de navegación, el evento se almacenará en el sistema y volverá a aparecer la lista de eventos (con el nuevo evento incluido). Si se presionara el icono de cancelar se descartarian los cambios y se volvería a la pantalla «Lista de eventos».

Al pulsar sobre «Nueva condición» aparece un cuadro de diálogo preguntando al usuario si la condición que desea definir es ambiental o temporal. Al pulsar sobre «Nueva acción» aparece un cuadro de diálogo con una lista de todos los escenarios definidos en el sistema.

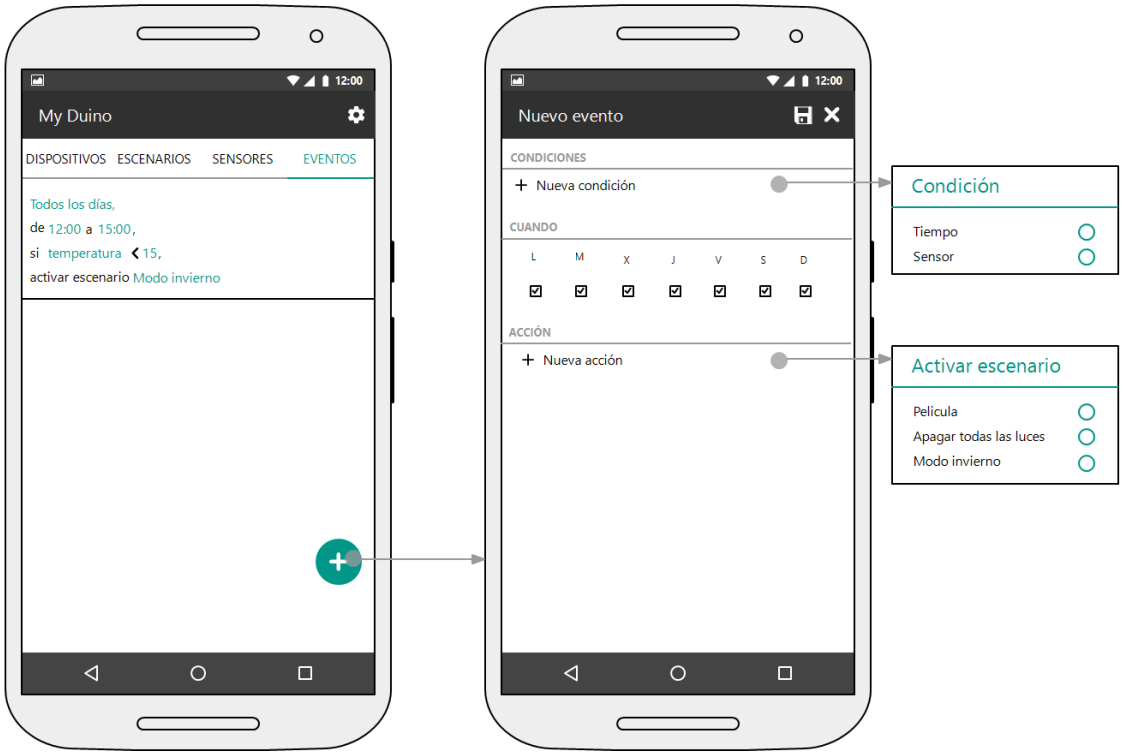


Figura 27: Pantalla crear evento

Añadir una condición ambiental a un evento

Si en el cuadro de diálogo «Nueva condición» el usuario presiona sobre «Condición ambiental» se abre un nuevo cuadro de diálogo que permite al usuario definir los atributos de la condición.

Tras rellenar los difrentes campos y pulsar en «Aceptar» la condición se añadirá a la lista de condiciones del evento actual.

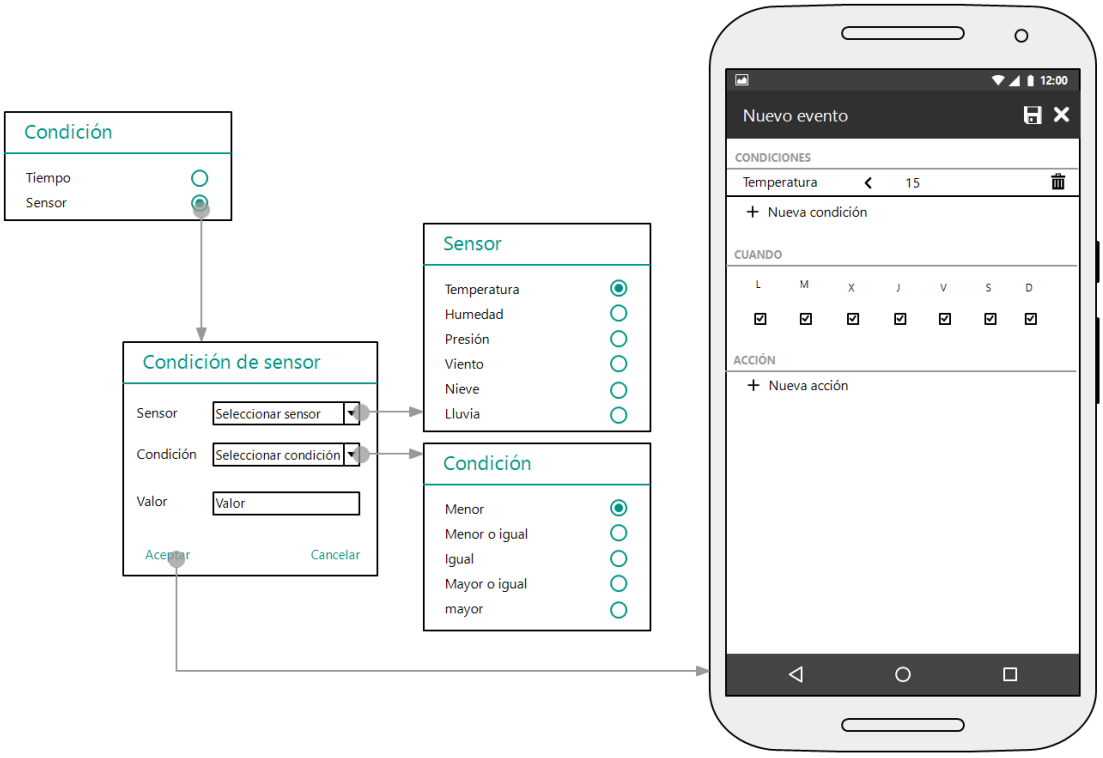


Figura 28: Acción añadir condición ambiental a un evento

Añadir una condición de tiempo a un evento

Si en el cuadro de diálogo «Nueva condición» el usuario presiona sobre «Condición temporal» se abre un nuevo cuadro de diálogo que permite al usuario definir la hora de inicio y fin de dicha condición. Tras ello, la condición se añadirá a la lista de condiciones del evento actual.

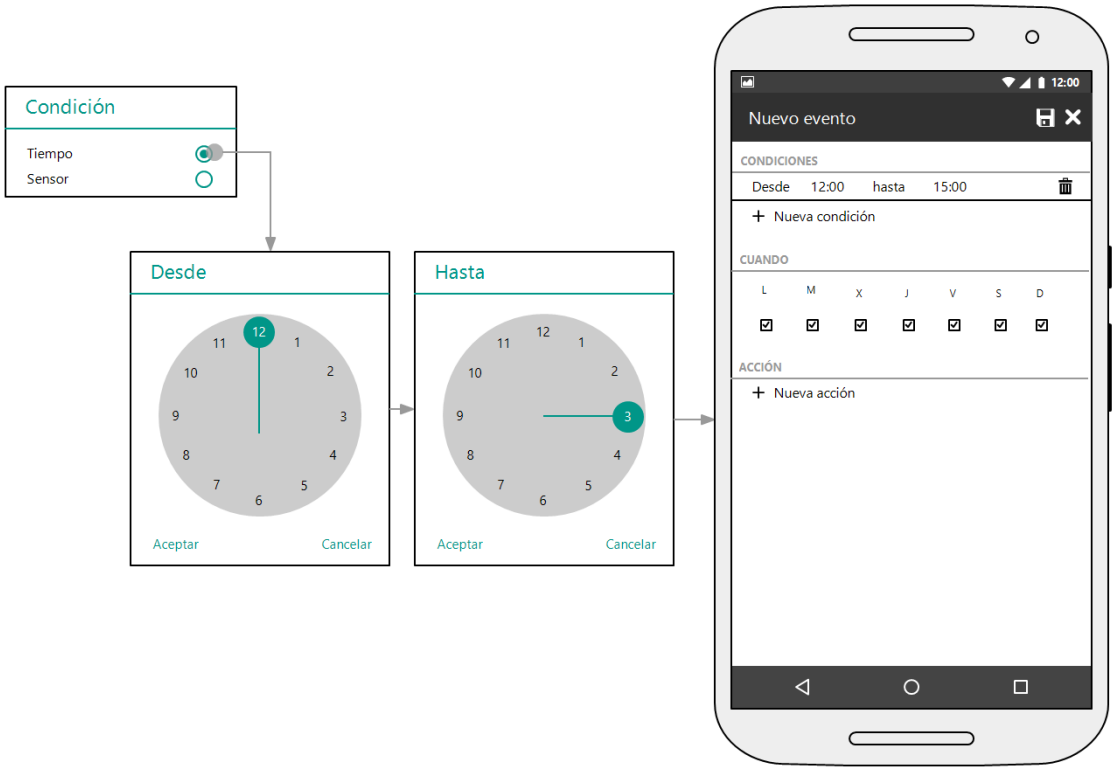


Figura 29: Acción añadir condición de tiempo a un evento

Añadir una acción a un evento

Al pulsar sobre un escenario este se añadirá a la lista de acciones del evento actual.

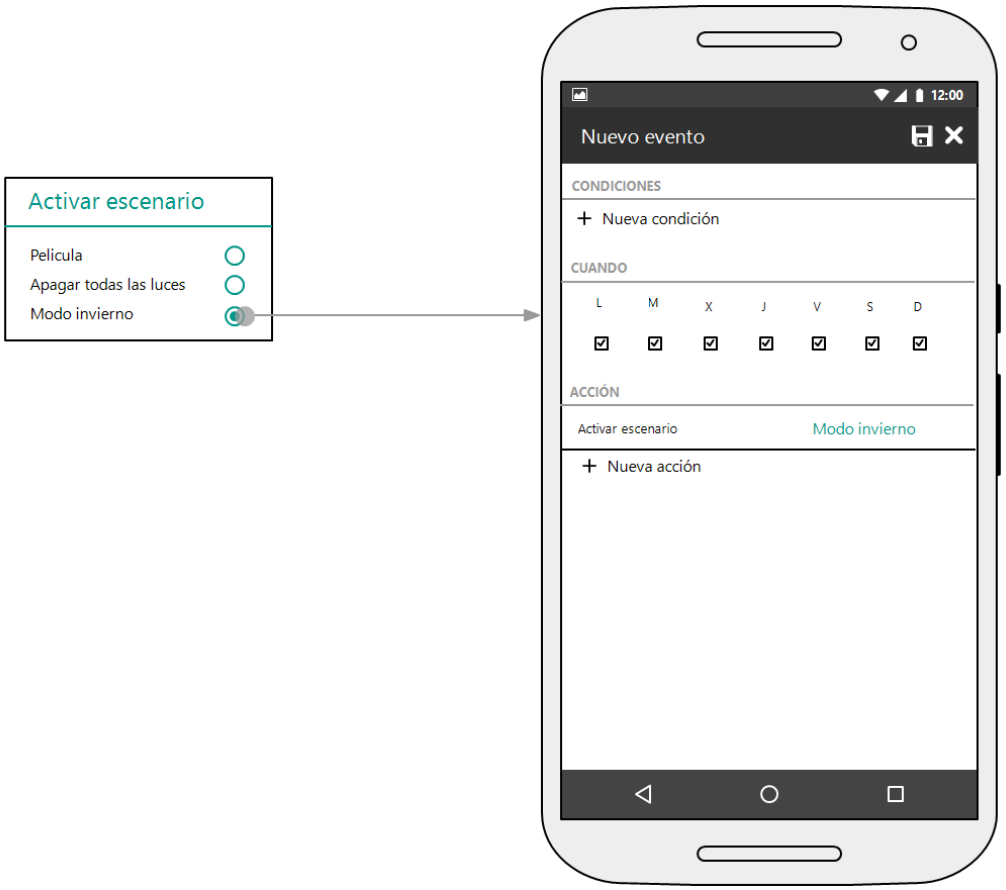


Figura 30: Acción añadir acción a un evento

Lista de sensores

En esta pantalla esta formada por dos secciones. En la primera, se muestra una lista con todos los sensores físicos que han sido definidos por el usuario. Cada elemento de la lista contiene una imagen que indica, en función del color, si el sensor está activo o no, o si se desconoce su estado.

En la segunda sección, se muestra una lista con todos los sensores virtuales disponibles: temperatura, presión, humedad, viento, lluvia y nieve, así como la información ambiental relativa a cada sensor.

Al presionar sobre el botón con el icono + se abre la pantalla «Nuevo dispositivo».

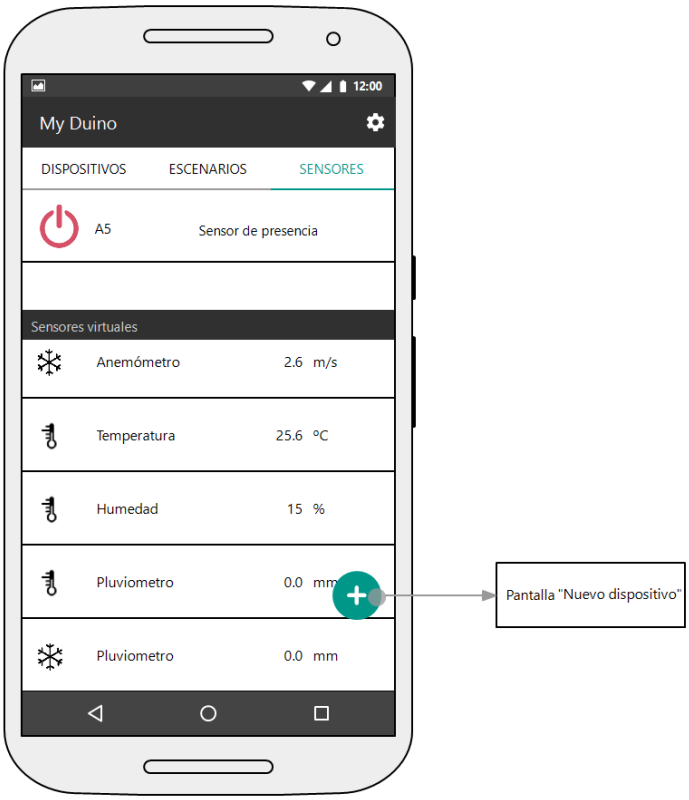


Figura 31: Pantalla lista de sensores X-10

Pantalla de preferencias de usuario

Esta pantalla permite configurar la dirección IP y puerto del servidor que se encarga de gestionar el sistema domótico, así como los tiempos de conexión y respuesta que debe esperar la aplicación como máximo para obtener una respuesta de dicho servidor ante una petición.

También desde aquí se puede indicar el país y la ciudad en la que se encuentra la vivienda en la que está instalado el sistema domótico para recoger la información ambiental en función de estos valores.

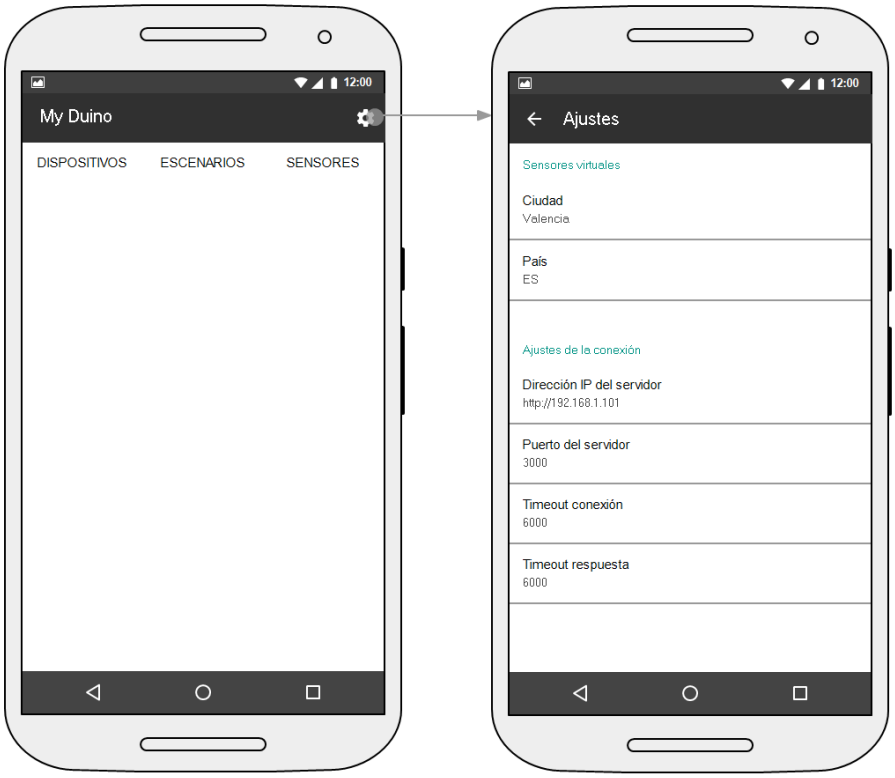


Figura 32: Pantalla de preferencias

9. Implementación física

9.1. Presupuesto

A continuación se muestra una tabla con los elementos empleados en el desarrollo del proyecto y sus precios[11] aproximados en el mercado a fecha de agosto de 2016:

Dispositivo	Precio €
Placa Arduino Uno Rev3	29,00
Placa Intel Galileo Gen2	79,00
Módulo X-10 de persiana SW12	55,83
Sensor X-10 de presencia MS13E	18,95
Módulo X-10 con receptor para mandos a distancia TM13	28,00
Módulo Interfaz bidireccional XM10	42,50
Dispositivo Android	-
Total	253,28

Tabla 17: Dispositivos y precios

9.2. Montaje

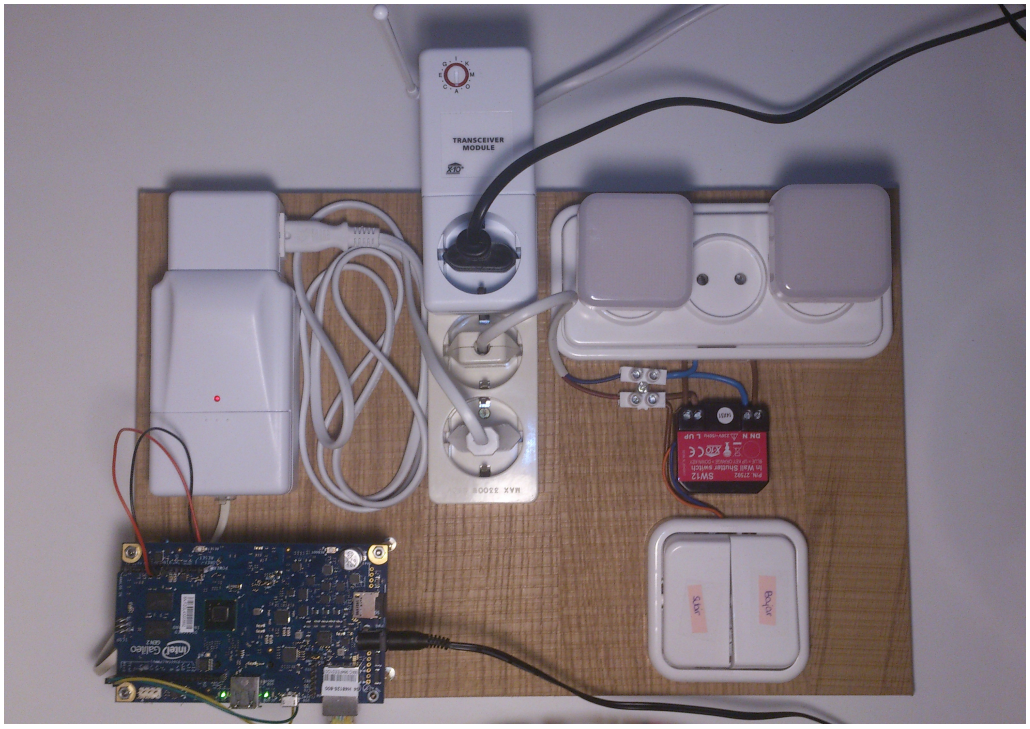


Figura 33: Vista del montaje del sistema de automatización

[11] Precios consultados en http://latiendadedomotica.com, www.arduino.cc y www.amazon.es

En la figura 33 puede verse:

- El módulo de persiana en la esquina superior derecha.
- Ambas placas en la esquina inferior izquierda (la Galileo encima de la Arduino).
- El receptor de radio frecuencia en la parte superior, en el centro.
- El PLC en la esquina superior izquierda.

Para hacer pruebas del sistema, el módulo de persiana se ha conectado a dos bombillas que simulan el comportamiento de la misma (la bombilla izquierda indica la activación del motor en sentido de subida y la derecha en el de bajada).

Por su parte, el receptor de radio frecuencia permite mandar comandos X-10 a través de un mando a distancia de radiofrecuencia. La utilidad de tener algo así en casa es innegable y, además, ha permitido simular la presencia de sensores para realizar pruebas durante el desarrollo del proyecto.

El receptor de radiofrecuencia ofrece, además, la función de módulo de aparato. A él se ha conectado una lámpara para poder probar su funcionamiento.

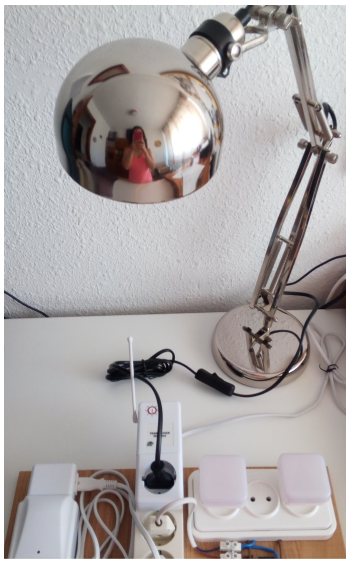


Figura 34: Receptor de radio frecuencia y lámpara conectada

La figura 35 muestra el detalle de las dos placas desmontadas de su ubicación (Arduino a la izquierda y Galileo a la derecha).

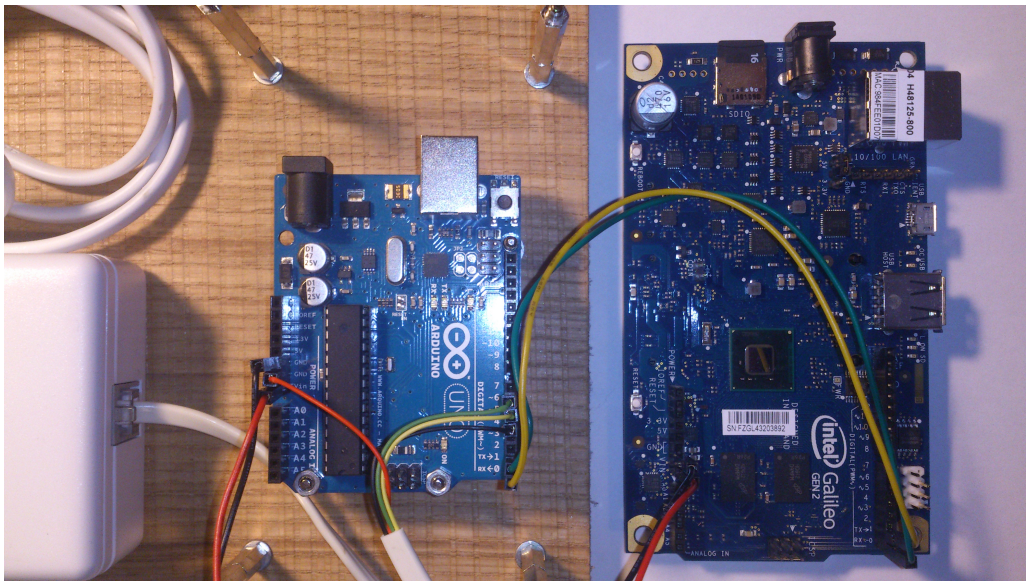


Figura 35: Vista de las conexiones

10. Conclusiones

Tras finalizar el proyecto y después de dar solución a todos los problemas encontrados, se puede concluir que el objetivo presentado en la sección 1.3 ha sido completamente cubierto. Se ha desarrollado un software fácil de manejar y a un precio asequible para alguien que quiera tener monitorizados algunos dipositivos en casa sin tener que entrar en grandes obras o instalaciones.

El interés de haber desarrollado un sistema completo propio reside en que se obtiene un sistema totalmente abierto, ampliable y personalizable, en caso de surgir nuevas necesidades.

También se ha desarrollado la aplicación Android que permite gestionar dicho sistema, lo que me ha permitido introducirme en el mundo de la programación en Android, que era una de las motivaciones personales que me llevaron a elegir este proyecto.

10.1. Trabajos futuros

Para el futuro lo más interesante sería mejorar de alguna manera la persistencia de los datos en la propia aplicación Android de manera que no hiciera falta pedir al servidor todos los datos de unidades, escenarios y eventos cada vez que se iniciara la aplicación. Aunque por un lado se incrementaría el espacio ocupado por la aplicación en la memoria del teléfono móvil, por el otro se reduciria el tiempo de arranque que, en algunos casos, puede llegar a ser considerable. Sería interesante buscar el punto de equilibrio entre ambos intereses.

También sería interesante mejorar la persistencia del sistema cambiando el sistema de ficheros JSON por una base de datos no relacional MongoDB. El sistema operativo Linux instalado en la placa Intel Galileo viene preparado con todo lo necesario para comenzar a utilizar este tipo de tecnológia que proporciona una alternativa más eficiente y limpia que la utilizada en este proyecto.

Mongo es una base de datos no relacional (NoSQL) de código abierto que guarda los datos en documentos tipo JSON pero en forma binaria (BSON) para hacer la integración de una manera más rápida. Se pueden ejecutar operaciones en JavaScript en su consola en lugar de consultas SQL. Además tiene una gran integración con Node gracias a la librería Moongose. Debido a su flexibilidad es muy escalable y ayuda al desarrollo ágil de proyectos web.

Aunque si no se conoce con anterioridad resulta más complejo empezar a utilizar esta tecnología que trabajar directamente con ficheros JSON, una vez se aprende, su uso es mucho más sencillo y eficiente para el desarrollador que manejar ficheros del sistema de archivos. Además MongoDB tiene un controlador que permite controlar y manejar la base de datos MongoDB desde NodeJS.

Anexo I: Interfaz de Programación de Aplicaciones del servidor

El servidor desarrollado proporciona una API que puede ser utilizada desde cualquier aplicación o cliente de servicios REST. Una interfaz hecha para desarrolladores que proporciona una vía de acceso a los recursos del sistema de automatización desarrollado de una forma entendible y accesible, permitiendo un cierto filtrado y devolviendo al usuario un fichero JSON de respuesta.

URL base

http://server-ip-address:server-port

GET

GET /modules

Devuelve toda la información existente de todos los módulos X-10 que hay en el sistema.

Ejemplo de respuesta


```
[
  {
    "url": "/A/1/",
    "type": 1,
    "on": true,
    "name": "Luz pasillo"
  },
  {
    "url": "/A/2/",
    "type": 2,
    "brightness": 80,
    "name": "Ventana salón"
  },
  {
    "url": "/B/3/",
    "type": 3,
    "on": false,
    "name": "Aire acondicionado"
  }
]
```

GET /modules/:house

Devuelve toda la información existente de todos los módulos X-10 que hay en el sistema para un determinado código de casa.

Parámetros

house requerido	Un código de casa. Debe estar comprendido entre la A y la P

Ejemplo de petición

```
GET http://192.168.1.117/modules/A
```

Ejemplo de respuesta

```
[
  {
    "url": "/A/1/",
    "type": 1,
    "on": true,
    "name": "Luz pasillo"
  },
  {
    "url": "/A/2/",
    "type": 2,
    "brightness": 80,
    "name": "Ventana salón"
  }
]
```

GET /modules/:house/:unit

Devuelve toda la información existente para el módulo X-10 especificado en los parámetros de la url.

Parámetros

house requerido	Un código de casa. Debe estar comprendido entre la A y la P
unit requerido	Un código de unidad. Debe estar comprendido entre 1 y 15 (en valor hexadecimal). Valores ejemplo: 1, 2, A, F

Ejemplo de petición

```
GET http://192.168.1.117/modules/A/1
```

Ejemplo de respuesta


```
{
  "url": "/A/1/",
  "type": 1,
  "on": true,
  "name": "Luz pasillo"
}
```


GET /scenarios

Devuelve todos los escenarios que haya definidos en el sistema domótico.

Ejemplo de respuesta


```
{
  "0": {
    "enabled": false,
    "description": "Regulador de temperatura",
    "units": [{
      "house": "A",
      "unit": 2,
      "state": 2
    }]
  },
  "1": {
    "enabled": true,
    "description": "Modo película",
    "units": [{
      "house": "A",
      "unit": 2,
      "state": 3
    }]
  }
}
```


GET /scenarios/:id

Devuelve el escenario con el identificador especificado

Parámetros

id requerido	Identificador del escenario

Ejemplo de petición

```
GET http://192.168.1.117/scenarios/1
```

Ejemplo de respuesta


```
{
  "enabled": false,
  "description": "Regulador de temperatura",
    "units": [{
    "house": "A",
    "unit": "2",
    "state": "2"
  }]
}
```


GET /events

Devuelve todos los eventos que haya definidos en el sistema domótico.

Ejemplo de respuesta


```
{
  "0": {
    "timeConditions": [{
      "from": "17:00",
      "to": "18:00",
      "daysOfWeek": [1,2,3,4,5]
    }]
  },
  "1": {
    "weatherConditions": [{
      "weather": "temperature",
      "condition": "isLowerOrEqual",
      "value": 50
    }]
  }
}
```

GET /events/:id

Devuelve el escenario con el identificador especificado

Parámetros

id requerido	Identificador del evento

Ejemplo de petición

```
GET http://192.168.1.117/events/1
```

Ejemplo de respuesta

```
{
  "timeConditions": [
    {
      "from": "17:00",
      "to": "18:00",
      "daysOfWeek": [1,2,3,4,5]
    }
  ]
}
```

POST

POST /modules

Crea un nuevo módulo X-10.

Cuerpo de la petición

Objeto JSON con el siguiente formato:

- `url`: formada por el código de casa y el código de unidad. Por ejemplo: '/A/2/'
- `type`: tipo de módulo X-10:
  0 - Desconocido
  1 - Dispositivo
  2 - Regulador
  3 - Sensor
- `name`: nombre o breve descripción del módulo X-10

Ejemplo de petición

```
POST http://192.168.1.117/modules
Content-Type: application/json
body:
```

```
{
  "url": "/A/5/",
  "type": 1,
  "name": "Luz habitación"
}
```

Ejemplo de respuesta

```
Contenido:
{
  "url": "/A/5/",
  "type": 1,
  "name": "Luz habitación"
}
```

POST /scenarios/:id

Crea un nuevo escenario.

Parámetros

id requerido	Identificador del escenario

Cuerpo de la petición

Objeto JSON con el siguiente formato:

- `description`: cadena de caracteres de longitud menor que 30 que describe el escenario
- `units`: lista que contiene las unidades que forman parte del escenario
  - `home`: código de casa
  - `unit`: código de unidad
  - `state`: estado
  - `brightness`: intensidad en caso de que se trate de un dispositivo de tipo regulador

Ejemplo de petición

```
POST http://192.168.1.117/scenarios/1
Content-Type: application/json
body:
```

```
{
  "enabled": false,
  "description": "Modo película",
  "units": [{
    "house": "A",
    "unit": "2",
    "state": "2"
  }]
}
```

Ejemplo de respuesta

```
Código de estado: 200
Contenido: ScenarioCreatedSuccessfully
```

POST /events/:id

Crea un nuevo evento.

Parámetros

id requerido	Identificador del evento

Cuerpo de la petición

Objeto JSON con el siguiente formato:

- `weatherConditions`: lista de objetos, cada uno de los cuales representa una condición ambiental
  - `weather`: cadena de caracteres que representa la condición ambiental a tener en cuenta. Puede tomar los valores: 'temperature', 'humidity', 'pressure, 'wind', 'rain' o 'snow'
  - `condicion`: representada por un entero indica cuál debe ser la condición que ha de cumplirse

< 0	$= 2$	> 4
≤ 1	≥ 3	

  - `value`: valor con el que se realizará la comparación

- `timeConditions`: lista de objetos, cada uno de los cuales representa una condición temporal
  - `from`: hora de inicio
  - `to`: hora de fin
  - `daysOfWeek`: lista con los días de la semana en que debe activarse el escenario
- `actions`: lista que contiene los identificadores de los escenarios que deberán ejecutarse cuando se cumplan todas las condiciones

Ejemplo de petición

```
POST http://192.168.1.117/events
Content-Type: application/json
body:
```

```
{
  "timeConditions": [{
    "from": "17:00",
    "to": "18:00",
    "daysOfWeek": [1,2,3,4,5]
  }],
  "weatherConditions": [{
    "weather": "temperature",
    "condition": 1,
    "value": 50
  }],
  "actions": [2]
}
```

Ejemplo de respuesta

```
Código de estado: 200
Contenido: EventCreatedSuccessfully
```

PUT

PUT /modules/:house/:unit

Actualiza la información asociada al módulo X-10 especificado en la URL.

Parámetros

house requerido	Un código de casa. Debe estar comprendido entre la A y la P
unit requerido	Un código de unidad. Debe estar comprendido entre 1 y 15 (en valor hexadecimal). Valores ejemplo: 1, 2, A, F

Cuerpo de la petición

on opcional	Indica si el módulo debe ser activado, desactivad o aumentar/disminuir su intensidad. Sus valores pueden ser: 2 Encender 3 Apagar 4 Disminuir intensidad 5 Aumentar intensidad
type opcional	El tipo de dispositivo. Su valor puede ser: 0 Desconocido 1 Dispositivo (*Appliance*) 2 Regulador de intensidad (*Dimmer*) 3 Sensor
brightness opcional	Un valor de intensidad comprendido entre 0 y 100
name opcional	Una descripción del módulo. Debe tener una longitud inferior a 10 carácteres.

Ejemplo de petición

```
PUT http://192.168.1.117/modules/A/1
body: on=3&type=2
```

Ejemplo de respuesta

```
{
  "url": "/A/1/",
  "type": 2,
  "on": false,
  "name": "Luz pasillo"
}
```

PUT /scenarios/:id

Actualiza un escenario.

Parámetros

id requerido	Identificador del escenario

Cuerpo de la petición

Objeto JSON con el siguiente formato:

- `description`: cadena de caracteres de longitud menor que 30 que describe el escenario
- `units`: lista que contiene las unidades que forman parte del escenario
  - `home`: código de casa
  - `unit`: código de unidad
  - `state`: estado
  - `brightness`: intensidad en caso de que se trate de un dispositivo de tipo regulador

No se requiere incluir todos los campos, si no solo aquellos que hayan cambiado de valor.

Ejemplo de petición

```
POST http://192.168.1.117/scenarios/1
Content-Type: application/json
```

`body:`

```
{
  "description": "Modo película de terror",
}
```

Ejemplo de respuesta

```
{
  "enabled": false,
  "description": "Modo película de terror",
  "units": [{
    "house": "A",
    "unit": "2",
    "state": "2"
  }]
}
```

PUT /scenarios/:id/state/:state

Enciendo o apaga el escenario con el identificador especificado

Parámetros

id requerido	Identificador del escenario
state requerido	Nuevo estado del dispositivo

Ejemplo de petición

`PUT http://192.168.1.117/scenarios/1/state/2`

Ejemplo de respuesta

```
{
  "scenario": 1,
  "state": 2
}
```

PUT /events/:id

Actualiza un evento.

Parámetros

id requerido	Identificador del evento

Cuerpo de la petición

Objeto JSON con el siguiente formato:

- `weatherConditions`: lista de objetos, cada uno de los cuales representa una condición ambiental
  - `weather`: cadena de caracteres que representa la condición ambiental a tener en cuenta. Puede tomar los valores: 'temperature', 'humidity', 'pressure', 'wind', 'rain' o 'snow'
  - `condicion`: representada por un entero indica cuál debe ser la condición que ha de cumplirse

    < 0 $= 2$ > 4
    ≤ 1 ≥ 3
  - `value`: valor con el que se realizará la comparación
- `timeConditions`: lista de objetos, cada uno de los cuales representa una condición temporal
  - `from`: hora de inicio
  - `to`: hora de fin
  - `daysOfWeek`: lista con los días de la semana en que debe activarse el escenario
- `actions`: lista que contiene los identificadores de los escenarios que deberán ejecutarse cuando se cumplan todas las condiciones

No se requiere incluir todos los campos, si no solo aquellos que hayan cambiado de valor.

Ejemplo de petición

```
PUT http://192.168.1.117/event/1
Content-Type: application/json
body:
```

```
{
  "timeConditions": [{
    "from": "15:00",
    "to": "18:00",
    "daysOfWeek": [6,7]
  }]
}
```

Ejemplo de respuesta

```
{
  "timeConditions": [{
    "from": "15:00",
    "to": "18:00",
    "daysOfWeek": [6,7]
  }]
}
```

PUT /settings/date/ntp

Habilita o deshabilita el protocolo de tiempo de red (NTP, del inglés *Network Time Protocol*) en el servidor

Parámetros

activate requerido	Booleano que indica si el protocolo ntp debe ser habilitado o deshabilitado

Ejemplo de petición

```
PUT http://192.168.1.117/settings/date/ntp
body: activate=true
```

Ejemplo de respuesta

```
Código de estado: 200
```

PUT /settings/date

Actualiza la fecha y hora del servidor. Sólo tiene sentido si el NTP está deshabilitado.

Cuerpo de la petición

date requerido	Fecha en el formato: "YYYY-MM-DD HH:MM:SS"

Ejemplo de petición

```
PUT http://192.168.1.117/settings/date
body: date="2014-11-08 06:40:00"
```

Ejemplo de respuesta

```
Código de estado: 200
```

PUT /settings/timezone

Cambia la zona horaria del servidor.

Cuerpo de la petición

timezone requerido	Zona horaria en la que debe estar el servidor

Ejemplo de petición

```
PUT http://192.168.1.117/preferences/timezone
body: timezone=Europe/Paris
```

Ejemplo de respuesta

```
Código de estado: 200
```

PUT /preferences/virtual-sensors

Habilita o deshabilita el uso de los «sensores virtuales».

Cuerpo de la petición

virtualsensors requerido	Booleano que indica si los sensores virtuales deben estar habilitados o deshabilitados

Ejemplo de petición

```
PUT http://192.168.1.117/preferences/virtual-sensors
body: virtualsensors=true
```

Ejemplo de respuesta

```
Código de estado: 208
```

PUT /preferences/city

Indica, en caso de que el uso de los sensores virtuales esté habilitado, sobré qué ciudad deben consultarse los datos del clima.

Cuerpo de la petición

city requerido	Un cadena con el nombre de la ciudad, seguido por el código del país, separando ambas cadenas con una coma.

Ejemplo de petición

```
PUT http://192.168.1.117/preferences/virtual-sensors
body: city=Valencia,ES
```

Ejemplo de respuesta

```
Código de estado: 200
```

DELETE

DELETE /modules

Elimina la información de todos los módulos X-10 del sistema.

Ejemplo de respuesta

```
Código de estado: 200
Contenido: ModulesRemovedSuccessfully
```

DELETE /modules/:house

Elimina la información de todos los módulos X-10 del sistema para una determinada casa.

Parámetros

house requerido Un código de casa. Debe estar comprendido entre la A y la P

Ejemplo de petición

```
DELETE http://192.168.1.117/modules/A
```

Ejemplo de respuesta

```
Código de estado: 200
Contenido: ModulesRemovedSuccessfully
```

DELETE /modules/:house/:unit

Elimina toda la información del módulo X-10 especificado en la url.

Parámetros

house requerido	Un código de casa. Debe estar comprendido entre la A y la P
unit requerido	Un código de unidad. Debe estar comprendido entre 1 y 15 (en valor hexadecimal). Valores ejemplo: 1, 2, A, F

Ejemplo de petición

```
DELETE http://192.168.1.117/modules/A/1
```

Ejemplo de respuesta

```
Código de estado: 200
Contenido: ModulesRemovedSuccessfully
```

DELETE /scenarios

Elimina todos los escenarios que haya en el sistema.

Ejemplo de respuesta

```
Código de estado: 200
Contenido: ScenariosRemovedSuccessfully
```

DELETE /scenarios/:id

Elimina el escenario con el id especificado en la url.

Parámetros

id requerido	Identificador del escenario a eliminar

Ejemplo de petición

```
DELETE http://192.168.1.117/scenarios/1
```

Ejemplo de respuesta

```
Código de estado: 200
Contenido: ScenariosRemovedSuccessfully
```

DELETE /events

Elimina todos los eventos que haya en el sistema.

Ejemplo de respuesta

```
Código de estado: 200
Contenido: EventssRemovedSuccessfully
```

DELETE /events/:id

Elimina el evento con el id especificado en la url.

Parámetros

id requerido	Identificador del escenario a eliminar

Ejemplo de petición

```
DELETE http://192.168.1.117/events/1
```

Ejemplo de respuesta

```
Código de estado: 200
Contenido: ScenariosRemovedSuccessfully
```

DELETE /reset

Elimina toda la información que hay en el sistema.

POST /demo

Inicializa el sistema en un modo demostración creando algunos módules, escenarios y eventos de prueba.

Control de errores

A continuación se listan los mensajes de error que puede devolver el servidor:

Mensaje	Código	Interpretación
HouseIdNotValid	400	Si el identificador de casa no es un carácter comprendido entre la A y la P
UnitIdNotValid	400	Si el identificador de unidad no es un valor comprendido entre 0x0 y 0xF
InvalidDeviceType	400	Si el valor pasado en el cuerpo de la petición para el parámetro `type` no está comprendido entre 0 y 3
EmptyBody	400	Si se hace POST sin especificar nada en el cuerpo de la petición
CommandNotAllow	400	Si el parámetro `on` o el parámetro `state` no están dentro del rango de valores permitidos
DeviceNotFound	404	Si no existe ningún módulo X-10 con el código de casa y el código de unidad especificados
InvalidDevice	400	Si al crear un módulo, el cuerpo de la petición no tiene un formato JSON válido
ScenarioIdNotValid	400	Si el identificador de escenario no es un número entero
ScenarioNotFound	404	Si no existe ningún escenario con el identificador especificado
InvalidScenario	400	Si al crear un escenario, el cuerpo de la petición no tiene un formato JSON válido

Anexo II: Instalación, configuración y primeros pasos

Poner en marcha el proyecto puede resultar algo tedioso la primera vez. A continuación se presenta una lista detallada de los pasos que se deben seguir para poner el marcha el proyecto:

1. Instalar en Galileo la imagen IoT. Los pasos a seguir pueden encontrarse en la página web oficial de Intel.
2. Conectar la placa Galileo al router con un cable Ethernet.
3. Conectar la placa Galileo a una fuenta de alimentación.
4. Cargar mediante un cable USB el sketch get_galileo_address en Galileo para obtener la dirección IP que le ha sido asignada dentro de la red local (existen varias manera de averiguar la IP y todas de válidas), y dejarla anotada. Una vez averiguada la IP de Galileo desconectar la fuente de alimentación
5. Cargar el sketch X10_Ethernet en Arduino. Una vez cargado desconectar el cable USB.
6. Realizar las conexiones entre el PLC, Arduino y Galileo que se muestran en la figura 14.
7. Conectar nuevamente la placa Galileo a la alimentación.
8. Subir a Galileo el código del servidor (bien con el entorno de desarrollo Intel XDK bien via ssh) y ejecutarlo.

Una vez hecho todo esto, el siguiente paso será configurar la aplicación *MyDuino*. Para ello, desde la pantalla de preferencias se deben indicar el puerto y dirección IP de nuestro servidor. El puerto es el 3000 y la dirección IP será la que se haya averiguado en el paso 4 de la instalación.

Ahora, ya podemos empezar a programar y personalizar nuestro sistema domótico.

Anexo III: Manual de usuario

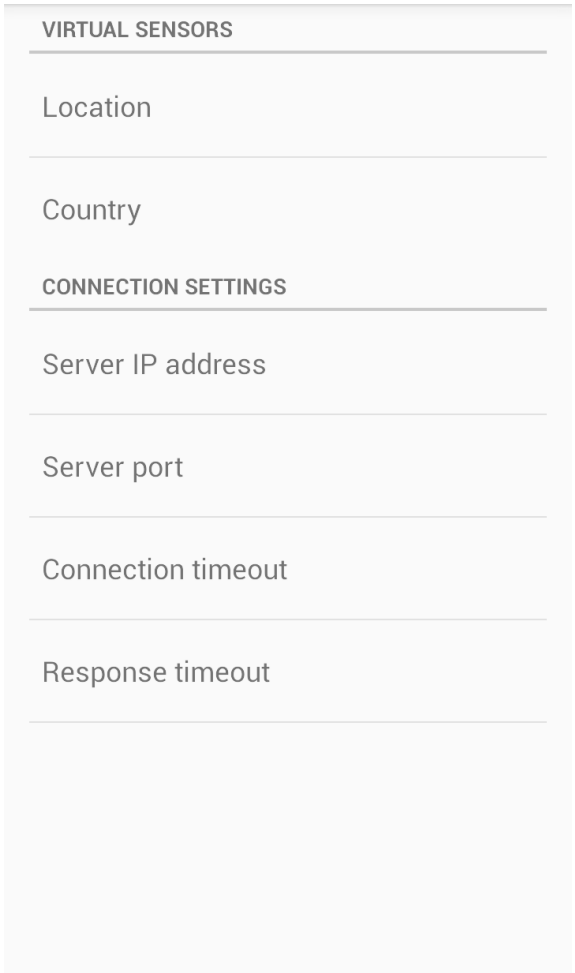
Primeros pasos

Lo primero que debe hacer el usuario una vez instalada la aplicación es configurarla con los datos de su servidor y vivienda. Para ello debe abrir la pantalla de «Preferencias de usuario» pulsando sobre el icono de la barra de navegación:

Pantalla de preferencias de usuario

Esta pantalla permite configurar la dirección IP y puerto del servidor que se encarga de gestionar el sistema domótico, así como los tiempos de conexión y respuesta que debe esperar la aplicación como máximo para obtener una respuesta de dicho servidor ante una petición.

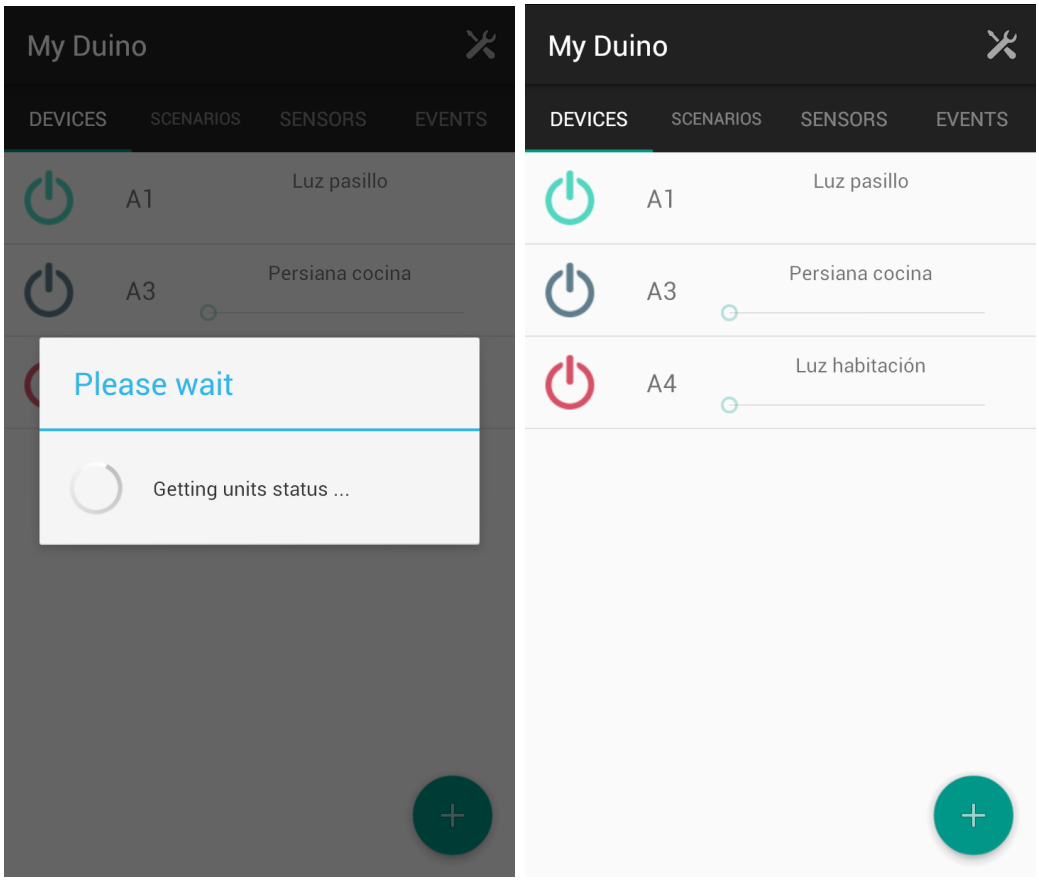
También desde aquí se puede indicar el país y la ciudad en la que se encuentra la vivienda en la que está instalado el sistema domótico para recoger la información ambiental en función de estos valores.



Lista de dispositivos

Lo primero que hará la aplicación nada más arrancar será consultar al servidor los estados de cada uno de los dispositivos que forman parte del sistema. Mientras dure esta operación se mostrará un mensaje informativo al usuario, quien no podrá realizar ninguna otra acción hasta que no acabe esta tarea. Una vez recogida toda la información se mostrará la pantalla «Lista de dispositivos».

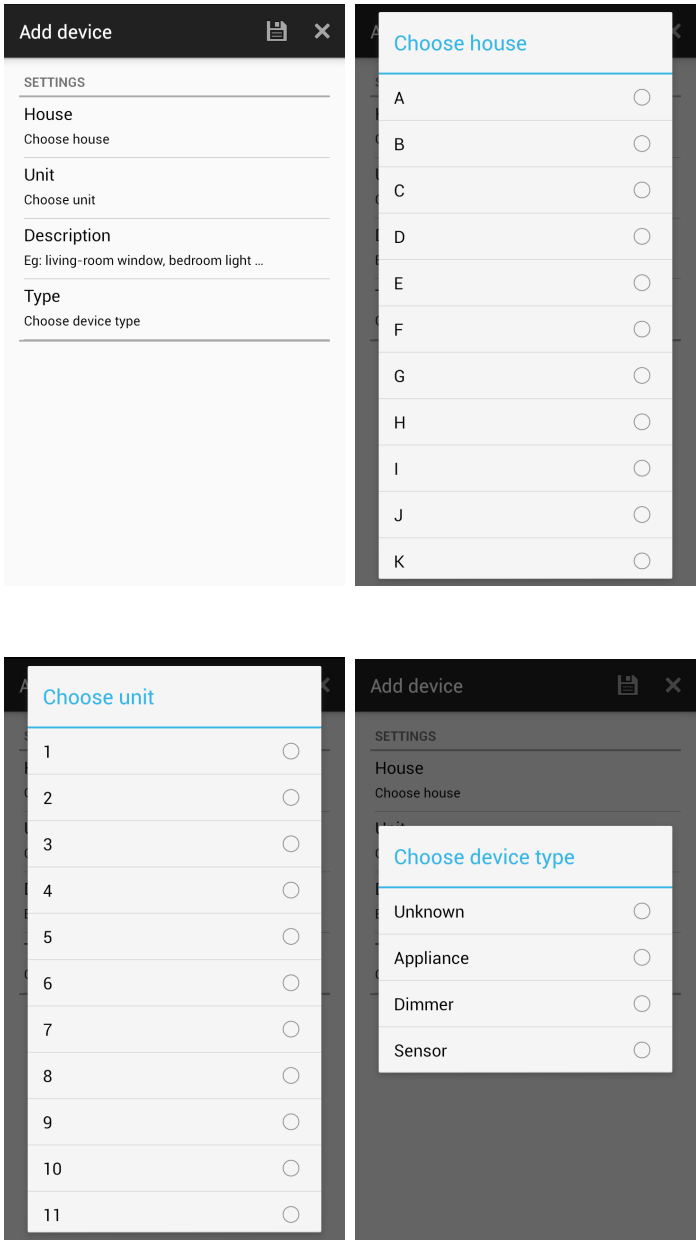
La pantalla «Lista de dispositivos» muestra una lista con todos los dispositivos presentes en el sistema X-10 que sean de tipo dispositivo, regulador o desconocido. Cada elemento de la lista contiene una imagen que indica, en función del color, si la unidad está encendida, apagada, o se desconoce su estado.



Crear un nuevo dispositivo

Al presionar sobre el botón con el icono + que aparece en la pantalla «Lista de dispositivos» se abre la pantalla «Nuevo dispositivo». En ella el usuario puede dar valor a los diferentes atributos que definen un dispositivo. Para ello ha de pulsar sobre el atributo al que desea dar valor con lo que aparecerá un cuadro de diálogo para poder escribir.

Si se presiona sobre el icono de guardar de la barra de navegación, el dispositivo se almacenará en el sistema y volverá a aparecer la lista de dispositivos (con el nuevo dispositivo incluido). Si se presionara el icono de cancelar se descartarían los cambios y se volvería a la pantalla «Lista de dispositivos».

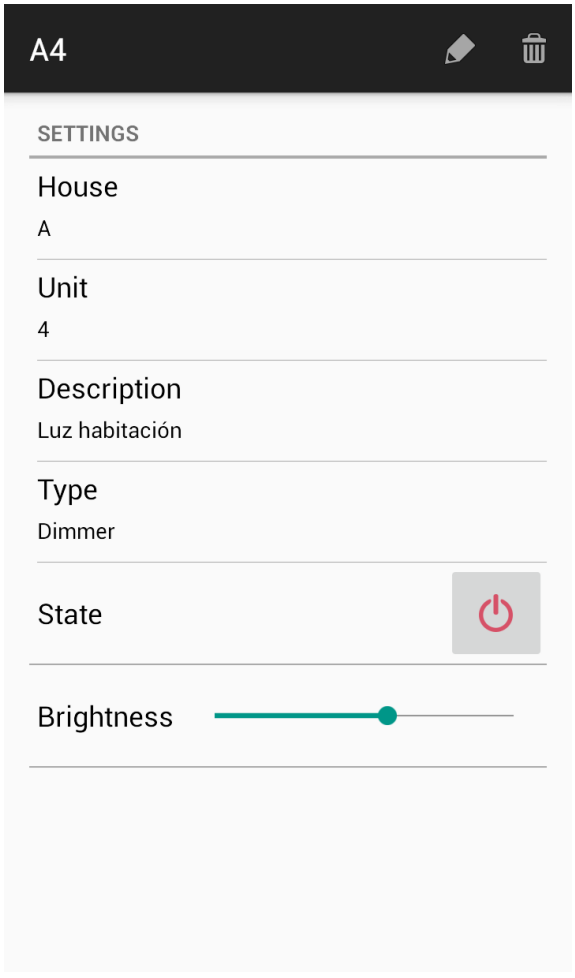


Editar, encender o apagar, reducir o aumentar la intensidad de un dispositivo

Al presionar sobre cualquier elemento de la lista de dispositivos aparece la pantalla «Detalle dispositivo» desde donde se puede consultar toda la información de dicho dispositivo.

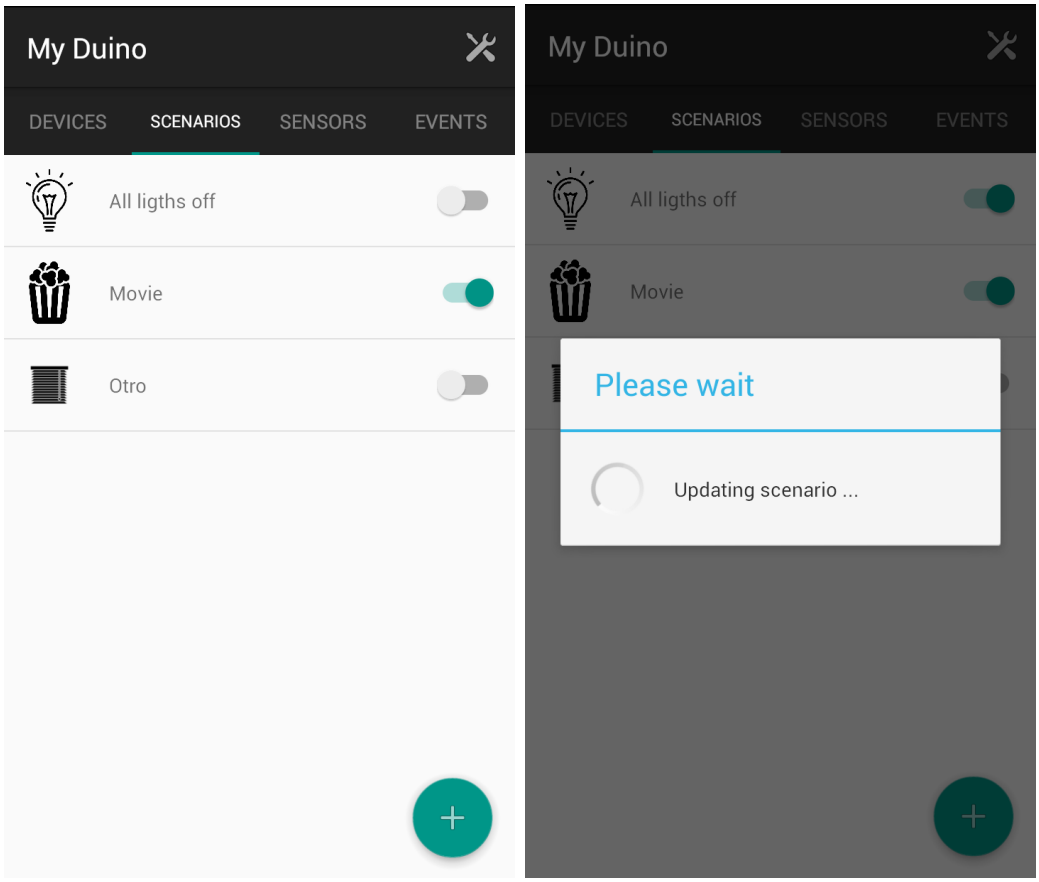
Desde esta misma pantalla también podemos:

1. Indicar que queremos apagar o encender el dispositivo
2. Aumentar o disminuir su intensidad (si es de tipo regulador)
3. Eliminarlo del sistema presionando sobre el icono de la papelera situado en la barra de navegación



Lista de escenarios

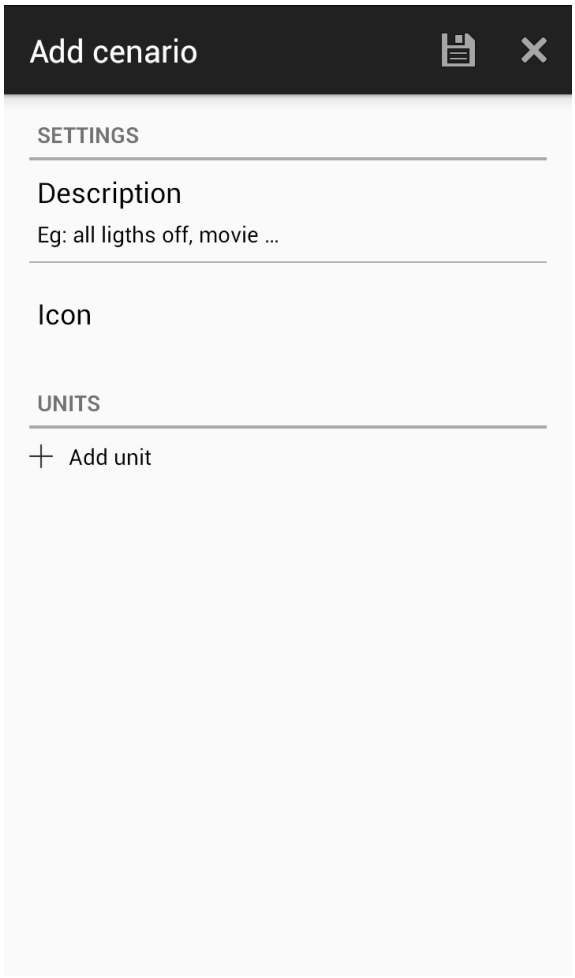
En esta pantalla se muestra una lista con todos los escenarios que han sido definidos por el usuario. Cada elemento de la lista contiene un *switch*, que indica si el escenario se encuentra en ese momento activo o no y, a su vez, permite activarlo o desactivarlo.



Crear un escenario

Al presionar sobre el botón con el icono + que aparece en la pantalla «Lista de escenarios» se abre la pantalla «Nuevo escenario». En ella el usuario puede dar valor a los diferentes atributos que definen un escenario.

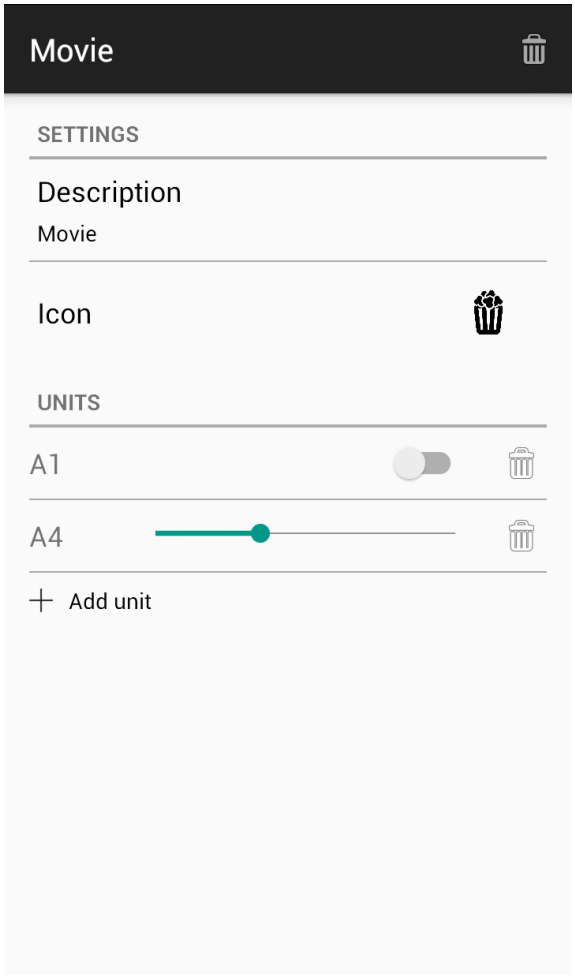
Si se presiona sobre el icono de guardar de la barra de navegación, la unidad se almacenará en el sistema y volverá a aparecer la lista de unidades (con la nueva unidad incluida). Si se presionara el icono de cancelar se descartarían los cambios y se volvería a la pantalla «Lista de unidades».



Editar un escenario

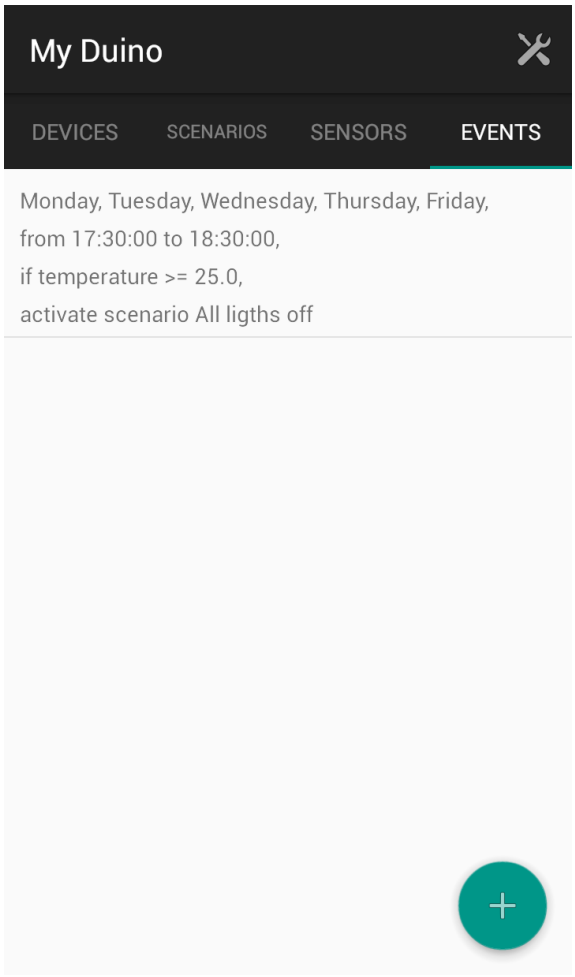
Al presionar sobre cualquier elemento de la lista de escenarios aparece la pantalla «Detalle escenario» desde donde se puede consultar toda la información asociada a dicho escenario.

Desde esta misma pantalla también podemos eliminar el escenario del sistema presionando el icono de la papelera que aparece en la barra de navegación.



Lista de eventos

En esta pantalla se muestra una lista con todos los eventos que han sido definidos por el usuario.



Crear un evento

Al presionar sobre el botón con el icono + que aparece en la pantalla «Lista de eventos» se abre la pantalla «Nuevo evento». En ella el usuario puede dar valor a los diferentes atributos que definen un evento.

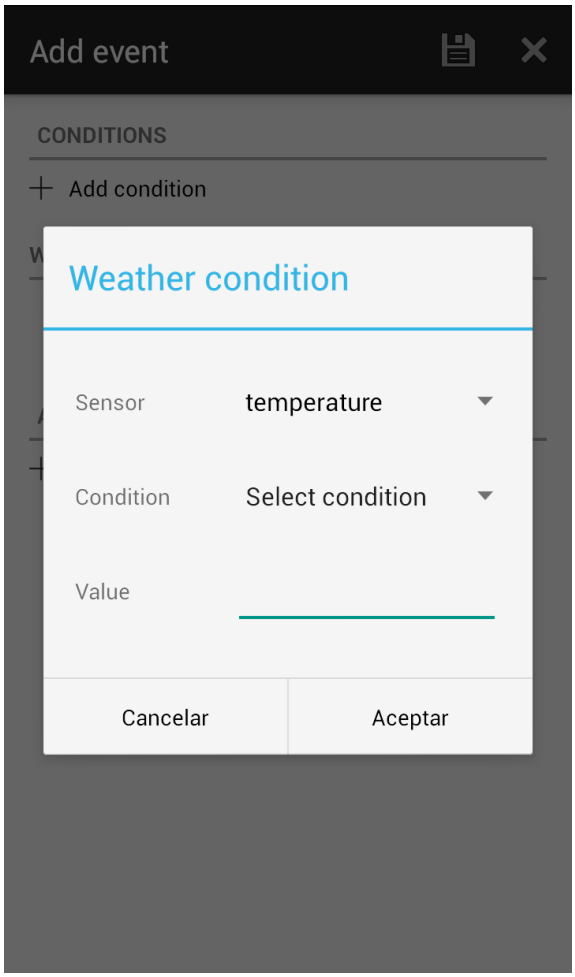
Si se presiona sobre el icono de guardar de la barra de navegación, el evento se almacenará en el sistema y volverá a aparecer la lista de eventos (con el nuevo evento incluido). Si se presionara el icono de cancelar se descartarian los cambios y se volvería a la pantalla «Lista de eventos».

Al pulsar sobre «Nueva condición» aparece un cuadro de diálogo preguntando al usuario si la condición que desea definir es ambiental o temporal. Al pulsar sobre «Nueva acción» aparece un cuadro de diálogo con una lista de todos los escenarios definidos en el sistema.

Añadir una condición ambiental a un evento

Si en el cuadro de diálogo «Nueva condición» el usuario presiona sobre «Condición ambiental» se abre un nuevo cuadro de diálogo que permite al usuario definir los atributos de la condición.

Tras rellenar los difrentes campos y pulsar en «Aceptar» la condición se añadirá a la lista de condiciones del evento actual.

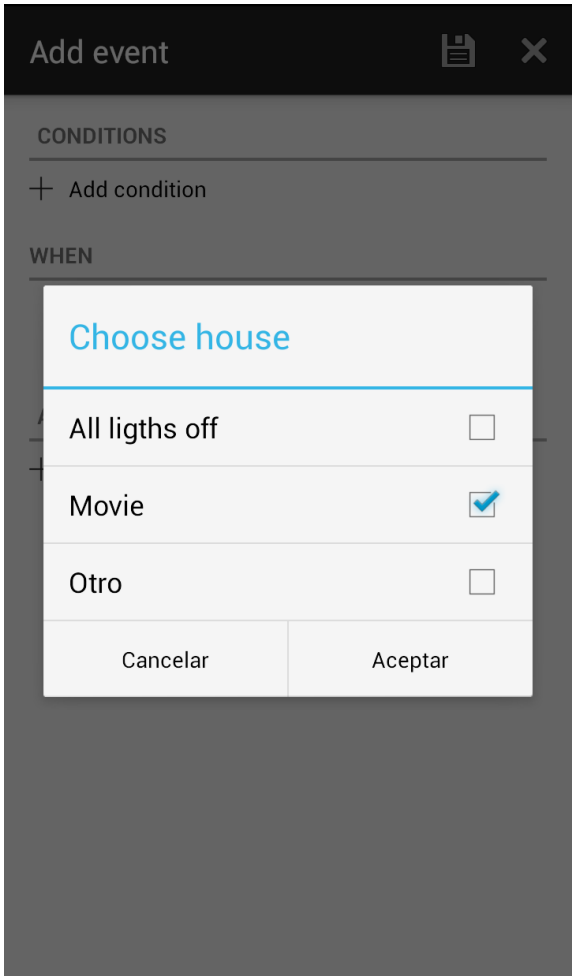


Añadir una condición de tiempo a un evento

Si en el cuadro de diálogo «Nueva condición» el usuario presiona sobre «Condición temporal» se abre un nuevo cuadro de diálogo que permite al usuario definir la hora de inicio y fin de dicha condición. Tras ello, la condición se añadirá a la lista de condiciones del evento actual.

Añadir una acción a un evento

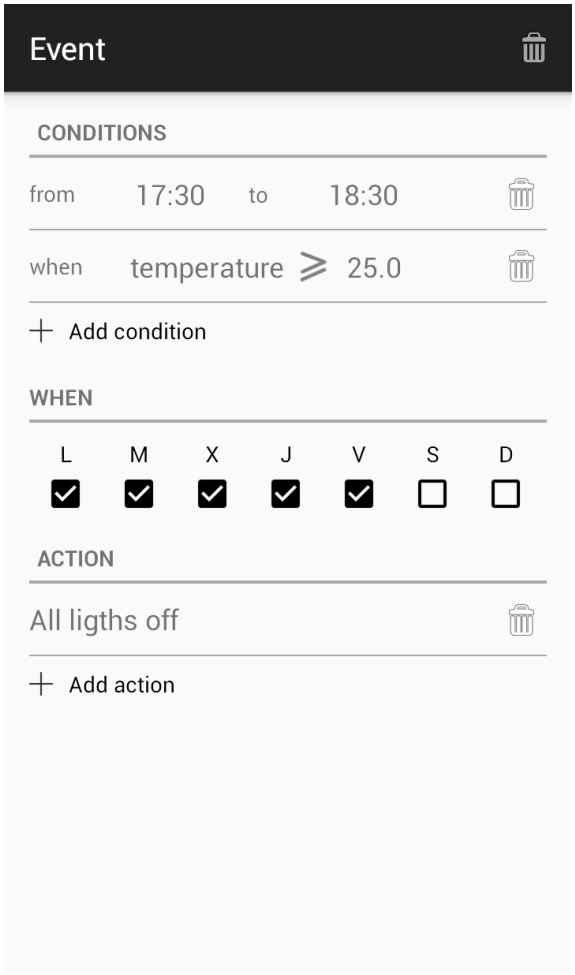
Al pulsar sobre un escenario este se añadirá a la lista de acciones del evento actual.



Editar un evento

Al presionar sobre cualquier elemento de la lista de eventos aparece la pantalla «Detalle evento» desde donde se puede consultar y modificar toda la información asociada a dicho evento.

Desde esta misma pantalla también podemos eliminar el evento del sistema presionando el icono de la papelera que aparece en la barra de navegación.



Lista de sensores

En esta pantalla esta formada por dos secciones. En la primera, se muestra una lista con todos los sensores físicos que han sido definidos por el usuario. Cada elemento de la lista contiene una imagen que indica, en función del color, si el sensor está activo o no, o si se desconoce su estado.

En la segunda sección, se muestra una lista con todos los sensores virtuales disponibles: temperatura, presión, humedad, viento, lluvia y nieve, así como la información ambiental relativa a cada sensor.

Al presionar sobre el botón con el icono + se abre la pantalla «Nuevo dispositivo».

Bibliografía

[1] Manuel Peña Alcaraz. Comunicaciones en el entorno doméstico (domótica). Comparación KNX - LONWorks. [En línea] http://www.sistemamid.com/panel/uploads/biblioteca/2014-06-14_07-47-08105068.pdf, 2012.

[2] Óscar Torrente Artero. *Arduino. Curso práctico de formación.* RC Libros, 2013.

[3] KNX Association. KNX Partners - List. [En línea] http://web.archive.org/web/20131231231228/http://www.knx.org/knx-partners/knxeib-partners/list.

[4] Massimo Banzi. *Introducción a Arduino.* Anaya, 2ª edition, 2012.

[5] Mike Cantelon, Marc Harter, TJ Holowaychuk, and Nathan Rajlich. *Node.js in action.* Manning, 2014.

[6] Kantar Worldpanel ComTech. *Smartphone OS sales market share.* [En línea] www.kantarworldpanel.com/global/smartphone-os-market-share, May 2016. 21

[7] Intel Corporation. Tutorial básico de Linux para la tarjeta Intel Galileo Gen 2. [En línea] https://engage.intel.com/community/es/galileo/centro-de-aprendizaje/video-tutoriales/, 2015.

[8] Comunidad de desarrolladores de Node. mraa module documentation. [En línea] https://www.npmjs.com/package/mraa.

[9] Asociación Española de Domótica e Inmótica. Qué es domótica. [En línea] http://www.cedom.es/sobre-domotica/que-es-domotica. 6

[10] Asociación Española de la domótica e inmótica CEDOM. Estudio de mercado 2012 - 2014. [En línea] http://www.cedom.es/sobre-domotica/publicaciones/cedom-estudio-de-mercado-2012-2014, 2015. 7

[11] ECMA International (Asociación europea para la estándarización de sistemas de información y comunicación). *Standard ECMA 404: The JSON Data Interchange Format.* [En línea] www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, October 2013. 18

[12] EuroX10. ¿Qué es X-10? [En línea] http://www.eurox10.com/Content/x10information.html.

[13] Jesús Tomás Gironés. *El gran libro de Android.* Marcombo, 5ª edition, 2016. 22

[14] Intel. Intel® Galileo Gen 2 Development Board. [En línea] http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-overview.html. 2

[15] Marco Lastri. *The dark side of porting Arduino sketches on Intel Galileo.* [En línea] https://garretlabs.wordpress.com/2014/03/18/the-dark-side-of-porting-arduino-sketches-on-intel-galileo-part-one, 2014.

[16] Marmitek. *X-10 protocol for the Marmitek XM10 OEM Controller.* [En línea] http://plantron.gr/sites/plantron.gr/files/pdf/xm10_manual.pdf, 2014. 44

[17] Thomas Mittet. *X10 RF & IR remote using Arduino.* [En línea] http://blog.lookout.no/2010/06/x10-plc-rf-ir-and-computer-interface.html, 2015. 2, 41

[18] Inc. MongoDB. Reinventando la gestión de datos. [En línea] https://www.mongodb.com/es, 2016.

[19] El Mundo. TECNOLOGÍA | Montaje de un hogar digital. [En línea] http://www.elmundo.es/elmundo/2010/03/10/suvivienda/1268237467.html, 2010. 5

[20] Fundación Node.js. API 4.x de Express. [En línea] http://expressjs.com/es/4x/api.html, 2014.

[21] Manoel Carlos Ramón. *Intel Galileo and Intel Galileo Gen 2. API features and Arduino projects for Linux programmers.* Apress open, 1ª edition, 2016.

[22] Jose Mª Maestre Torreblanca. *Domótica para ingenieros.* Ediciones Paraninfo, SA, 2015. 30