UNIVERSIDAD REY JUAN CARLOS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FINAL DE CARRERA

DISEÑO DE INTERFAZ WEB BAJO DJANGO PARA PLACAS ELECTRÓNICAS ARDUINO

AUTOR: ISRAEL DÍAZ DE LA VEGA

TUTOR: GREGORIO ROBLES MARTÍNEZ

9 de junio de 2013

Título: *DISEÑO DE INTERFAZ WEB BAJO DJANGO PARA PLACAS ELECTRÓNICAS ARDUINO.*

Autor: *ISRAEL DÍAZ DE LA VEGA*

Tutor: *GREGORIO ROBLES MARTÍNEZ*

La defensa del presente Proyecto Fin de Carrera se realizó el día 19 de Enero de 2005; siendo calificada por el siguiente tribunal:

Presidente:

Secretario

Vocal

Habiendo obtenido la siguiente calificación:

Calificación:

Presidente	**Secretario**	**Vocal**

Agradecimientos

En primer lugar quisiera agradecer a Gregorio Robles la oportunidad que me ha brindado para realizar este proyecto y aprender de él, así como su labor para corregir y mejorar esta memoria.

A mis compañeros en Flir, aunque no haya estado mucho tiempo con vosotros me habéis dado la oportunidad de dar el salto al mundo laboral, aprender mucho y encontrarme con un gran ambiente de trabajo.

A mis padres, que siempre han estado ahí y sé que siempre lo van a estar; por apoyarme, confiar en mí y por intentar aprender y conocer qué es lo que estudia un "Teleco".

Agradecer a mis compañeros de clase, sois muchos y muy buenos amigos y no quiero dejar a ninguno sin mencionar: Abel, Javi, Juli, Luis, Poker, Helen, Naza, Adri, Guille, Berty, Bernard, Fran y Muerte. Y en definitiva a toda aquella gente con la que compartido momentos dentro de la universidad y fuera de ella, espero que os haya echo disfrutar lo mismo que yo he disfrutado con vosotros.

A ti Sélica, porque tener tu cariño siempre cerca es muy importante para mí.

A mis amigos de siempre, o como dice mi padre a la tropa: Adri, Manu, Jote, Jesús, Hermo y Yepes. Gracias por ser como sois, por transformar siempre los malos momentos en buenos y por que no me imagino un futuro sin vosotros(y unas cañas por medio).

A todos, gracias de verdad.

Cuando lo hayas encontrado, anótalo.

Charles John Huffam Dickens

When you make the finding yourself

– even if you're the last person on Earth to see the light–

you'll never forget it

Carl Sagan

Resumen

El presente proyecto abarca 2 mundos diferenciados en su forma pero muy parecidos en espíritu. Son el Hardware y el Software Libre. El hardware representado bajo la plataforma Arduino y el software por el framework-web Django. Más conocido para el gran público es el término software libre, sin embargo, los proyectos bajo el lema "Hazlo tú mismo" (DIY , del inglés *Do It Yourself*) están ganando mucho protagonismo. Son proyectos sencillos, sumamente baratos y entretenidos que animan al usuario básico a adentrarse en el mundo de la electrónica.

Según la Free Software Foundation, el software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, y estudiar el mismo, e incluso modificar el software y distribuirlo modificado. En el caso del hardware libre, el concepto libre se refiere a las especificaciones y diagramas esquemáticos del hardware que son de acceso público.

Aunando estos dos conceptos, el objetivo del proyecto consiste en la utilización de una placa Arduino conectada a un ordenador donde está corriendo un Django, que actúa como servidor. Para el usuario novato, la programación de Arduino puede ser bastante engorrosa ya que requiere de conocimientos de programación. El sistema integrado en Django permite, a través de una sencilla web, la configuración de los distintos sensores conectados a la placa, así como visualizar sus valores remotamente. Es decir, el objetivo final es la implantación de una interfaz gráfica para la placa Arduino en Django donde el usuario podrá indicar qué sensores (temperatura, humedad, luminosidad,...) dispone y qué hacer con ellos (alarmas, actuadores, notificación de valores, gráficas,...).

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

INTRODUCCIÓN

Este capítulo marca el comienzo de la memoria de este proyecto fin de carrera. Los apartados siguientes intentan dar una pequeña introducción acerca del contexto a tratar y algunos objetivos que se han marcado para el proyecto. En un último apartado se explica la estructura de esta memoria.

1.1. Contexto y descripción del problema

Realizando una búsqueda rápida con las palabras clave "Arduino + proyecto", la mayoría de los resultados tienen una característica común: son proyectos muy relacionados con la domótica. Básicamente un sistema montado con Arduino dispone de tres elementos claves: la placa Arduino, un conjunto de sensores y un dispositivo donde poder configurar la placa Arduino (comúnmente un ordenador).

Conectar los sensores a la placa conlleva algunos conocimientos de electrónica; son en su mayoría sensores muy simples pero a la vez muy útiles. Tienen la capacidad de obtener valores de diversas magnitudes; como temperatura, humedad y luminosidad entre otros. Es decir, magnitudes indispensables para un sistema domótico.

Por tanto, queda la última parte, la configuración de la placa Arduino para que obtenga el valor de los sensores conectados previamente. Esta parte se antoja la más complicada para el usuario medio ya que se necesitan conocimientos de programación. Es en esta parte donde el presente proyecto tiene su razón de ser, ya que una vez montado el sistema y conectada la placa al dispositivo que funciona como servidor, el usuario sólo tendrá que configurar los sensores conectados a través de una interfaz web sencilla. Además, el uso del servidor tiene algunas ventajas considerables para el sistema domótico, ya que se pueden configurar alarmas

para que el sistema sea de alguna forma autosuficiente. Por otro lado, permite acceder a la información de los valores remotamente al tratarse de un server con Django, asimismo realizar una acción desde cualquier punto, siendo un sistema domótico en toda regla.

1.2. Objetivos

Echando un vistazo al "problema" a tratar, es posible que no queden claros los verdaderos objetivos del proyecto. Sí, es cierto que se trata de un proyecto sobre Arduino y Django; que intenta desarrollar una interfaz web para poder controlar este tipo de placas pero ¿cuáles son los verdaderos objetivos?

1. Creación de una interfaz web para el control de placas Arduino.

   El objetivo más claro y el que además da nombre al proyecto por ser el más importante y para el que el proyecto fue principalmente diseñado.

2. Profundizar en desarrollos con Django.

   Durante la carrera Django fue protagonista en la asignatura SAT (Servicios y Aplicaciones Telemáticas), además, es una asignatura impartida por el tutor de este proyecto: Gregorio. El interés despertado en la asignatura y la popularidad que va ganando día a día convierten este objetivo en uno de los más útiles para el autor.

3. Familiarizarse con Arduino.

   El uso en el proyecto de Arduino permite profundizar en dos campos aprendidos en la ingeniería: programación y electrónica. No debe suponerse que la electrónica en este proyecto es meramente testimonial, al contrario, mucho del diseño del código tiene en cuenta las capacidades que ofrecen estas placas.

4. Diseño de la web.

   La web donde se produce toda la "acción" del proyecto tiene que ser sencilla y útil. En el apartado propio sobre el diseño se explicará este objetivo más detalladamente.

5. Obtener alguna aplicación práctica.

   Objetivo muy importante debido a la naturaleza práctica del proyecto, en diversos puntos de la memoria se comentarán las aplicaciones ideadas. Se verán los resultados en el apartado correspondiente a las pruebas.

6. Aprender.

   Este proyecto intenta simplemente eso, dar al autor una forma de enriquecer sus conocimientos acerca de estas tecnologías.

En el apartado final Conclusiones, se intentará hacer un repaso acerca de estos objetivos y comprobar hasta que punto ha sido posible completarlos.

1.3. Desarrollo del Proyecto

En este apartado se verá fase por fase el desarrollo a lo largo del tiempo de este proyecto, desde la elección del mismo hasta la redacción de esta memoria.

1. Elección de PFC

   Desde un principio se quiso realizar un proyecto orientado hacia el mundo libre, un proyecto para aprender nuevas tecnologías y utilizar aquellas que despertaban curiosidad acerca de las posibilidades que ofrecían. Muy posiblemente, las prácticas en la empresa Flir [3] trajeron consigo alguna de las ideas que se plasmaron en el código, como pueden ser las funciones de alarma o el uso de sensores de temperatura. Flir es una empresa dedicada a la manufactura de dispositivos de imagen térmica e infrarrojos con contratos en seguridad, industria y tecnología de imagen; tanto para fines comerciales como militares.

   Con todo ello, se decidió en un inició crear una aplicación en Django que en un principio mostrara información sobre la placa Arduino. La parte web se quiso desarrollar debido al interés y las posibilidades laborales que ofrece.

2. Selección de componentes

   El comienzo del proyecto tuvo como protagonista a la electrónica, se buscó información por doquier, consultando tutoriales, búsqueda de tiendas, etc. Esta labor de investigación se ve reflejada en el apartado de la memoria que describe los sensores utilizados y placa elegida. Destacar que los precios más bajos en este tipo de componentes se consiguen en tiendas chinas, el problema de este tipo de tiendas es la duración del envío que en algunos casos puede llegar al mes.

3. Inicio con Arduino

   Con todos los componentes requeridos llegados desde China, comenzaron las primeras pruebas con sencillos sketches para familiarizarse con Arduino, el funcionamiento del puerto serie y el comportamiento de los sensores.

4. Python + Arduino

   Antes de iniciar el desarrollo de la aplicación en Django se tenía que comprobar el funcionamiento del protocolo Firmata. Su uso y entendimiento se llevo a cabo a partir de pequeños programas, de nuevo, los tutoriales encontrados por la red fueron indispensables, entre los pequeños logros conseguidos en esta fase del proyecto se encuentran el encendido de un LED o mostrar la temperatura actual a través de una shell.

5. Django

   Dejando de lado por el momento la tecnología Arduino, debía tomar partido el desarrollo de la aplicación web. En este periodo del desarrollo se diseñaron la base de datos y su jerarquía. Primero con la implementación para placas, sensores y sus correspondientes vistas. Una vez alcanzado este estado se procedió al diseño del controlador y de todo el sistema de triggers y actuadores llevándose consigo una gran parte de tiempo debido a su complejidad. A partir de aquí se desarrolló el programa demonio junto con la conexión a Xively para poder tener un control de lo que se estaba midiendo. Fue un código con bastantes rectificaciones hasta que fue válido para probar en el laboratorio de la universidad.

6. Prueba Laboratorio

   Una explicación detallada sobre las pruebas puede encontrarse en el apartado de la memoria con el mismo nombre. Básicamente se llevo al laboratorio un código ya plenamente funcional. Pese a los numerosos problemas ajenos, las pruebas trajeron consigo una protección y alguna corrección de pequeños errores.

7. Interfaz

   El código que se probó en el laboratorio carecía de una cuidada interfaz a nivel visual. Simplemente las vistas de Django se mostraban en HTML plano y con tablas. Pero una aplicación de estas características necesita que el aspecto visual sea acorde a lo que está ofreciendo. Sin embargo, no había mucho tiempo disponible para investigar sobre cómo trazar la interfaz web. Una alternativa disponible era Bootstrap, a la postre la tecnología utilizada, que siendo un framework facilitaba en mucho (y al final ha sido así) la labor del programador web, creando una web de aspecto muy profesional con el poco tiempo que se disponía.

1.4. Estructura de la memoria

La memoria consta de 4 capítulos y un apéndice. Este primer capítulo introduce el proyecto a tratar. Los capítulos 2 y 3 conforman el grueso de la memoria, Estado del Arte (Capítulo 2) -donde se recogen las tecnologías involucradas y la posición de este proyecto dentro del arte- y Estructura e Implementación (Capítulo 3) que resume el desarrollo de todo el proyecto y como se ha ido gestionando desde un principio. A continuación, las Pruebas y Resultados (Capítulo 4) ayudan a comprobar el buen funcionamiento del proyecto. Por último, las Conclusiones (Capítulo 5) se centran en mostrar las posibles mejoras del proyecto y un presupuesto final.

Capítulo 2

ESTADO DEL ARTE

Bajo el siguiente capítulo se hace un repaso por el contexto donde se enmarca el proyecto, es decir, el estado del arte o la técnica. Se repasan las tecnologías utilizadas y los diferentes proyectos que están relacionados con la aplicación del Proyecto Fin de Carrera. Tampoco pueden faltar en este estado del arte los conceptos ,muy repetidos a lo largo de la memoria, de software y hardware libre, así como, la definición de domótica.

2.1. Mundo Libre

Como ya se ha mencionado anteriormente en el resumen del proyecto, el hardware y el software libre reúnen características muy similares pese a las diferencias obvias que se aprecian a simple vista. Dos mundos que tienen como objetivo principal llegar al mayor público posible, de forma que el mismo público pueda colaborar e incluso modificar el proyecto libre al que han accedido para así mejorarlo y volver a distribuirlo. A continuación se realizará un breve recorrido sobre estos dos mundos.

2.1.1. Software Libre

Para que un programa pueda considerarse libre, tiene que cumplir una serie de requisitos. Requisitos que se resumen en lo siguiente: libertad de usar el programa con cualquier fin, sin necesidad de comunicarlo a los desarrolladores; libertad de estudiar el código fuente del programa y modificarlo adaptándolo a nuestras necesidades, sin necesidad de hacer públicas las modificaciones; libertad de distribuir copias, tanto binarios como código fuente, modificadas o no, gratis o cobrando por su distribución; libertad de modificar el programa y publicar las mejoras para beneficio de la comunidad. Aunque el término libre suele estar asociado a gratis,

esto no tiene por qué ser así, se verá más claramente con el hardware libre. Algunos proyectos más conocidos del software libre son:

- Mozilla Firefox: es un navegador web libre coordinado por la Corporación Mozilla. Desarrollado para varias plataformas y con un código donde se implementan actuales y futuros estándares web. Destacable su cuota de mercado, a día de hoy el tercero tras Chrome e Internet Explorer pero fue el primer navegador que pudo hacerle frente al navegador de Microsoft en términos de uso. `http://www.mozilla.org/`

- GIMP (GNU Image Manipulaton Program / Programa GNU para el Manejo de Imágenes): como su propio nombre índica, se trata de una verdadera alternativa libre al famoso software PhotoShop. `http://www.gimp.org/`

- OpenOffice: es una suite offimatica que incluye procesador de textos, hoja de cálculo , presentaciones o bases de datos. Mantenida actualmente por la Apache Software Foundation. Es considerada por muchos la gran alternativa a Microsoft Office. `http://www.openoffice.org/es/`.

- VLC media player: es un reproductor multiplataforma que admite la mayoría de formatos multimedia, así como diversos protocolos de transmisión. `http://www.videolan.org/vlc/`

Mención aparte merece Linux, el núcleo libre de sistema operativo basado en Unix. Normalmente se utiliza junto a un paquete de software dando lugar a lo que se conoce como distribución de Linux o "distro". Presente en numerosos sistemas y mantenido por un numeroso grupo de programadores es el mejor ejemplo disponible de software libre.

2.1.2. Hardware Libre

Retomando lo mencionado en el resumen del proyecto, la andadura del hardware libre es muy corta comparada con el software. Por tanto, aún no existe una definición exacta del hardware libre. El término libre implica que la información sobre el hardware es fácilmente accesible. Es decir, tanto el diseño del hardware (esquemáticos, listado de componentes, código fuente HDL,...) como el software que controla este hardware están distribuidos bajo los términos del movimiento libre. Con la aparición y el crecimiento de los dispositivos de lógica progamable, compartir el código HDL es mucho más sencillo que distribuir los esquemáticos. A su vez, estos códigos HDL pueden ser fácilmente instalados en SoCs (System-on-a-chip) o placas FPGAs

(field-programmable gate array). Se ofrece un listado con proyectos interesantes (Arduino tendrá su sección propia) del hardware libre:

- Raspberry Pi: uno de los proyectos que más aceptación está teniendo. Consiste en un pequeño ordenador del tamaño de una tarjeta de crédito. Ideado por la Raspberry Pi Foundation con la intención de dar a conocer la informática en las escuelas, sin embargo, es una máquina muy versátil, puede ser utilizada como servidor web, servidor de descarga, sistemas HTPC (*Home Theater Personal Computer*) y robótica, siendo además muy compatible con Arduino. Los esquemáticos de la placa son de acceso público. Una vista de el aspecto de la placa puede verse en la figura 2.3.

- Makey-Makey: curioso proyecto donde los haya, partiendo de una placa sencilla que tiene un aspecto similar al mando de la primera videoconsola de Nintendo, se conectan unos cocodrilos (los electrónicos) para hacer que cualquier objeto se transforme en un tecla (key), por ello el nombre de Makey-Makey (crear teclas).

- Qbo: el robot de código abierto y con espiritú Arduino. Desarrollado por la española The Corpora, fruto de 6 años de desarrollo permite tener un pequeño robot a modo de mascota por un precio bastante competitivo. Cuenta con sensores y sistemas reconocimiento de caras, voces y objetos. Desarrollado íntegramente en España [4], la versión final del robot se ofrece en la figura 2.2.

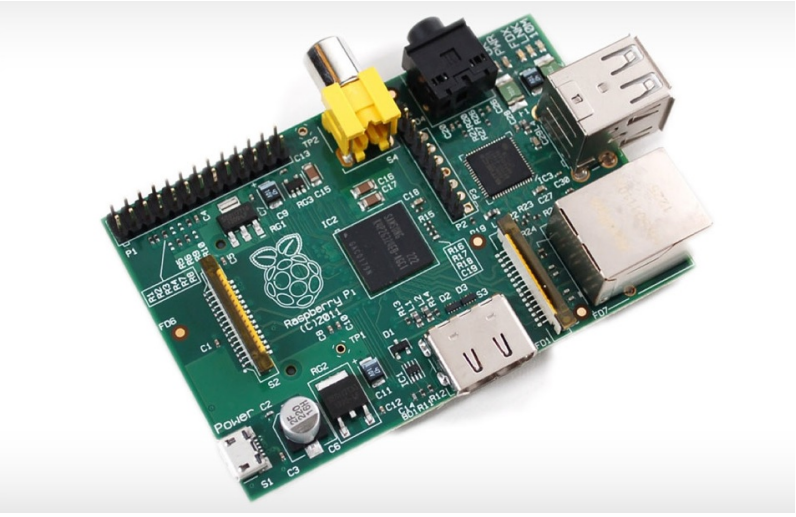


Figura 2.1: *Placa Raspberry PI. Actualmente goza de mucha popularidad.*

Figura 2.2: *Aspecto final del robot Qbo, su electrónica se basa en Arduino.*

Problemática actual

Pese al empuje que está teniendo esta disciplina, se encuentran una serie de problemas que dificultan mucho la consecución de forma práctica de muchos de los proyectos relacionados con el hardware libre[5]. Son problemas que no aparecen en la disciplina del software y son la verdadera causa de que el hardware libre no tenga tanta presencia actualmente. Se mencionan a continuación algunos de ellos.

- **Dependencia tecnológica extranjera de los componentes**

  Al intentar fabricar un diseño, es posible encontrarse con el problema de la falta de material. En un país puede no darse este problema, pero en otros puede que no se encuentren los materiales necesarios y está demostrado que las TIC, son herramientas indispensables para el desarrollo de las naciones por lo cual es de vital importancia a la vez que estratégica el que cada nación no dependa de otra para su desarrollo tecnológico.

- **Altos costos de producción**

  La persona que desea utilizar el hardware que un tercero ha diseñado, primero lo tiene que fabricar, para lo cual tendrá que comprobar los componentes necesarios, construir el diseño y verificar que se ha hecho correctamente. Todo esto tiene un costo.

- **El conocimiento lo poseen pocas empresas**

  Se sigue reteniendo el conocimiento en las grandes industrias productoras; como resultado el consumidor del producto tiene que adecuarse al producto que ofrece el mercado que es por lo general un producto genérico que no cumple con las necesidades muy específicas de un determinado consumidor; allí es cuando se ata, a las decisiones de las empresas productoras, al usuario y no se le da la libertad de elegir.

- **Gran inversión de tiempo en trabajos de diseño redundantes**

  Tanto en el hardware como en el software propietario existe mucho diseño redundante, es decir, se "reinventa la rueda" en vez de usar ese conocimiento previo e innovar en nuevas áreas de investigación y producción.

2.2. Selección de Tecnologías

Muchas de la tecnologías que tienen aquí cabida son muy conocidas, si bien otras, bien por su novedad o por su especificidad, no lo son tanto. Por ello en este apartado se analizan con profundidad todas ellas.

2.2.1. Arduino

La plataforma Arduino nace con un objetivo muy especifico, permitir que el uso de la electrónica sea más accesible [6]. Arduino es una plataforma de electrónica abierta para la creación de prototipos, basada software y hardware flexibles y fáciles de usar. Toda placa Arduino permite la conexión, a través de sus pines de entrada, de toda una gama de sensores, a su vez, con los pines de salida controlar luces, motores y otros actuadores. Inició su andadura en Ivrea, Italia; recibe su nombre del marqués de Ivrea Arduino I que a la postre fue rey de Italia durante las primeras décadas del siglo XI.

Hardware

Al tratarse de una plataforma abierta, los diseños se encuentran disponibles para el público. Esto permite modificar los diseños para adecuarlos a proyectos específicos (un buen ejemplo es la placa FreeDuino [7]) si así es necesario, además, al tener el listado de componentes pueden ser construidas a mano o por el contrario, compradas directamente de fábrica. La mayoría de las placas oficiales han sido desarrolladas por la empresa italiana Smart Projects aunque existen algunos modelos que tienen su origen en la empresa americana SparkFun Electronics. Debido

a la naturaleza del hardware libre, existen otras muchas compañías que también fabrican las placas Arduino, sin embargo, al comprar una placa oficial se contribuye al desarrollo del proyecto Arduino.

Pese a la variedad de modelos, las placas Arduino presentan una arquitectura muy similar. Básicamente son un microcontrolador Atmel AVR y puertos de entrada/salida. Los pines se disponen de forma estándar, siendo esto una de las características más importantes de Arduino ya que permiten la conexión de nuevos módulos intercambiables que añaden más características al sistema. Estos módulos reciben el nombre de *shields*.

Software

Para la programación de la placa Arduino se necesita de un software y un lenguaje de programación[?]. A continuación se describen los elementos clave del software.

- **Arduino IDE**

  Es un software que se encuentra disponible de forma gratuita en la web de Arduino, ofrece compatibilidad con todas las placas Arduino oficiales. Su código deriva de los proyectos Processing y Wiring (de la que por cierto Arduino es un fork). Es un programa multiplataforma y está escrito en java. Como su nombre indica, el IDE (del inglés *Integrated development environment*) permite a los usuarios crear programas de forma sencilla y dispone de herramientas comunes en este tipo de software como identación automática, colorido en palabras clave y búsqueda de paréntesis; permite la compilación del código implementado (usando la herramienta AVR Libc), así como la carga del mismo en la placa (utilizando avrdude). La figura 2.3 muestra el aspecto del programa.

- **Lenguaje de programación**

  Un programa escrito para Arduino recibe el nombre de Sketch. Para su programación es necesario tener conocimientos de C o C++ ya que son los lenguajes reconocidos por el compilador. Como se verá más adelante, esta última afirmación no es del todo correcta, siendo este proyecto un ejemplo de ello, ya que cualquier lenguaje que permita escribir en el puerto serie podrá comunicarse con Arduino. Por otro lado, el software Arduino IDE lleva incorporado una librería derivada del proyecto Wiring, que permite simplificar el código relacionado con entradas y salidas. Un sketch tiene dos funciones clave

  1. setup():

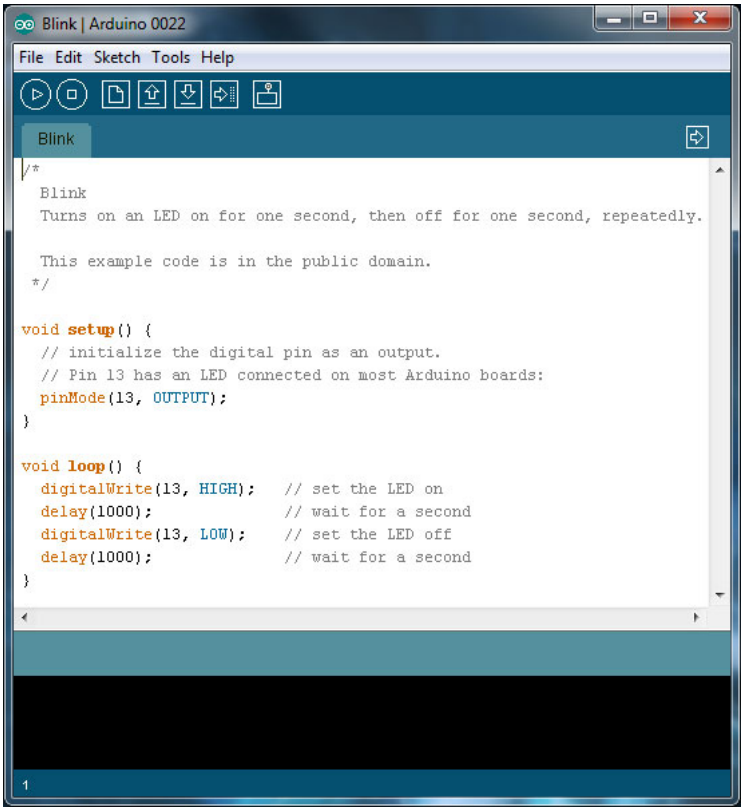


Figura 2.3: *Vista principal del software Arduino IDE con el sketch Blink*

Función que sólo se ejecuta una vez, sirve para inicializar las variables y la configuración de la placa (por ejemplo baudios del puerto serie).

2. loop():

   Es el cuerpo principal del sketch. Se ejecuta continuamente hasta que se apague la placa.

El sketch de iniciación a Arduino y que puede considerarse el Hola-Mundo de la electrónica es el *Blink*. Consiste en hacer lucir intermitentemente un led de la placa. A continuación se muestra el sketch *Blink*.

```
#define LED_PIN 13

void setup () {
 pinMode (LED_PIN, OUTPUT); // enable pin 13 for digital output
}

void loop () {
 digitalWrite (LED_PIN, HIGH); // turn on the LED
```

```
  delay (1000); // wait one second (1000 milliseconds)
  digitalWrite (LED_PIN, LOW); // turn off the LED
  delay (1000); // wait one second
}
```

2.2.2. Django

En el framework de desarrollo web Django [1], recae la otra parte importante del proyecto. Con un código abierto y escrito en Python, permite la creación de páginas web de forma mucho más sencilla. Su premisa básica es la reutilización y el "No te repitas"(DRY, del inglés Don't Repeat Yourself), además de una conectividad y extensibilidad de sus componentes.

Como muchos desarrollos abiertos, Django se utilizó en un principio en producción y después fue liberado. A partir de 2008, es mantenido por la Django Software Foundation (DSF).

Arquitectura

El desarrollo de Django sigue en cierta medida el Modelo Vista Controlador (MVC). El núcleo del framework consiste en un mapeador objeto-relacional el cual media entre los modelos de datos (implementados como clases de Python) y una base de datos relacional ("Model"); un sistema que atiende las peticiones con un conjunto de plantillas web ("Templates") y un despachador de URLs basado en expresiones regulares.

2.2.3. Boostrap

Para la interfaz de la web, se ha utilizado el framework Twitter Boostrap [11]. Es un conjunto de herramientas de acceso libre (de hecho, es el proyecto más popular dentro de GitHub) que permite la creación de páginas web. Contiene plantillas basadas en HTML y CSS, asimismo, incluye extensiones en JavaScript. Es compatible con la mayoría de navegadores y un soporte casi completo de HTML5 y CSS3. Por ejemplo, incluye bordes redondeados, gradientes y sombras; características propias de CSS3.

Boostrap fue desarrollado dentro de Twitter como una forma de conseguir consistencia entre los distintos equipos de trabajo que se dedicaban a la interfaz de Twitter. Debido a su inmediato éxito la mayoría de equipos de interfaz se alinearon con este framework. En agosto de 2011 fue liberado para el público.

Una de las características más importantes es la inclusión del "responsive design"que permite al diseño de la web ajustarse dinámicamente al tamaño de la pantalla del dispositivo donde se

visualice la web. Por otro lado, dispone de una serie de plugins JavaScript basados en jQuery. Permiten utilizar los típicos elementos de las interfaces de usuario, por ejemplo, tooltips, menús extensibles, scrolls, popovers, etc.

Es utilizado por la NASA y la cadena estadounidense MSNBC entre otros. Algunos ejemplos de su uso se pueden ver en la página Build with Boostrap.

2.2.4. Xively

Es un servicio online [9] que permite a los usuarios enviar los datos obtenidos por sus sensores para crear sus propias aplicaciones a partir de esos datos. Para el presente proyecto tiene la función de realizar las gráficas de las distintas magnitudes que miden los sensores conectados a la placa Arduino. Con el subtítulo que anuncia en su web *The Internet of Things* (Internet de las Cosas) hace referencia a un concepto de web futura en el que todo está conectado e identificado desde objetos cotidianos hasta incluso personas. De hecho, se define como una plataforma segura, escalable y cómoda que permite conectarte y construir el "Internet de las Cosas". Un esquema de este concepto se muestra en la figura 2.4.

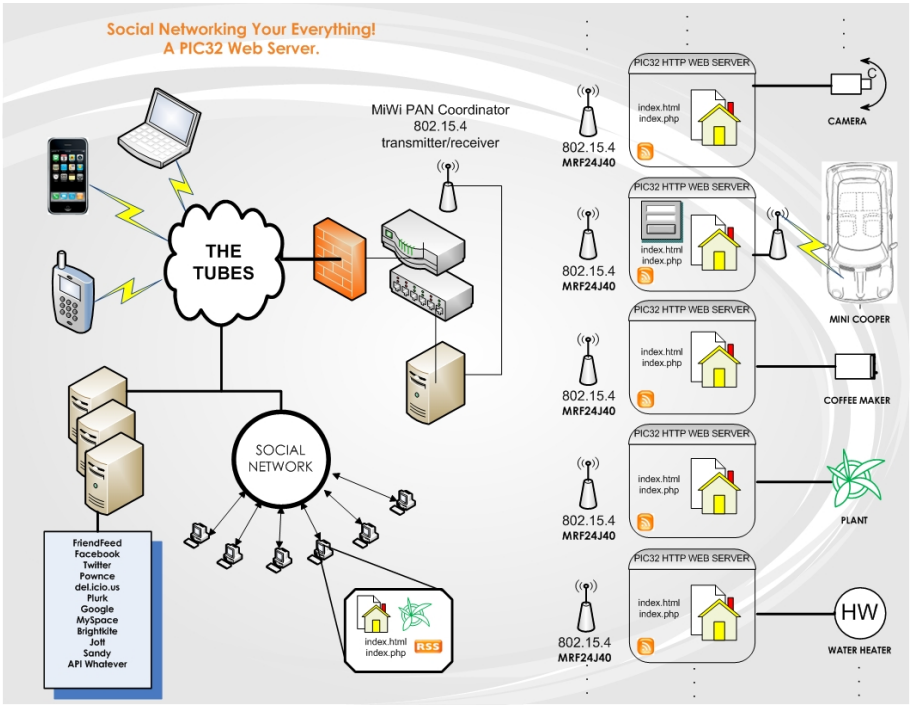


Figura 2.4: *Diagrama del concepto que define "Internet de las Cosas", donde todo está literalmente conectado*

Destacar que cuando se inició el proyecto el servicio recibía el nombre de Cosm, pero a partir de mayo del 2013 cambió su nombre a Xively. Este hecho ha provocado cambios en la API de Xively que se han propagado a la aplicación web. La API para utilizar el servicio está disponible en varios lenguajes (Java, JavaScript, C, Python, Objetive-C) y dispositivos (Android, ARM, Arduino). Para este caso, se ha utilizado la librería xively-python.

2.2.5. Python

Este lenguaje de programación, de alto nivel y propósito general, es utilizado en dos partes concretas de la aplicación. Se verá más concretamente en el apartado de la implementación del proyecto, pero tanto la aplicación web Django como el demonio que permite recoger los distintos valores de los sensores así como hacer funcionar los actuadores se encuentran escritos en Python.

Los códigos escritos en Python se caracterizan por su legibilidad y una menor cantidad de líneas de código que otros lenguajes de programación como C. Soporta múltiples paradigmas de programación, incluyendo orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Ideado por Guido van Rossum, su primera versión vio la luz en 1991. Pese a ser un lenguaje muy moderno y aún con poco recorrido en lo que antigüedad se refiere, Python tiene un amplia penetración de uso en distintas aplicaciones. Además, compañías y organizaciones como Google, Yahoo!, CERN o la NASA utilizan Python en sus desarrollos. Para constatar este hecho, el sitio PYPL PopularitY of Programming Language index ofrece la popularidad de los lenguajes de programación basándose en las búsquedas de tutoriales a estos lenguajes y la herramienta *Google Trends*. La figura 2.5 muestra la posición de Python y su evidente trayectoria ascendente.

Según la página web de Python [2] los principios diferenciadores de este lenguaje son:

- Sintaxis clara y legible.
- Fuerte capacidad de introspección en el funcionamiento del mismo.
- Orientación a objetos intuitiva.
- Expresión natural de código procedimental.
- Modularidad total, soportando jerarquía de paquetes.
- Tratamiento de errores mediante excepciones.

Position June 2013	Position June 2012	Delta in position	Programming language	Share in June 2013	Twelve month trends
1	1		Java	26.9 %	-0.9 %
2	2		PHP	14.3 %	-1.4 %
3	3		C#	10.4 %	+0.8 %
4	6	↑↑	Python	10.2 %	+2.1 %
5	4	↓	C++	9.4 %	+0.1 %
6	7	↑	Javascript	8.2 %	+1.0 %
7	5	↓↓	C	7.4 %	-1.3 %
8	8		Objective-C	5.0 %	+0.7 %
9	9		Visual Basic	3.4 %	-0.6 %
10	10		Ruby	2.9 %	-0.1 %
© 2013 Pierre Carbonnelle					

Figura 2.5: *Popularidad de los lenguajes de programación a junio de 2013. PYPL*

- Librerías estándar completas y módulos de terceros para prácticamente cualquier tarea.
- Fácilmente extensible mediante código en C/C++ u otros lenguajes.
- Empotrable en aplicaciones como una interfaz de scripting.

Utilidad de en el proyecto

Como recoge la web propia de Python, existen librerías y módulos de terceros para diversas funciones. Pues bien, este proyecto hace un uso intensivo de estas librerías; su incorporación en el desarrollo del proyecto ha ahorrado una gran cantidad de trabajo. Estas librerías son mencionadas a lo largo de la memoria debido a su uso recurrente en la implementación, a continuación se muestra una lista de ellas. La funcionalidad de cada una de ellas será detallada en el capítulo dedicado a la implementación del proyecto.

- pySerial
- pyDuino
- xively-python

La principal desventaja de Python es que la ejecución del código suele ser más lenta que en los lenguajes de programación compilados. En las magnitudes que trabaja el proyecto y por el tipo de aplicaciones para las que está diseñado esto no supone realmente ningún problema.

2.2.6. Sublime Text

Es un editor de texto y código multiplataforma [10]. Es el único programa utilizado para el desarrollo que no es libre ni tampoco es gratuito, se trata pues de un programa con software privativo. Afortunadamente, se puede obtener una licencia para su uso ilimitado, pero el no disponer de esta no genera ninguna limitación mas allá de una alerta cada cierto tiempo. Se encuentra actualmente en su versión 2.0.1.

Algunas de sus características son:

- Minimapa: consiste en una previsualización de la estructura del código, es muy útil para desplazarse por el archivo cuando se conoce bien la estructura de este.
- Soporte nativo para infinidad de lenguajes: Soporta de forma nativa infinidad de lenguajes de programación y texto plano.
- Syntax Highlight configurable: El remarcado de sintaxis es completamente configurable a través de archivos de configuración del usuario.
- Búsqueda Dinámica: Se puede hacer búsqueda de expresiones regulares o por archivos, proyectos, directorios, una conjunción de ellos o todo a la vez.
- Auto completado y marcado de llaves: Se puede ir a la llave que cierra o abre un bloque de una forma sencilla.
- Acceso rápido a linea o archivo: Se puede abrir un archivo utilizando el conjunto de teclas Cmd+P en Mac OS X o Ctrl+P en Windows y Linux y escribiendo el nombre del mismo o navegando por una lista. También se puede ir a una línea utilizando los dos puntos ":z el número de línea.
- Pestañas: Se pueden abrir varios documentos y organizarlos en pestañas.
- Resaltado de paréntesis e indentación: cuando el usuario coloca el cursor en un paréntesis, corchete o llave, los resalta y el paréntesis, corchete o llave de cierre correspondiente.

  Se ha elegido este programa debido a que es una de las herramientas más utilizadas y por haber trabajado anteriormente con ella.

2.2.7. LaTeX

LaTeX es un lenguaje marcado para la creación de documentos y un sistema para la preparación de documentos basado en la escritura TeX. No se debe confundir LaTeX con un programa,

es sólo el lenguaje con el que están escritos estos documentos, para poder escribir es necesario un editor de textos. En concreto para la realización de esta memoria se ha utilizado el editor Texmaker. Es muy utilizado en entornos académicos debido a la alta calidad de formato que pueden alcanzar los documentos TeX. Facilitan la elaboración de este tipo de documentos debido a la versatilidad de sus componentes. Para esta memoria son muy útiles su división por capítulos y secciones, el uso de la bibliografía, la inclusión de figuras y tablas, apéndices, etc.

2.3. Domótica

Como se ha mencionado anteriormente, la domótica es el nexo principal entre los diferentes proyectos con Arduino y a la postre es la utilidad que más salta a la vista al tratar con este tipo de sensores. Durante este apartado se explicará el concepto de domótica, incluyendo una breve historia y los elementos que intervienen.

2.3.1. ¿Qué es domótica?

Una definición válida para domótica [12] es la del conjunto de herramientas que permiten automatizar una vivienda. Automatizar una vivienda, las tareas domésticas y todas aquellas actividades que tienen lugar en una casa. Sin embargo, esta definición puede ampliarse a cualquier recinto cerrado que permiten un diseño inteligente con la aplicación de tecnologías.

El término domótica viene de la unión de las palabras domus (que significa casa en latín) y tica (de automática, palabra en griego, 'que funciona por sí sola').

2.3.2. Historia

La llegada de la electricidad a los hogares marca el inicio de la historia de la domótica, una idea que a principios del pasado siglo sólo tenía cabida en las novelas de ciencia ficción. Los primeros electrodomésticos aparecieron a entre 1915 y 1920, sin embargo, sólo estaban disponibles para las viviendas más acaudaladas debido a su alto precio.

Hasta mitad de siglo, las ideas de "casas futuristas" surgieron durante las exposiciones universales e incluso se mostraron algunos ejemplos de su uso. A pesar de su importancia, el acceso a casas domóticas sigue siendo terreno vetado, sólo al alcance de entusiastas de la electrónica o gente rica. Además, la no existencia de un protocolo sencillo y único desanima a los usuarios interesados en esta tecnología.

2.3.3. Descripción de un sistema domótico

Un sistema domótico esta formado por varios elementos esenciales, entre los que destacan:

- Sensores: pueden ser de muchos tipos y captar diferentes magnitudes como temperatura, humedad, luz, detectores de movimiento, etc.
- Controlador: es la unión entre las medidas recogidas por los diferentes sensores y los actuadores que permiten automatizar la vivienda. Esta función la puede realizar un ordenador personal o un controlador desarrollado para la aplicación domótica.
- Actuadores: se encargan de poner en funcionamiento las instrucciones que indica el controlador. Se basan en motores, valvulas, relé, LEDs, etc.

Estos elementos requieren una red para conectarse entre ellos, que puede ser por cable, inalámbrica, o tener un protocolo u otro. La cuestión del protocolo posiblemente sea la parte más comprometida al no existir un estándar.

Por otro lado, se necesita de una interfaz de usuario, que permite la observación de los valores y poner cotas al controlador. Esta interfaz puede visualizarse en varios dispositivos y día a día ganan más protagonismo las aplicaciones en tablets o teléfonos smartphone.

Figura 2.6: *Ejemplo de uso de interfaces táctiles para el control de una casa domótica.*

2.3.4. Aplicaciones Domóticas

Los ámbitos donde la domótica es útil pueden dividirse en:

Climatización

Permiten controlar el sistema de calefacción, aire acondicionado o ventilación. Se basan en sensores de humedad y temperatura que una vez conectados al controlador le permite poner en funcionamiento la climatización de la casa así como encender remotamente y con la intensidad deseada la calefacción o el aire acondicionado. Además, permite un ahorro energético considerable.

Luminosidad y comfort

Conectando la iluminación de la vivienda al entorno domótico posibilita el encendido y apagado autómatico mediante detectores de movimiento, por horario o de forma remota. El uso de leds RGB permiten modificar el ambiente de una sala y el uso de sensores de luminosidad proporcionan una posible interacción con cortinas o toldos.

Seguridad

En el ámbito de la seguridad la estos sistemas juegan un papel importante. Con la ayuda de cámaras, permiten monitorizar en todo momento lo que ocurre en la vivienda, gestionar el cierre de puertas y ventanas o la detección de incendios y fugas de gas o agua. Todo ello conectado a un sistema de alarmas que garantiza la seguridad completa del hogar.

Otras aplicaciones

También están disponibles:

- Riego automático.
- Alimentar mascotas de forma remota.
- Cuidado automático de piscinas.
- Teleasistencia médica.
- Jardinería.

2.4. Proyectos similares

Controlar una placa Arduino desde la web es el objetivo principal de este proyecto. Si bien un Arduino básico no dispone de una interfaz de red, existe un *shield* en el mercado que permite añadir una interfaz Ethernet a la placa. Con este *shield* es sencillo crear un simple modelo Servidor-Cliente, aunque, debido a la potencia que ofrece, no se puede esperar grandes funcionalidades desde la placa, podrá procesar pequeñas peticiones o enviar sus datos a través de una trama IP pero por cuestiones obvias un servidor Django no puede ser instalado en Arduino.

Por tanto, esta interfaz puede servir como una nueva forma de conexión con un servidor Django, a través de Ethernet. La mayoría de los proyectos encontrados se basan crear un servidor web en la propia placa y que esta reciba o envíe la información según se pida en cada momento. Por ejemplo el sketch RESTduino [14], provee a la placa de una interfaz REST a través de la shield Ethernet. Como ejemplo de uso para encender un led en el pin 9 bastará con acceder al recurso de a siguiente forma http://192.168.1.177/9/HIGH. De forma similar existe una pequeña API conocida como Teleduino que permite realizar acciones a través de un navegador

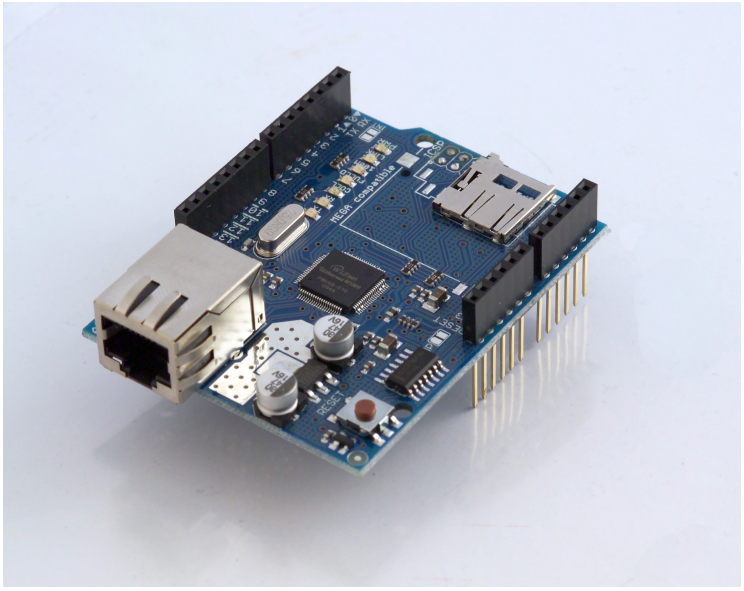


Figura 2.7: *Shield Ethernet para placa Arduino. Se observa la clavija RJ45 propia de Ethernet y un lector de tarjetas SD. La disposición de los pines aparece inalterada con respecto a la placa original.*

Son desarrollos útiles y que permiten simplificar el uso de Arduino pero no son suficientes para poder compararlos con este proyecto. Es fácil encontrar proyectos relacionados con Arduino

controlados desde una interfaz web. Son en su mayoría proyectos para conectar placa y servidor, realizados para un objetivo concreto y que no alcanzan la versatilidad de este proyecto que permite la conexión al vuelo de distintos sensores o circuitos y desde luego no disponen del control proporcionado por triggers y actuadores que se detallará más adelante.

Es más, realizando una búsqueda más intensiva en el la forja **GitHub** [13] con las palabras clave Arduino + Django sólo se encuentran disponibles 8 proyectos, donde ninguno de ellos destaca por ser algo más que pequeños ejemplos sobre cómo encender un LED o transmitir algún tipo de mensaje hacia el puerto serie de Arduino.

Encontrar una solución similar o completa parece que se antoja complicado. Existen soluciones que con algo más de desarrollo podrían llegar a las funcionalidades que ofrece este proyecto. Son dos mundos, Arduino y Django, que están en verdadero auge y dónde cada día se pueden ver nuevos proyectos que proponen nuevas soluciones.

2.5. Los sensores como negocio tecnológico

En este epígrafe de la memoria, la domótica ha ocupado un gran lugar como aplicación principal que tienen los sensores. Posiblemente sea la más conocida, pero en este apartado se pretende dar a conocer otras capacidades que presentan los sensores desde el punto de vista del negocio. Consiste en utilizar los sensores como una red de dispositivos conectados a Internet, permitiendo que monitorizen cualquier actividad ya sea humana o física. Facturando ya centenares de miles de millones al año, este sector, donde los sensores están conectados a una central permite cosas fantásticas (algunas de ellas mencionadas con anterioridad) como que los sistemas de tráfico se autoadministren; vigilar en tiempo real la salud de un paciente y detectar si va a tener un ataque al corazón; controlar las cosechas; dibujar la trazada de un vehículo; saber dónde está un niño o un anciano y miles de cosas más. Todo eso porque estos dispositivos -cada vez más usados en la industria, los automóviles, la medicina o la seguridad- están convirtiendo objetos inertes en inteligentes.

Es este un negocio de miles de millones. La producción de los sensores, controlada por grupos como Bosch, STMI, Schneider, Infineon o Honeywell, se ha disparado de 40.000 a 62.000 millones de euros entre 2007 y 2011 y se espera que llegue a 100.000 en los próximos tres años. Si bien todos estos grupos mencionados no son españoles, según los expertos, España puede beneficiarse de esta nueva fiebre con los sensores. Pese a no tener industria de fabricación de chips, existirá negocio con el desarrollo de aplicación e instalación de sistemas M2M (*machine*

Tipo de Sensor	2011	2016
Total	62.000	98.000
Para la automoción	14.500	20.700
De capacidad	7.000	13.000
De presión	5.110	7.350
e-Salud para móviles	407	5.016
De movimiento	1.600	4.800
Mágneticos	1.200	2.000
Wireless	790	6.700

Tabla 2.1: *Mercado de Sensores. Crecimiento en el mundo. En miles de millones*

to machine. A raíz de esto último, ya han surgido en España varias empresas, aún pequeñas, pero que crecerán en los próximos años con el sector. Oriunda de Zaragoza, Libellium ya tiene su propio dispositivo, el Waspmote, que puede ser utilizado para todo tipo de funciones, desde control de tráfico, monitorización de las vides en una bodega, invernaderos, hasta la protección de incendios en los bosques de Asturias. Las 60 versiones del Waspmote, y su venta a operadoras e integradores de estos sistemas, permite a Libellium tener una facturación de 3,5 millones al año. Urbiótica, de Barcelona y Sayme, de Santander, concentran su desarrollo en las llamadas ciudades inteligentes, monitorizando estructuras de edificios, puentes o túneles. En el caso de Sayme parten con ventaja, ya que son el único miembro español de ZigBee [16], la alianza que define el estándar en redes de sensores inalámbricas.

Las operadoras también están tomando partido en el sector. Según Cisco, el tráfico M2M se multiplicará por 22 hasta 2016, de 23 a 508 terabytes al mes. Telefónica fundo en 2011 su unidad de negocio M2M[15], cuyo tráfico creció el 30 % en 2012. Tiene ya más de 6 millones de líneas aunque a día de hoy la mayor parte de ellas la usan empresas relacionadas con las alarmas o los comercios.

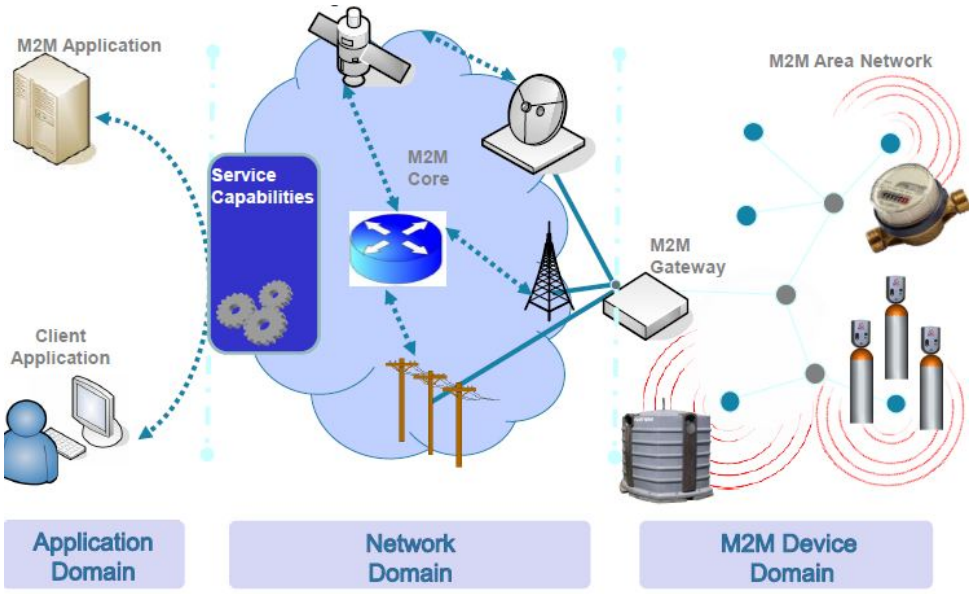


Figura 2.8: *Diagrama básico de un sistema con arquitectura M2M*

Capítulo 3

ESTRUCTURA DEL PFC E IMPLEMENTACIÓN

Este capítulo conforma una de las partes más importantes del proyecto. En el se explica detalladamente como se ha procedido a la implementación de la interfaz web, es decir, la aplicación Django. Por otro lado, tampoco se olvida de la parte electrónica del proyecto, que consiste en la conexión de los sensores a la placa Arduino, así como el protocolo utilizado para comunicar la placa con la aplicación web. Por último y menos importante se detalla el uso del demonio (escrito también en Python) para la captación de valores.

3.1. Aplicación Django

Como toda aplicación [18] Django que se precie, el desarrollo de la misma se inicia con un proyecto Django. En este caso recibe el nombre de sensorINO, que a la postre será el nombre de la web. Una vez creado, se comenzó con la creación de una aplicación Django, denominada sensorsData que es el verdadero núcleo y parte más importante del proyecto de la página web.

3.1.1. Funcionamiento de la web

Se ha comentado anteriormente que la función principal del proyecto es la de poder configurar una placa Arduino con sus sensores conectados de forma sencilla. La conexión de los sensores no tiene por qué darse al principio, se puede configurar la placa de forma autónoma y después añadir los sensores que en el momento que se necesiten o incluso cambiar el pin al que está conectado cada sensor. A continuación se muestran los pasos a seguir para poner en

funcionamiento la aplicación.

1. Instalar el proyecto Django así como todas sus librerías necesarias.
2. Conectar la placa Arduino a través del USB del ordenador que va a ser utilizado como servidor.
3. Entrar en la web sensorIno y crear un usuario a fin de poder garantizar un control de usuarios y una mejor gestión de placas y sensores asociados a cada uno.
4. Añadir una placa al sistema, con un nombre de placa. También es necesario agregar el puerto al que está conectado la placa.
5. Con la placa añadida, es el momento de los sensores. Se les debe dar un nombre, el pin de la placa al que están conectados y el propio nombre de la placa. También se necesita si el sensor es de tipo digital o analógico y añadir el modelo de sensor. Por defecto existen una serie de modelos ya configurados, pero si se requiere un modelo nuevo, con una pequeña actualización sería suficiente.
6. Configurar triggers y actuadores. Este es un punto importante, por ello se explican más adelante reservándolos un apartado propio.
7. Por último, sólo queda poner en funcionamiento el demonio y visualizar las medidas tomadas en las vistas de cada sensor.

Tras explicar brevemente el funcionamiento de la web. Llega el turno de triggers y actuadores. Son éstos los que cumplen la función propiamente dicha de controlador. En resumidas cuentas estas son sus funciones.

Triggers

Como su propio nombre indica, los triggers (gatillo en inglés) son disparadores de los actuadores. El usuario debe colocar una condición dentro del trigger, si ésta se cumple, el trigger desencadenará la acción asociada. Se han diseñado varios tipos de triggers para poder controlar correctamente todo el sistema domótico. Aquí se explican todos ellos.

- Frecuencia: se dispone de un valor (en segundos) que el usuario introduce al configurar el trigger. Se verifica en cada momento si se ha cumplido ese intervalo para activar el trigger.

- Cron: similar al comando de Linux, al introducir una fecha concreta el sistema verifica si coincide con la actual. En ese caso, también se activa el trigger.
- Asociados al valor de un sensor: configurados con un sensor y un valor condición, el trigger comprueba el ultimo valor guardado del sensor y comprueba si es mayor, menor o igual según lo dictado por el usuario.

Actuadores

Los actuadores permiten añadir una parte activa al sistema. Se configuran asociados a un trigger que en el caso de verificar su condición como activa, provocarán el funcionamiento del actuador. Un único trigger puede tener asociados varios actuadores. Sin embargo, esta afirmación en sentido contrario no ocurre. Al igual que los trigger, se han implementado varios tipos de actuadores.

- Actuador Arduino
  Implementado para escribir un valor deseado, si se encuentra activado en modo analógico o un valor 1/0 si su configuración es digital. Simplemente basta con añadir el pin al que se quiera acceder.
- Actuador Lectura
  Quizás el más importante de los implementados, permite provocar una lectura del sensor que tenga asociado, además al guardar la información también comunica con Xively.
- Actuador Log
  Permite escribir en un fichero un texto que se le ha pasado por formulario, también se podría conseguir asociarlo a un sensor para escribir medidas en un fichero de forma periódica.

Debido al diseño de triggers y actuadores, es muy sencillo añadir nuevas funcionalidades, tanto en el lado de las condiciones (triggers) como en el lado de los actuadores. En el apartado de posibles mejoras se propondrán algunas ideas que también pueden implementarse sin tener la necesidad de variar esta estructura.

3.1.2. Estructura de la web

El diseño de la web se basa en dos premisas básicas, sencillez y una estética que facilite su uso. El usuario puede acceder a toda la información de forma rápida, para ello se han diseñado una serie de vistas (views) y plantillas (templates) adaptadas al contenido que están mostrando.

Las vistas muestran la información de forma jerarquizada según la estructura que sigue la base de datos. Es decir, se parte de una placa que a su vez tiene sensores conectados a sus pines. Por tanto se tiene una vista para mostrar todas las placas almacenadas en el sistema, desde aquí se puede acceder a la vista individual de cada placa donde se puede ver un listado con los sensores conectados a sus pines así como la información asociada. Asimismo, los sensores también disponen de una vista personalizada, que incluye sus datos, los últimos valores obtenidos y la gráfica facilitada por Xively. De forma similar, la información sobre triggers y actuadores se muestra también jerarquizada, ya que como se ha comentado un trigger puede tener varios actuadores vinculados. Estas relaciones entre elementos se ven más detalladamente en el aparto referido a la base de datos.

Durante el apartado de funcionamiento de la web se ha hablado de que hay que añadir placas, sensores y demás configuración. Esta información se consigue a partir de una serie de formularios que piden la información al usuario cuando se necesita. Estos formularios permiten vincular sensores a placas, actuadores a triggers, etc. Además, disponen de los campos necesarios para identificar y crear todos los elementos del sistema. Algunos de estos formularios se crean dinámicamente según el estado de la base de datos. Así por ejemplo al añadir un nuevo sensor, el constructor del formulario busca entre todas las placas creadas por el usuario para que él pueda seleccionar la que corresponda al sensor. Asimismo, la aplicación dispone de protección frente a errores, por ejemplo, no permite añadir sensores si la base de datos no tiene almacenada ninguna placa. La configuración en varios pasos es necesaria para algunos elementos, según el tipo de actuador o trigger tendrá sentido un tipo de formulario u otro (añadir fecha, sensor asociado, condición del valor, etc), pero para no complicar al usuario sea cual sea el tipo de trigger o actuador, se configuran desde una misma vista inicial que después permite el acceso a los demás formularios relacionados.

Para facilitar la tarea al usuario, existen y son visibles desde la página principal numerosos accesos directos tanto para visualizar elementos como para crearlos. A los usuarios también pueden resultarles útiles algunos recursos creados para mostrar la instalación de los sensores, algunas páginas web con más información sobre Arduino y la domótica; o una recopilación de tipos de sensores. Para ello, la web dispone de carruseles con imágenes para mostrar toda esta información.

La aplicación web, está diseñada según el enfoque de diseño Transferencia de Estado Representacional (Representational State Transfer) o REST . Las características principales de este enfoque son:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)

- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE.

- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.

- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

Por tanto, en la medida de lo posible se ha intentado cumplir estas premisas básicas sobre REST. Con la facilidad que ofrece Django para utilizar las operaciones HTTP, por ejemplo, los formularios se construyen al realizar el GET y se validan y guarda su información cuando se utiliza POST. A continuación se muestran algunos de los recursos disponibles en la web sensorINO para mostrar el diseño basado en REST.

- /boards
  Muestra todas las placas almacenadas.

- /showBoard/nameBoard
  Para visualizar la placa con el nombre nameBoard.

- /addSensor y /addBoard
  Permite añadir sensores y placas.

- /showBoard/nameBoard/sensor/nameSensor
  Para visualizar el sensor nameSensor asociado a la placa nameBoard.

- /triggers
  Muestra todos los triggers guardados.

- /addTrigger y /addAction
  Permite añadir triggers y acciones.

- /removeBoard/nameBoard
  Para borrar la placa nameBoard del sistema

- /removeSensor/nameBoard/nameSensor
  Para borrar el sensor asociado a la placa nameBoard (esto es así para evitar borrar dos sensores con mismo nombre pero distinta placa).

3.1.3. Interfaz de Usuario

Bajo el uso del framework Bootstrap, la interfaz web consigue un gran aspecto que la hace muy atractiva para los usuarios. Se ha intentado, en la medida de lo posible, simplificar al máximo todo el diseño de la web. Como resultado, la aplicación tiene un aspecto sencillo e intuitivo. El usuario puede acceder en todo momento a las acciones básicas que son añadir elementos (placas, sensores, triggers, etc) y visualizar dichos elementos.

Anteriormente se comentaron las prácticas desempeñadas en la empresa Flir Systems, al igual que con la idea sobre el proyecto, el conocimiento de la interfaz debe mucho a las prácticas allí realizadas. Ya que el trabajo se realizaba en el grupo de interfaces de usuario, en primer lugar con la aplicación para Windows (.NET) y finalmente la app para iOS. Es conocida la importancia de las interfaces de usuario en dispositivos móviles. En muchas ocasiones el éxito de una app puede depender del diseño de interfaz que ofrece.

Presentes en Bootstrap, las tecnologías HTML5 , CSS3 y JavaScript son fácilmente reconocibles. Sin embargo, no se hace un uso abusivo de todo lo que estas tecnologías pueden aportar. En el caso concreto de JavaScript, sólo se le ha dado uso para hacer la interfaz algo más amigable, utilizando los plugins que se suministran con Bootstrap y realizando pequeños cambios allí donde se ha creído conveniente hacerlos. Siguiendo el esquema mencionado, la página está dotada con elementos típicos en toda interfaz de usuario como pueden ser popovers, menús desplegables, listas, elementos ocultables o carruseles de imágenes. Las figuras 3.1, 3.2, 3.3 y 3.4 realizan un repaso a la interfaz visible en la aplicación web.

Figura 3.1: *Aspecto de la página de inicio de la web.*

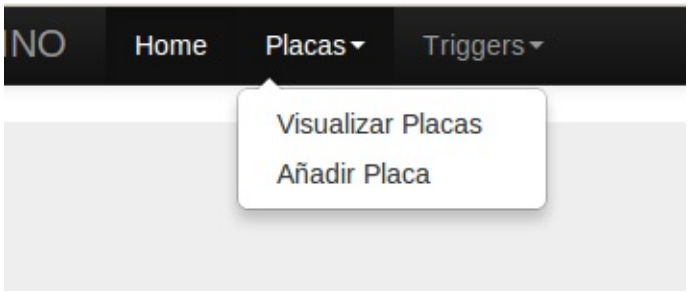


Figura 3.2: *Detalle del uso de menús contextuales.*

3.1.4. Estructura del código

Al tratarse de un proyecto Django, el código implementado va a albergar una serie de elementos propios de este tipo de desarrollos. Como Django utiliza el Modelo Vista Controlador, el controlador se encuentra representado por el archivo *views.py* (tiene un nombre contradictorio ya que significa "vistas" en inglés) y las vistas por los templates.

En views.py se localizan todos los controladores para hacer frente a las peticiones del usuario. Estas peticiones pasan por *urls.py* que contiene todos los recursos disponibles para sensorINO. Una vez llega la petición a este archivo, éste lanza el método correspondiente de views.py que tenga asociado ese recurso. Los recursos pueden ser accedidos mediante operaciones PUT o GET, para ello los métodos se encuentran preparados para en un caso sólo mostrar (GET) o

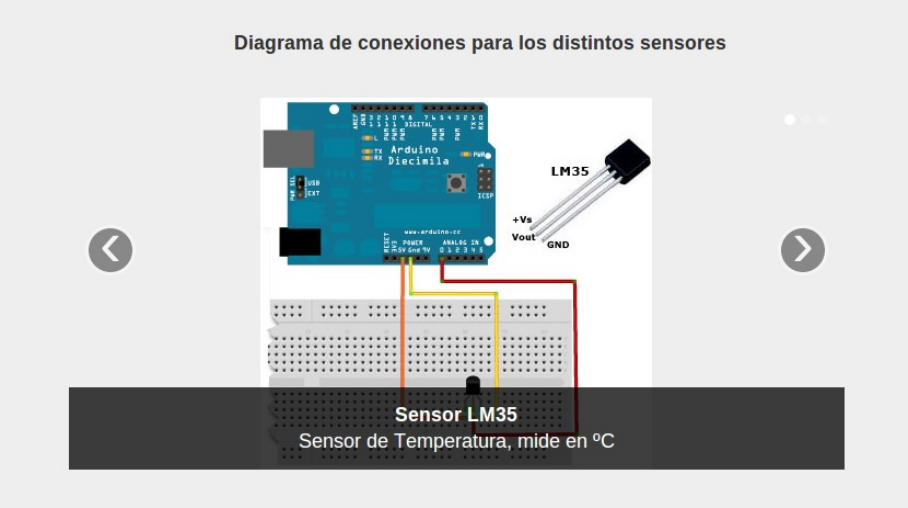


Figura 3.3: *Carrusel de imágenes para mostrar las instalaciones.*

para guardar la información (PUT). Como se ha visto anteriormente, la información se trata a través de formularios que tienen su cabida en el archivo *forms.py*. En ese archivo se encuentran los constructores que se llaman al realizar la operación GET desde el navegador. Por otro lado, los formularios deben estar perfectamente completados para que su información se procese, en caso contrario, la página ofrecerá un mensaje de error para que el usuario pueda volver a rellenar los datos de forma correcta.

Cuando la petición tiene como objetivo mostrar la información de un elemento, es necesario realizar una *Query* a la base de datos guardada en el sistema. Django facilita el uso de bases de datos al disponer de un potente motor de búsqueda para bases de datos. Todo lo relacionado con la estructura de la base de datos se encuentra disponible en el siguiente apartado de la memoria.

Existen una serie de clases para cada tipo de sensor, estas clases nombran al sensor dentro del código Django, así como su unidad de magnitud, además, disponen del método *read* que permite transformar el valor obtenido a través de Firmata en la magnitud correspondiente de temperatura, humedad, etc. Esta transformación tiene que ver con el ADC de la placa Arduino que se explicará en el apartado sensores de esta misma memoria.

Django dispone de una gran herramienta para realizar los templates o plantillas, es el lenguaje de templates. Este lenguaje permite insertar variables y bloques tag dentro del código HTML típico de la web. Las variables se encuentran contenidas entre los símbolos "{{" y "}}"; a través del contexto (*context*), el sistema mapea los nombres de variable que existen dentro del template y a su vez el renderizado del template (*render*) se encarga de rellenar con el contenido

Figura 3.4: *Detalle de formulario para un sensor.*

de la variable. Los bloques de tag (*tag block*), contenidas entre los símbolos "{ %" y " %}" se utilizan para incluir estructuras de control (ya sean estructuras if o bucles for), acceder a la base de datos o permitir el acceso a otros templates. El uso de esta última función permite tener un archivo template llamado *base.html* del que derivan el resto de plantillas y que incluye todo lo necesario para ejecutar el contenido de *Boostrap* (JavaScript , CSS) ,así como, el formato base de la web.

En los apéndices se puede encontrar un listado con todo los ficheros implicados en el código del proyecto.

Sublime Text es el programa utilizado para la elaboración de todo el código del proyecto. El programa ha sido elegido por su sencillez, su magnifico autocompletar y la posibilidad de compilar el proyecto entero desde su propio terminal. Es completamente compatible con Django y de hecho suele ser utilizado incluso en entornos profesionales para esta labor.

3.1.5. Base de Datos

Con el objetivo de modelar todo lo anterior, se ha creado una base de datos acorde a todas las relaciones descritas anteriormente. Se tienen diferentes entidades, cada una de ellas representada por una clase. Al tratarse de un proyecto desarrollado con Django, todas estas clases se encuentran incluidas en la clase *models.py*. En total existen 17 clases creadas a fin de

gestionar el sistema de la forma más coherente. Además de las 17 clases implementadas, también se hace uso de la clase *User* que dispone de todos los atributos necesarios para modelar un usuario típico de una web, como pueden ser nombre de usuario, contraseña o correo electrónico entre otras.

Para comprender esta gran cantidad de relaciones y compendio de clases, en la figura 3.5 se hace uso del comando graph_model, incluido en el paquete django-extensions y que produce en su salida una imagen .png de un diagrama basado en **GraphViz**.

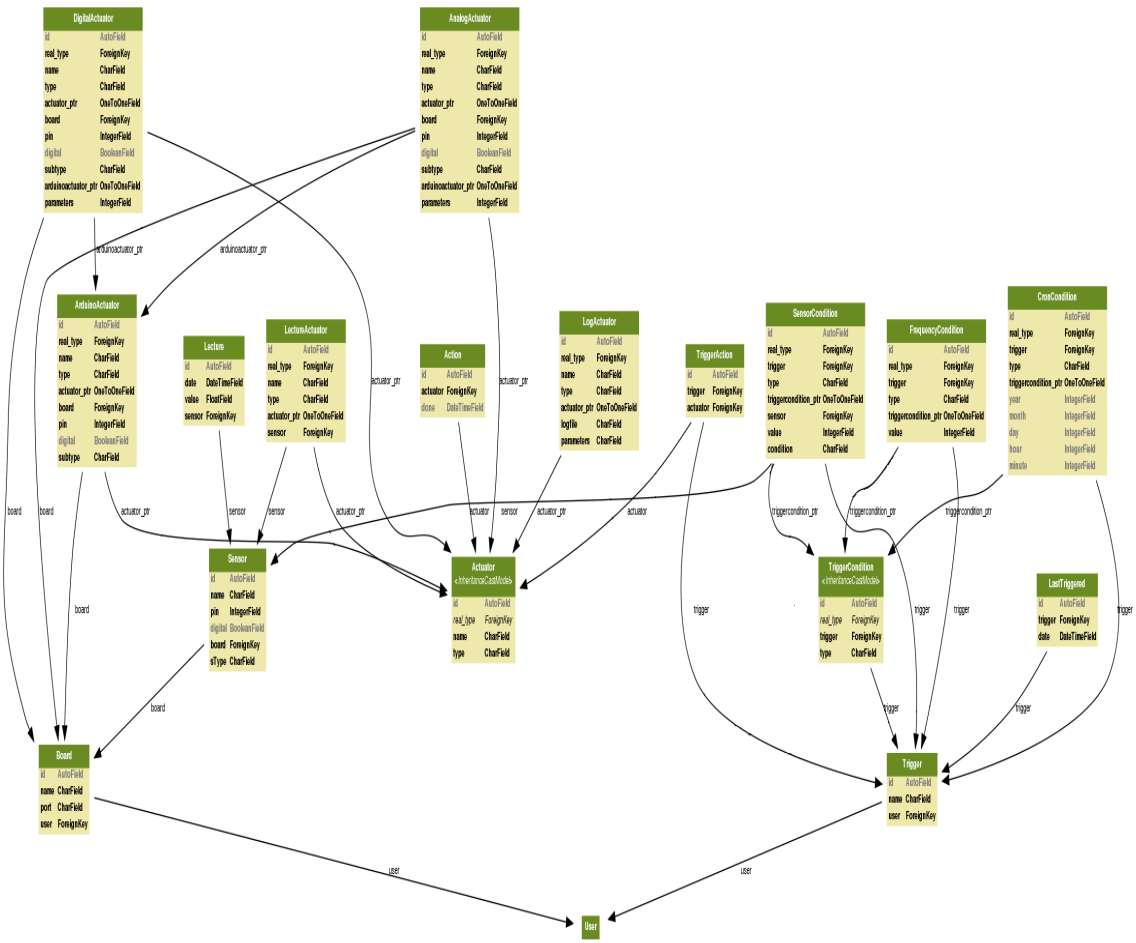


Figura 3.5: *Diagrama que representa la base de datos implementada en la aplicación*

Como se puede observar claramente en la figura, partimos de dos clases madres que son Board (Placa) y Trigger. En primer lugar nos centraremos en Board, como cabría esperar, de ella surgen las relaciones hacia Sensor y User (Usuario). Asimismo, se encuentra conectado la

entidad que corresponde al actuador de Arduino (Arduino Actuator) con sus dos tipos analógico (AnalogActuator) y digital (DigitalActuator). Estos tipos permiten al sistema escribir en el pin que tienen asociados, por tanto es indispensable conocer que placa es la que contiene dicho pin. La clase Sensor se encuentra unida a Lecture (lectura o medida), LectureActuator, que se encarga de guardar la última medida obtenida en el sensor y SensorCondition, necesaria para vincular la condición de activación del trigger con el sensor. Partiendo desde el trigger, se vuelve a ver la relación con User para que cada usuario tenga su propia configuración. Se visualizan las tres condiciones implementadas para activar el trigger: SensorCondition, CronCondition y FrecuencyCondition; todas ellas explicadas en la sección de triggers. Aparte, se observa el modelo LastTriggered que se utiliza para comprobar y almacenar si se ha ejecutado un trigger o no. Importante sin duda es la entidad TriggerAction, es la encargada de relacionar Trigger y Actuator (actuador). Esta última tiene ancladas las clases que modelan los distintos tipos de actuadores mencionados con anterioridad: ArduinoActuator, LogActuator (escribir en un fichero) y LectureActuator.

Un apunte importante que destacar es la modularidad de esta base de datos. Si se quiere añadir una nueva funcionalidad en los actuadores o una nueva condición en los triggers, bastaría con la adición de una nueva clase que represente la novedad implementada y realizar las relaciones pertinentes. Por último reseñar que para utilizar la base de datos se ha elegido el sistema de gestión **SQLite** [19].

3.1.6. Conexión Xively

Según lo comentado anteriormente, Xively permite la inclusión de gráficas para poder visualizar los valores obtenidos en los sensores de forma clara. Si se repasa el contenido del apartado sobre el funcionamiento de la web, se llega a la conclusión de que para que un sensor guarde valores en el sistema es necesario haber configurado un actuador de lectura para dicho sensor. El actuador de lectura incluye una conexión con Xively que permite guardar cada lectura en el servidor de Xively.

La API del servicio sigue la filosofía REST, permite el acceso a siete recursos: Products, Devices, Keys, Feeds, Triggers, Datastreams, and Datapoints. Cada recurso tiene asociados una serie de atributos que pueden ser opcionales y permiten completar la información de cada recurso. El formato por defecto de comunicación es JSON pero también se pueden utilizar XML y CSV. Así pues, se pueden incluir atributos para indicar el tipo de proyecto relacionado, el lugar en el mapa, el estado, tener el feed privado o público, el nombre del feed, etc. Para el caso de datastreams es obvio que es necesario incluir un atributo con unidades de la magnitud

del sensor, este valor se encuentra dentro de la clase que identifica a cada sensor dentro de la aplicación Django.

Para la elaboración de sensorINO, se ha utilizado exclusivamente los recursos Feeds, Datastreams y Datapoints. Un Feed es un conjunto de canales o datastreams vinculados a un único dispositivo. Cada placa almacenada se comporta como un feed en Xively. El canal propiamente dicho de comunicación entre la aplicación y Xively es el Datastream. Cada Datastream representa una magnitud o unidad de información, es decir, un sensor suministrando un valor de temperatura, humedad, luminosidad, etc. Con un identificador único para poder hacer peticiones contra el Datastream, este recurso consiste en una colección de Datapoints (valor y timestamp de tiempo) que se modela en sensorINO como una medida tomada en el sensor (lectura). Además, similar a las APIs RESTFull, es necesario tratar con una key para permitir la escritura en los Feeds.

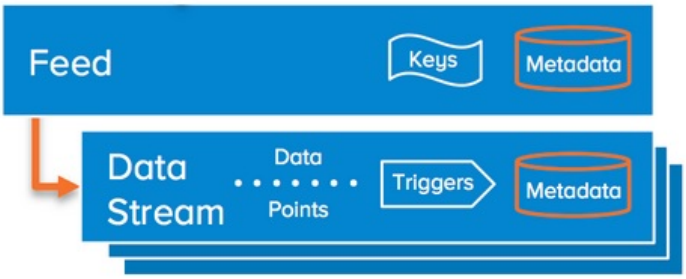


Figura 3.6: *Diagrama que representa la jerarquía implementada en Xively entre Feeds, Datastreams y Datapoints*

Elaborando todo lo anterior, la gráfica resultante de añadir los Datapoints puede conseguirse como un recurso HTML de fácil inserción en la vista propia del sensor. Si el usuario sólo quiere ver la información acerca de sus sensores, Xively proporciona una dirección web del tipo **https://xively.com/feeds/FeedID** que puede accederse desde cualquier navegador web e incluso ser pública.

En la introducción a la tecnología Xively se mencionaba la disponibilidad de varías librerías escritas en distintos lenguajes. En esta caso se utiliza la referente para Python xively-python. Con ella el desarrollador se abstrae de todo lo relacionado con el uso de JSON y el intercambio necesario para guardar los datos en la web. Además, facilita el acceso a todos los datos de DataStream y Feeds mediante sus ID únicos de cada elemento; permite escribir e incluso actualizar el contenido con métodos escritos en Python que se pueden utilizar directamente en el código

de la aplicación Django. Es esta una librería actualmente en desarrollo y de reciente publicación, de hecho, se han intercambiado correos con el responsable de esta librería en Xively con posibles mejoras o problemas, en definitiva un uso ejemplar de lo que puede ofrecer el software libre.

3.2. Electrónica

La memoria en este punto se centrará en la parte hardware, más concretamente en la electrónica asociada al proyecto. La placa Arduino, sus características básicas, el modo de conectarlo al servidor, el protocolo utilizado y sus limitaciones; así como un vistazo a los propios sensores, mencionando los tipos más comunes y la forma de conectarlos a la placa.

3.2.1. Arduino

En el apartado donde se explicaba la tecnología bajo Arduino, se mencionaba la gran variedad de modelos que existen. Seguramente si el proyecto hubiera comenzado en el momento en que se escribe esta memoria, la elección de la placa Arduino podría haber sido otra. Por ejemplo, a mes de mayo de 2013, una nueva gama de placas Arduino que reciben el nombre de Arduino Yún, se basan en la unión de una placa Arduino con otra placa que permite ejecutar Linux en su interior. Instalando el servidor y la aplicación Django en esa placa, se podría evitar el uso de un ordenador personal para ejecutarlo. También existe la otra posibilidad mencionada anteriormente al repasar los proyectos más interesantes de hardware libre. Es la placa Raspberry Pi, con suficiente potencia para poder instalar una distribución Linux y además totalmente compatible con Arduino. Sin embargo, el precio tan económico que tiene Arduino, la posibilidad de comprar la placa en tiendas no oficiales debido a su naturaleza libre y la potente comunidad de desarrolladores que tiene detrás, hacen de Arduino una solución realmente acertada. Recapitulando lo anterior, este proyecto tiene la posibilidad de ser instalado en distintas soluciones hardware totalmente válidas, evidentemente se necesitarían algunas modificaciones pero todas ellas relacionadas con la conexión de la placa a la aplicación Django, el resto de elementos quedarían intactos.

Una vez seleccionado Arduino como principal tecnología para el hardware, se tuvo que elegir entre toda la gama de placas que dispone Arduino. Según el tipo de aplicación se debe optar por una u otra placa. En definitiva, la diferencia entre placas consiste en su velocidad de refresco, el tamaño de palabra o la disposición de los pines. Bajo el proyecto aquí desarrollado, existen tres opciones que son: Arduino Leonardo, Arduino Uno (reemplazada actualmente por Leonardo) y Arduino Mega. Arduino Mega queda un poco descartada debido a su precio mayor y la nombrada

Microcontrolador	ATmega32u4
Voltaje de trabajo	5V
Voltaje de entrada	7-12V
Pines E/S digitales	20
Canales PWM	7
Entradas Analógicas	12
Memoria Flash	32 KB (ATmega32u4)
Velocidad de reloj	16 MHz

Tabla 3.1: *Características del hardware asociado a Arduino Leonardo según su datasheet.*

Uno recomiendan pasarse a Leonardo por tener un microcontrolador más moderno. Así pues, Leonardo es la placa seleccionada. En la tabla 3.1 y la figura 3.7 se muestran las características básicas de la placa Arduino Leonardo, así como una imagen de la misma.

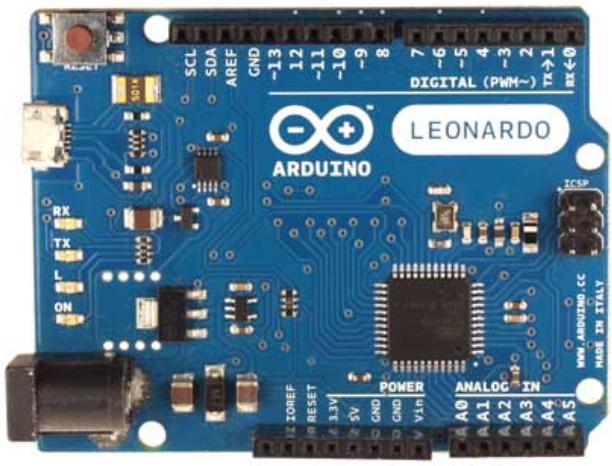


Figura 3.7: *Placa Arduino Leonardo Oficial. Se pueden observar los distintos pines así como el microcontrolador ATmega32u4.*

Por último, queda la conexión propiamente dicha con el servidor. Debido a la versatilidad de Arduino, dispone de varios módulos que le permiten obtener nuevas funciones, son las denominadas *shields*. El presente proyecto no utiliza ninguna de estos módulos adicionales ya que conecta directamente la placa a través del puerto USB del ordenador. Esta conexión USB, Arduino la ve como una conexión por el puerto serie, decir que absolutamente todas las placas Arduino tienen al menos un puerto serie, donde se permite leer para recibir datos (LED RX

parpadea) o enviar (escribir) datos en el puerto (LED TX parpadea).
Aunque esta sea la opción escogida, existen otras posibilidades:

- Shield WiFi
- Shield antena GSM
- Shield bluetooth
- Shield Ethernet

El uso de estas otras opciones también traería consigo un cambio en la implementación del protocolo de comunicación. El protocolo utilizado (Firmata) para la comunicación USB a través del puerto serie se explica a continuación.

Firmata

Firmata es un protocolo genérico que permite conectar la placa Arduino con un software alojado en un ordenador. El protocolo establece una comunicación mediante el paso de mensajes para hablar con la placa. El objetivo de Firmata es la completa usabilidad de la placa Arduino desde el software alojado en el PC. Es un protocolo que utiliza un formato de mensajes similar a MIDI, pero no utiliza el protocolo MIDI completo. Permite escribir y leer en todas las entradas que tenga el Arduino conectado, ya sean analógicas o digitales. Sin embargo, todos los sensores que se encuentran disponibles en el mercado, que son funcionales al conectarlos con Arduino individualmente, pueden no funcionar con Firmata. Esto es debido a que determinados sensores tienen un protocolo de comunicaciones propio, como pueden ser OneWire o I2C y que se comentarán en el apartado reservado para los sensores en esta misma memoria. Este problema es posible que tenga una solución próximamente, ya que el protocolo se encuentra en constante evolución e implementación. Actualmente se encuentra en su versión v2.3.5 y en la sección *proposals* (propuestas) se encuentran menciones a la integración de los citados protocolos propios de los sensores.

Por otro lado, existen dos opciones para que Firmata funcione en Arduino: una es utilizado las llamadas incluidas en la librería creando un sketch propio y teniendo en cuenta que pines están en uso; la segunda y más común, es bajar el sketch StandarFirmata a la placa. Esta opción evita conocer desde un principio que pines están siendo utilizados y según se ha visto anteriormente esto es de vital importancia debido a la estructura del proyecto. Existen numerosas librerías para el software alojado en el ordenador, escritas en varios lenguajes incluido Python, facilitan enormemente el trabajo del desarrollador añadiendo una capa de abstracción para el paso de

mensajes. Para este proyecto se ha utilizado la librería para Python denominada **Pyduino** que hace uso de otra librería denominada **PySerial**.

- PySerial

  Es un módulo, distribuido bajo licencia de software libre, para Python que encapsula el acceso al puerto serie, multiplataforma y muy conocida entre la comunidad Arduino. Instalando PySerial la lectura desde el puerto serie se simplifica enormemente, como ejemplo:

```
>>> import serial
>>> ser = serial.Serial('/dev/tty.usbserial', 9600)
>>> while True:
...         print ser.readline()
'1 Hello world!\r\n'
'2 Hello world!\r\n'
```

- PyDuino

  Es una librería que permite comunicarse con las placas Arduino cargadas con el protocolo Firmata mediante Python. Actualmente soporta la versión 2 de Firmata, luego lleva un poco de retraso frente al protocolo. Como contrapartida existe la librería pyFirmata, en principio más conocida y utilizada, sin embargo, en cuestión de resultados pyDuino siempre ha conseguido valores más razonables y coherentes. El correcto funcionamiento del programa demonio depende totalmente de esta librería. Para mostrar su sencillo uso el siguiente ejemplo permite obtener lecturas de una entrada analógica:

```
import pyduino
pin = 0
arduino = pyduino.Arduino(port)
#Analog in
arduino.analog[pin].set_active(1)
last_value = 0
while 1:
                arduino.iterate()
                value = arduino.analog[pin].read()
        if value != last_value:
                last_value = value
                print "Analog pin value is %d" % value
```

3.2.2. Sensores

Un sensor es un dispositivo que permite detectar magnitudes físicas o químicas y transformarlas en variables eléctricas. Una magnitud eléctrica puede ser una resistencia eléctrica (como en una RTD), una capacidad eléctrica (como en un sensor de humedad), una tensión eléctrica (como en un termopar), una corriente eléctrica (como en un fototransistor), etc. Una clasificación válida para los sensores es la siguiente.

1. Según la magnitud eléctrica que varía en función de la magnitud física

   a) Pasivos

   - Resistivos: Potenciómetros, NTC, PTC, LDR, Magnetoresistivos
   - Inductivos: Inductancia Variable, LVDT
   - Capacitivos
   - Ópticos (semiconductores): fotodiodos, fototransistores, CCDs

   b) Activos

   - Termopares
   - Efecto Hall
   - Piezoeléctricos

2. Según la conversión magnitud física- magnitud eléctrica

   - Directo
   - Elemento de acondicionamiento intermedio

En algunos casos, la magnitud eléctrica resultante no es apta para su lectura directa por lo que se usa un circuito de adaptación. Se requiere el uso de amplificadores o filtros para acondicionar la medida para el resto del circuito.

Todos los sensores tienen una serie de características que permiten clasificarlos e identificar su calidad respecto a otros que midan la misma magnitud física. Las especificaciones de los sensores se pueden resumir en:

- Rango: conjunto de valores entre los límites superior e inferior entre los que puede efectuarse la mediada
- Resolución: incremento mínimo de la variable de entrada que ofrece un cambio medible a la salida.

- Error: diferencia entre el valor medido y el exacto
  - Error absoluto y error relativo.
  - Errores aleatorios.
  - Errores sistemáticos.
- Veracidad (trueness) Grado de concordancia entre el valor medido obtenido de una gran serie de resultados y el valor verdadero
- Precisión (precision) : Grado de concordancia entre los resultados
- Exactitud (Accuracy): Veracidad y precisión

Para una mejor comprehensión de lo que indican estas características, se ofrece la figura 3.8

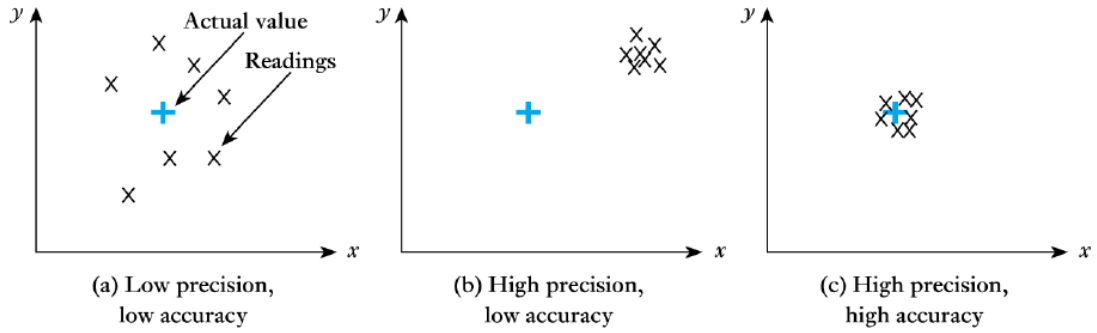


Figura 3.8: *Especificaciones del sensor. Diferencias entre precisión, veracidad y exactitud.*

Según el tipo de aplicación para la este desarrollada la aplicación sera necesario optar por un tipo de sensores u otro, como es lógico a mayor precisión y veracidad (exactitud) que pueda ofrecer el sensor, mayor será su precio.

Para finalizar con esta introducción sobre los sensores, la figura siguiente resume los tipos de sensor disponibles en el mercado según el la magnitud física que están midiendo.

Selección

Como se acaba de ver, la variedad de sensores es prácticamente infinita, existe un completo amalgama de magnitudes, salidas eléctricas, calidades y disponibilidad que no hacen sencilla la

Magnitud	Sensor	Salida eléctrica
Campo Magnético	Efecto Hall	Tensión
	Magnetorresistencia	Resistencia
Temperatura	Termopar	Tensión
	RTD	Resistencia
	Termistor	Resistencia
	CI	Tensión
	Infrarrojos	Corriente
Humedad	Capacitivo	Capacidad
	Infrarrojos	Corriente
Fuerza , Peso, Par, Presión	Galgas	Resistencia
	Célula de Carga	Resistencia
	Piezoeléctrico	Tensión o carga
	Mecánico	Resistencia, tensión, Cap.
Luz	Fotodiodo	Corriente
Movimiento, Vibración	LVDT	Tensión AC
	Piezoeléctrico	Tensión o carga
	Micrófono	Tensión
	Ultrasonidos	Tensión, resistencia, corriente
	Acelorómetro	Tensión
Flujo	Magnético	Tensión AC
	Másico	Resistencia
	Ultrasonidos	Frecuencia
	Hilo caliente	Resistencia
	Mecánico (turbina)	Tensión
Nivel, Volumen	Ultrasonidos	Tiempo
	Mecánico	Resistencia, tensión
	Capacitivo	Capacidad
	Interruptor	On/Off
	Térmico	Tensión

Tabla 3.2: *Tipos de Sensores según la magnitud física y su salida*

tarea de seleccionar que sensores son necesarios para cada aplicación.

Al tratarse de un proyecto de bajo coste y recordando el intento de conseguir algo sencillo para todo tipo de usuarios, la selección de sensores tipo se acota bastante. Echando un vistazo por tutoriales de introducción a Arduino siempre encontramos el tipico sensor de temperatura **LM35** (Figura 3.9). Es este sensor muy recomendable, barato, sencillo de instalar y con una buena precisión, además, es el que se ha utilizado como sensor de pruebas desde el principio del desarrollo. En la sección Instalación, se verá como se realiza su montaje y cómo se debe tratar la señal recibida para convertirla en temperatura (ºC).

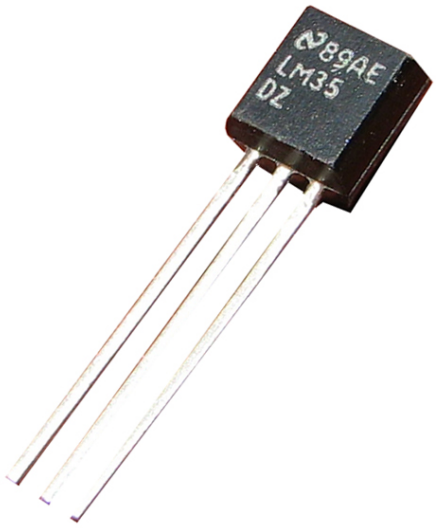


Figura 3.9: *Sensor de Temperatura LM35. Encapsulado con tres patillas.*

Para ver como podía funcionar el sistema con otro sensor de temperatura y observar el offset que existe entre ambos, se seleccionó el **LM335**, con arquitectura muy similar pero que requiere una instalación completamente distinta. Este sensor da su temperatura en grados Kelvin, por lo que la clase creada para el LM35 en el código no es reutilizable para este sensor.

Pero un sistema domótico como el que se intenta construir no sólo depende de la temperatura, al explicar las distintas aplicaciones de la domótica surge la necesidad de obtener valores de otras magnitudes como pueden ser la luminosidad o la humedad. Existen varios tipos de sensores que recogen el valor de luminosidad pero el más sencillo es la fotorresistencia o más conocido por sus siglas en inglés **LDR** (*light-dependent resistor*) (Figura 3.10) y en el proyecto es el sensor utilizado para medir la luminosidad de la sala.

La selección del sensor de humedad fue más problemática. En concreto dio lugar a conocer

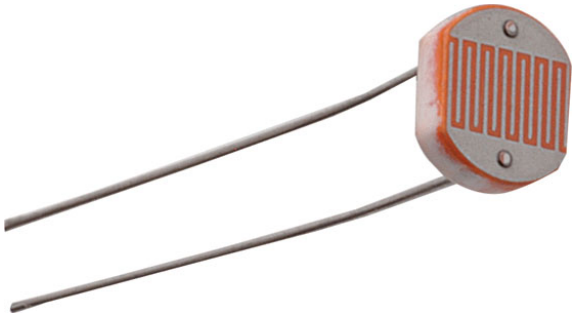


Figura 3.10: *Sensor de Luminosidad LDR.*

una limitación del protocolo Firmata. La recomendación más común para sensores de humedad es utilizar el sensor **DHT11** (Figura 3.11). Este sensor, que además de humedad también recoge valores de temperatura, tiene una peculiaridad importante: su salida es *digital*. Utilizarlo con Arduino individualmente es sencillo ya que se disponen de librerías que gestionan el protocolo del sensor para obtener directamente los valores de temperatura y humedad. Sin embargo, las implementaciones para trabajar con Firmata sólo trabajan con entradas analógicas típicas o en su defecto con entradas digitales de un valor establecido en 1/0 (On-Off) no un protocolo que depende de secuencias de bits como es el caso de este sensor. Evidentemente, esto se puede tratar como una posible mejora del proyecto pero actualmente el sistema sólo trabaja con sensores cuya salida sea completamente analógica. Este es el problema que se comentaba en la explicación del protocolo Firmata al tener determinados sensores protocolos propios como OneWire o I2C.

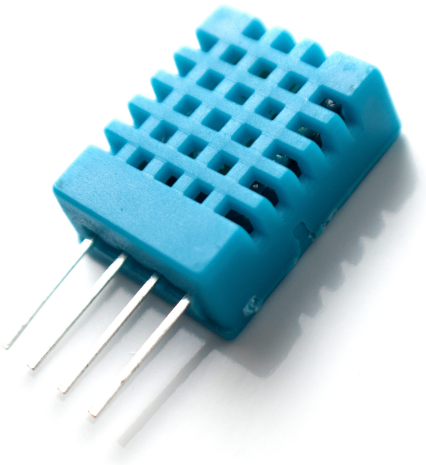


Figura 3.11: *Sensor de humedad y temperatura DHT11.*

Queda la solución de encontrar un sensor de humedad cuya salida sea analógica, el sensor **808H5V5** (Figura 3.12) sirve perfectamente para este trabajo pero su desventaja frente al DHT11 es su alto precio (4€ del DHT11 frente a más de 20€ por 808H5V5) y que sólo sirve para humedad. Existen también soluciones más económicas, pero la instalación de estos se complican enormemente.

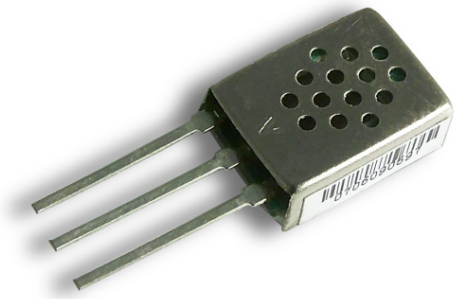


Figura 3.12: *Sensor de humedad y temperatura 808H5V5.*

Una vez evaluadas todas las opciones, se optó por conseguir los mencionados sensores de temperatura, un sensor LDR y una serie de LEDs para notificar resultados y el correcto funcionamiento de los actuadores y el sistema general. Durante esta sección, únicamente se han mencionado sensores, pero no se pueden dejar de lado los elementos que pueden beneficiarse de la función de los actuadores. Según se han definido, los actuadores terminan desembocando en las salidas de la placa Arduino. Pues bien, como se ha comentado, Arduino dispone de salidas digitales (On-Off) o analógicas de tipo PWM (*Pulse Wave Modulation*. La Modulación por Ancho de Pulso (PWM = Pulse Width Modulation) es una técnica para simular una salida analógica con una salida digital. El control digital se usa para crear una onda cuadrada, una señal que conmuta constantemente entre encendido y apagado. Este patrón de encendido-apagado puede simular voltajes entre 0 (siempre apagado) y 5 voltios (siempre encendido) simplemente variando la proporción de tiempo entre encendido y apagado. Con estas especificaciones se ofrecen algunos elementos que pueden funcionar en el sistema.

- Relé

  Sirve como interruptor, se acciona mediante un circuito electrónico que a su vez permite abrir o cerrar otro circuito electrónico. Sus utilidades son muy diversas: abrir grifos, encender luces, etc. Se puede observar su aspecto en la figura 3.13.

- Motor

  Suelen utilizarse motores de continua, transforman la energía eléctrica en mecánica. Pueden regular su velocidad según el voltaje que reciben.

- Servo

  Similar al motor pero los servos pueden ubicarse en cualquier posición dentro de su rango de operación, y de mantenerse estable en dicha posición. Muy utilizados en robótica y radiocontrol. Se puede observar su aspecto en la figura 3.14.

- Buzzer

  El zumbador o buzzer permite la conversión de energía eléctrica en energía acústica, es decir, podría ser utilizado como una alarma acústica de muy bajo coste. Se puede observar su aspecto en la figura 3.15.

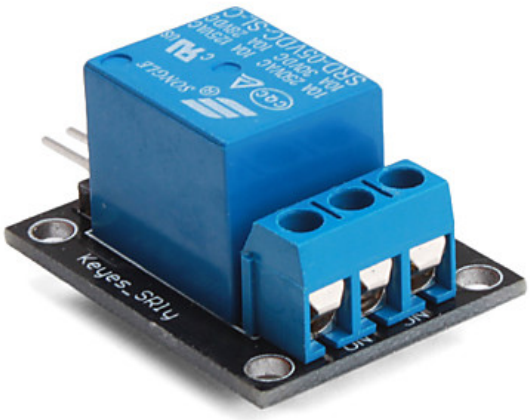


Figura 3.13: *Relé adaptado para Arduino. Este actuador es uno de los más populares.*

Instalación y uso

Como es lógico, cada sensor tiene una instalación o forma de conectarse a la placa. Sin embargo la mayoría de ellos se configuran en una simple protoboard y una serie de resistencias para conseguir la tensión deseada de funcionamiento. Los diagramas de instalación para los componentes elegidos en el proyecto pueden encontrarse en las figuras 3.16 , 3.17, 3.18.

Se puede observar que los sensores se encuentran todos conectados a entradas analógicas. Conectando los tres sensores, se utilizan tres entradas analógicas. Para obtener resultados fiables

Figura 3.14: *Servo sencillo con un rango de 360º.*

no se deben provocar lecturas de estos sensores simultáneamente, esto viene provocado debido a que el convertidor analógico-digital de Arduino se satura al recibir varias peticiones a la vez. En la aplicación domótica que se está simulando esto no supone ningún problema, simplemente se soluciona dejando un tiempo de guarda entre lecturas analógicas pero con otro tipo de aplicaciones es un factor a tener en cuenta.

Por otro lado, es necesario realizar una conversión desde la salida del conversor ADC hasta un valor de magnitud deseado. El protocolo Firmata facilita un valor entre 0 y 1 en decimales. La clase implementada para cada sensor en Python se encarga de realizar esta conversión en el método lectura. Por tanto, es esta la verdadera causa que hace que sea necesario tener distintas clases para distintos sensores. Básicamente la operación que se realiza es multiplicar la salida de la lectura Firmata por la resolución del sensor (por ejemplo en LM35 10mV por cada grado Celsius) y a su vez por la amplitud de voltaje de entrada, normalmente en Arduino son 5V como se ha mencionado en las características de la placa.

3.3. Demonio

La tercera parte clave para el correcto funcionamiento de todo el proyecto es el programa demonio. Se dice que un programa es un demonio o servicio (bajo entornos Windows) cuando el programa se ejecuta en segundo plano, sin interacción por parte del usuario. Esta nomenclatura tan poco agraciada tiene su origen la palabra inglesa Daemon que a su vez deriva de las siglas

Figura 3.15: *Buzzer adaptado para el uso en placas Arduino.*

D.A.E.M.O.N.(Disk And Execution Monitor).

Para su implementación se ha optado por utilizar un comando *custom* de *django-admin*. Siguiendo el tutorial de la pagina oficial de Django, se debe colocar en una ubicación determinada, derivar de la clase *BaseCommand* y contener un método denominado *handle()*.

Al tratarse de un comando propio de djando-admin, tiene la capacidad de acceder a la base de datos guardada por la aplicación **sensorsData**. Es necesario el acceso a la base de datos, ya que es el propio comando denominado *runboard* (por analogía con *runserver*) el que se encarga de alimentar una parte importante de la base de datos: las lecturas.

La estructura del código es similar a un sketch de Arduino, se dispone de una parte para la configuración o *setup* y otra sección con un bucle infinito o *loop*. En primer lugar, se procede al *setup*, este realiza una búsqueda en la base de datos para que a través de la librería pyduino (implementación Python de Firmata) se realice la conexión con las placas Arduino guardadas. Continua con una configuración de los pines, colocándolos en modo digital o analógico según sea necesario. Debido a las características de Firmata, este código sólo se ejecuta una vez, por tanto, en necesario ejecutar el comando de nuevo si hemos hecho un cambio a nivel de placas o sensores. Una vez la configuración esta terminada, el código implementa el bucle infinito, en cada pasada de bucle, se comprueba si los triggers están activados, de ser así se procede a ejecutar los actuadores que tenga cada trigger asociados. Al ejecutarse los actuadores en este momento, es el propio comando el que guarda las lecturas en la base de datos además de realizar otras acciones propias de los actuadores.

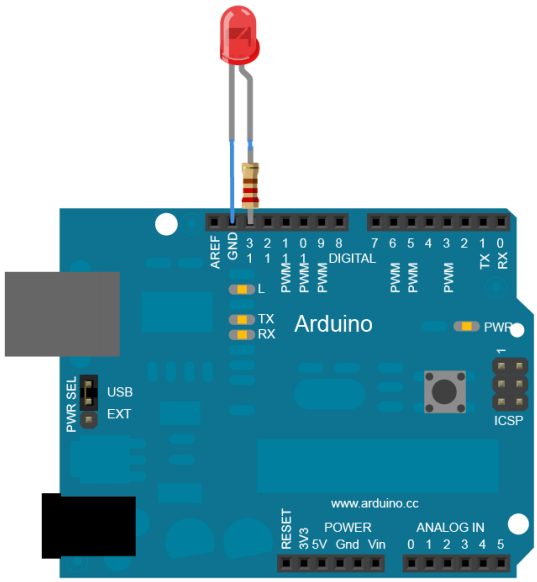


Figura 3.16: *Diagrama básico de instalación de un led.*

Cabe destacar que el demonio es independiente de la ejecución del servidor Django, tan sólo necesita que las placas estén debidamente conectadas para su configuración, que la base de datos este disponible y una conexión abierta para poder enviar los datos a Xively cuando se realizan lecturas en los sensores.

En el apartado posterior sobre las pruebas se podrá ver el funcionamiento de este demonio y el uso del comando cron [17] para Linux, que permite la ejecución de comandos de forma periódica. Este comando revisa su crontab, que no es más que un fichero de texto en el que se incluyen los comandos a ejecutar y su periodicidad. Para el comando runboard se establece una periodicidad de 5 min para comprobar que sigue funcionando.

```
# m h dom mon dow user command
 */5 * * * *      usuario /home/usuario/scripts/testrunboard.sh
```

Se puede comprobar que se ejecuta el script **testrunboard**. Este sencillo script tiene el siguiente aspecto:

```
#!/bin/sh

SERVICE='runboard'

if ps ax | grep -v grep | grep $SERVICE > /dev/null
then
    echo "$SERVICE service running, everything is fine"
```

```
else
    echo "$SERVICE␣is␣not␣running"
    python /home/usuario/pfc/sensorINO/manage.py runboard
```

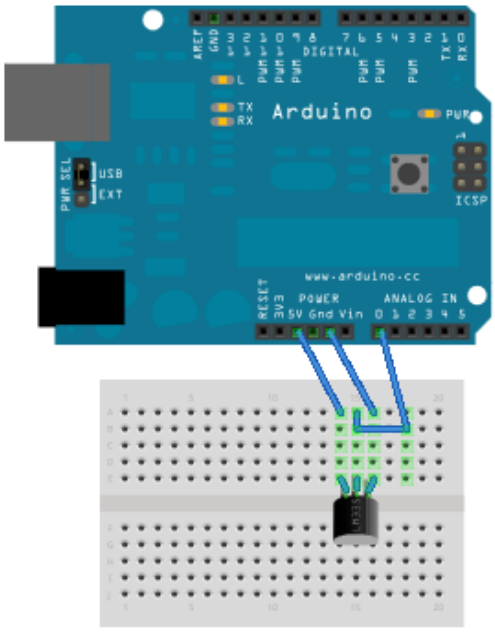


Figura 3.17: *Instalación de un sensor LM35.*

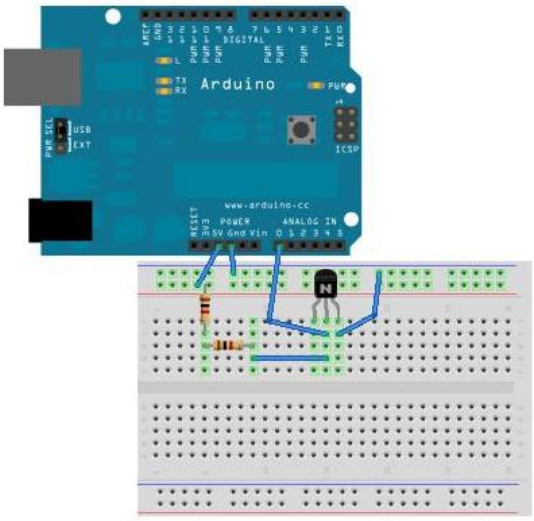


Figura 3.18: *Instalación de un sensor LM335.*

Capítulo 4

PRUEBAS Y RESULTADOS

Con un proyecto tan eminentemente práctico, se hace casi obligatorio la realización de una serie de pruebas para comprobar el funcionamiento del mismo. Es en esta sección donde se explicarán las pruebas acontecidas y los resultados de las mismas.

Es este un proyecto que sobre el papel demuestra muchas capacidades y soluciones posibles, bajo distintos entornos y realizando funciones muy dispares. Sin embargo, para obtener una verdadera prueba de estas capacidades siempre es recomendable, en la medida de los posible, la puesta en funcionamiento del sistema bajo un entorno real. Teniendo en cuenta esto, se diseñaron dos ubicaciones diferentes dando lugar a su vez a dos pruebas distintas. En primer lugar se dispuso la placa en la habitación dónde se realizó la implementación del proyecto, en este caso se trata de un dormitorio corriente. En segundo lugar, y ya con prácticamente todo el código escrito, se colocó el sistema formado por placa Arduino, servidor Django y demonio en un PC situado en uno de los laboratorios de la universidad en Fuenlabrada. Estos dos entornos junto con sus resultados se detallan a continuación.

4.1. Entorno de pruebas nº1: Habitación

Ésta no es una prueba que pueda considerarse puntual, sino que ha sido necesario hacer de su uso durante todo el desarrollo del proyecto. Desde un primer momento, se quiso dotar al sistema de veracidad, es decir, y según se ha visto en el apartado de Sensores, verificar si las medidas resultantes eran reales o por el contrario existía algún problema. Por tanto, es ésta una prueba que pretende verificar todo el sistema al completo. Para contrastar la veracidad de las medidas se colocó una estación meteorológica que dispone de medidores de temperatura en la misma habitación donde se encontraba la placa junto con los sensores de temperatura LM35

y LM335. La prueba se fue realizando por fases según se ha ido completando el código de la aplicación. Las fases son:

- Funcionamiento de la electrónica

  Cuando se recibieron los componentes, se inició la instalación de los sensores y la conexión con la placa. Para cada sensor se enviaba mediante la conexión serie Arduino la información obtenida para visualizarla a través del visor del puerto serial que se incluye con el IDE de Arduino. Sin tener aun nada de la implementación del servidor la prueba consistía en comparar las medidas frente a la estación meteorológica. Como curiosidad el clima durante este año ha sido especialmente variable lo que facilitaba la elaboración de las pruebas ante los cambios de temperatura. El resultado fue totalmente convincente ya que las medidas entre ambos sistemas eran muy similares. Sin embargo, en algunas ocasiones se producían medidas desde Arduino incoherentes, se investigó el motivo, junto con preguntas a la comunidad de desarrolladores de Arduino, llegando a la conclusión de que era un problema común en estas placas. Los motivos de estas medidas erróneas se encuentran en fallos de conversión del ADC además de que la tensión ofrecida a través del USB no es siempre constante por lo que la tensión de referencia del sensor puede variar.

- Conexión con Python y Django

  Para investigar el funcionamiento de Firmata, se realizó un pequeño código que mostraba por pantalla el valor leído por el sensor en determinado pin. Una vez comprobado que las medidas seguían siendo correctas, se pasó a desarrollar la aplicación en Django. En el inicio era una simple página HTML que abría una conexión con Arduino y mostraba lo recibido en el puerto serie con una lectura a través del protocolo Firmata. Más adelante con el desarrollo de la base de datos se pudieron mostrar las últimas medidas guardadas.

- Funcionamiento de todo el conjunto

  Una vez implementadas las clases que daban sentido a triggers y actuadores, se inició el desarrollo del demonio. Se probaron las diferentes características y funcionalidades. La prueba más interesante fue la de colocar un trigger asociado a un sensor de temperatura: al superar un umbral se debería encender el led asociado al actuador. En realidad, esta prueba constata el posible funcionamiento de una alarma básica de temperatura. Con todo el sistema funcionando era el momento de probar el proyecto en un entorno menos controlado y más grande que donde tenía lugar el desarrollo.

Figura 4.1: *Modelo de estación meteorológica inalámbrica utilizada para comprobar la veracidad del proyecto.*

4.2. Entorno de pruebas nº2: PC Laboratorio

Como antes se ha comentado, esta prueba se realizó en uno de los ordenadores de laboratorio de Linux en Fuenlabrada. El objetivo era instalar el sistema en el PC que corre en Linux, en concreto con la distribución Ubuntu 10.04 como la usada para el desarrollo del proyecto. Se dispuso de los sensores de temperatura y luminosidad para que con un trigger de frecuencia, tomaran datos del laboratorio. La duración de la prueba fue una semana y como aplicación práctica pretendía verificar si en el laboratorio había clase o no. Por tanto, sería una prueba para verificar el funcionamiento del sistema de forma continua y también la sensibilidad de los sensores. Desafortunadamente, la prueba en el laboratorio tuvo una serie de problemas, prácticamente todos ellos ajenos al propio proyecto.

4.2.1. Problemas

En primer lugar se contactó con el administrador del laboratorio para poder instalar la placa en Ubuntu del PC del laboratorio, en el ordenador donde se ha realizado el proyecto no hubo mayor problema que descargar el programa Arduino IDE y cargar a continuación los sketch requeridos. Sin embargo, debido a la configuración de los ordenadores en el laboratorio no fue posible habilitar el USB necesario para la conexión; se trataba de un problema con los permisos. Tras varios intentos se optó por utilizar una maquina virtual de Ubuntu para solventar el problema con los permisos. Utilizar una máquina virtual conlleva una pérdida de rendimiento, esta desventaja puede hacer que el demonio no realice correctamente su trabajo y tarde más en

enviar los datos a Xively. Otro problema es que el ordenador es de libre acceso y quien quiera puede acceder a él y cerrar la ejecución de la máquina virtual. Es más, durante el periodo de pruebas, pese a que el administrador bloqueó el reinicio de estos equipos (se procede a un reinicio automático cada mañana) tuvo lugar un examen donde se utilizó el ordenador para la realización del mismo, quedando la máquina virtual desactivada durante varias horas. En concreto este hecho hizo que el ordenador tuviera de nuevo la configuración de reinicio, así que cada mañana había que volver a ejecutar la máquina. Aparte del menor rendimiento, debido a la máquina virtual, el uso del resto de ordenadores (no siempre de forma correcta) conlleva a una bajada de rendimiento aún mayor por la configuración y la ejecución del $HOME en el servidor del laboratorio. Por otro lado, estas desventajas permitieron crear un código más robusto frente a posibles errores de falta de memoria o peor rendimiento. Además, se investigó el uso de la herramienta cron de Linux , para ejecutar el demonio de nuevo en caso de que se ejecución se viera interrumpida por causas ajenas. Tras añadirle esta robustez al código frente a errores, la aplicación se mantenía ejecutándose correctamente salvo por los reinicios que obligaban a una nueva ejecución de la maquina virtual como se ha comentado anteriormente.

Aún teniendo estos problemas, se obtuvieron los resultados que pueden observarse en las siguientes capturas de pantalla obtenidas en Cosm (antiguo Xively).

Básicamente se observan perfectamente la diferencia entre el día y la noche, el encendido de luces durante la tarde. En el caso de la temperatura se observan variaciones según la temperatura exterior, pero la sensibilidad de los sensores no permiten obtener datos de la presencia de personas en la sala. Quizás con el uso de otro tipo de sensores más precisos y menos sensibles a las variaciones en la tensión podrían ser útiles para llevar a buen puerto esta aplicación.

Figura 4.2: *Visualización de la gráfica en un intervalo de 6 horas. Se aprecia el aumento de la temperatura y la caída de la luminosidad según va avanzando la tarde.*

Figura 4.3: *Visualización de la gráfica en un intervalo de 1 día. El valor bajo de la luminosidad corresponde a la noche, también se aprecia la bajada de temperatura que ocurre por las noches*

Figura 4.4: *Visualización de la gráfica en un intervalo de 1 semana. Se aprecían claramente los días y las noches debido al sensor LDR. Además, la temperatura durante esa semana fue bajando, hecho constatable con las medidas del sensor de temperatura. La aparición de zonas con líneas muy rectas indican cuando la máquina fue reiniciada automáticamente y no pudo reanudarse hasta tiempo después*

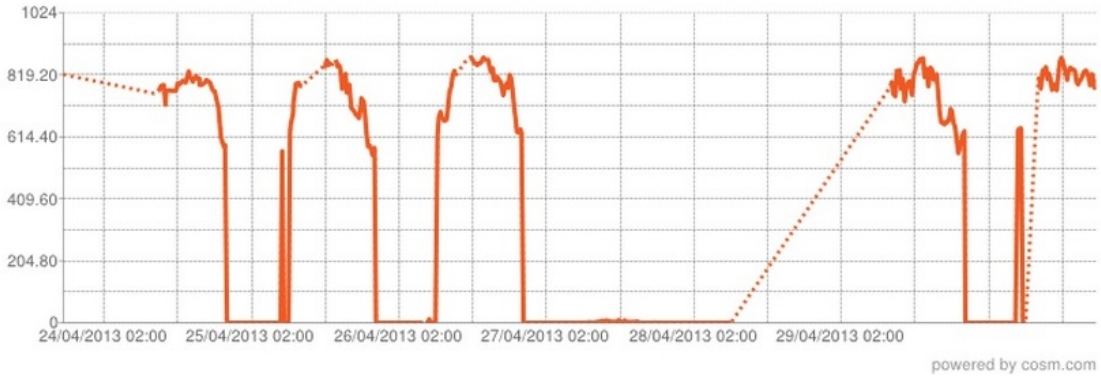


Figura 4.5: *Visualización de la gráfica en un intervalo de 1 semana para el sensor LDR. Las zonas de puntos indican la pérdida de conexión debido a las causas anteriormente mencionadas*

Capítulo 5

CONCLUSIONES

A modo de conclusión, este último capitulo contiene un repaso por los logros obtenidos, un apartado con posibles mejoras y por último una mención al presupuesto requerido.

5.1. Logros

Al principio de esta memoria, en el capítulo de Introducción, se establecieron los objetivos que se deberían llevar a cabo en el Proyecto Fin de Carrera. A partir de lo expuesto en el capítulo de Resultados se pueden extraer cómo y en qué medida se han cumplido cada uno de los objetivos.

1. Creación de una interfaz web para el control de placas Arduino. Según lo explicado a lo largo del capítulo 3, este objetivo está plenamente cumplido. Como todo proyecto de este tipo es evidente que se podría mejorar pero el proyecto es plenamente funcional para lo que ha sido diseñado.

2. Profundizar en desarrollos con Django.

   Con el proyecto se ha conseguido experimentar aún más con esta tecnología. Descubrir funcionalidades nuevas que podrán ser muy útiles de cara al futuro profesional. Por tanto, logro conseguido.

3. Familiarizarse con Arduino.

   Teniendo un presupuesto bajo como el que se observa en la sección referida a ello, la búsqueda de componentes ha requerido una investigación profunda en diversos foros y comunidades, que a su vez también contenían información acerca del funcionamiento de

estas placas y posibles ideas para proyectos Arduino. Con este objetivo se ha conseguido acercase a el tipo de programación con que se diseñan estas placas lo cual también añade un nuevo conocimiento para el autor.

4. Diseño de la web.

   Se ha conseguido obtener una web funcional y sencilla, requisitos indispensables para el diseño desde un principio. Es verdad que siendo la última fase del desarrollo no se ha podido disponer de todo el tiempo necesario para profundizar aun más en Boostrap y las tecnologías implicadas, pero ha servido para al menos conocer un poco su funcionamiento y las posibilidades que ofrecen.

5. Obtener alguna aplicación práctica.

   Las pruebas ofrecen una visión agridulce sobre este objetivo. La primera prueba fue llevada con éxito y la aplicación funcionaba perfectamente como estación, además de añadirle nuevas funciones como las de alarma. Sin embargo, debido a los problemas con el laboratorio y quizás debido a las cualidades de los sensores seleccionados, la segunda prueba no ha sido completamente correcta. Aunque es cierto que sí se pueden obtener datos acerca de esta prueba, la detección de personas no ha sido posible. Se ofrece una descripción detallada en la sección Pruebas.

6. Aprender.

   Resumiendo todo lo anterior, el autor de este proyecto ha aprendido. Y ha aprendido mucho teniendo en cuenta todas las tecnologías implicadas, los posibles aplicaciones que presenta, la tarea de enfrentarse a una tarea de esta magnitud, la búsqueda de información, obtener un presupuesto razonable. En definitiva, este objetivo se ha cumplido con creces.

5.2. Mejoras

Cuando se escriben estas líneas, el desarrollo se da por finalizado ya que si se observan los objetivos vemos que se han cumplido prácticamente en su totalidad salvo algunos pequeños detalles. Detalles que no pueden considerarse como fracasos, si no como posibles mejoras donde este desarrollo puede crecer y hacerse más completo. Durante el desarrollo del proyecto han surgido ideas interesantes, además debido a determinadas limitaciones de las tecnologías utilizadas también aparecen nuevos caminos para poder investigar en un futuro. Las posibles mejoras se resumen en:

1. Encontrar compatibilidad con una variedad mayor de sensores

   Esta es la limitación más grande del proyecto. El protocolo Firmata deja de lado por el momento a sensores con protocolo propio o digital. Es por el momento ya que la comunidad detrás de Firmata tiene varios *proposals* entre los que se encuentran OneWire, además de mejorar la implementación de I2C (actualmente muy limitada). Una posible solución sería incorporar un protocolo propio formado por mensajes a través de el puerto serie de Arduino ya que como se ha comentado este tipo de sensores si que es compatible con la placa.

2. Nuevas funcionalidades para los actuadores

   Entre las cualidades de este proyecto se encuentra la facilidad para actualizarse con nuevas funcionalidades. Al igual que se ha utilizado la API para Xively, también se podría haber implementado un actuador para enviar datos a una cuenta twitter o enviar la información a través de correo electrónico. No haría falta nada más que crear los formularios concretos con la información necesaria y después aplicar la lógica que siguen las APIs del servicio que queramos vincular.

3. Mejorar interfaz Web

   Una web siempre está en continua actualización, no sólo de su contenido y funciones, sino de el aspecto que presenta para hacerlo más atractivo al usuario. Por tanto, es este un buen punto para comenzar a mejorar el proyecto.

4. Nueva Interfaz

   La lógica del sistema es accesible a través de la web, pero ¿es posible crear una aplicación para controlar el sistema a través de un smartphone? Desde luego que sí, y es una muy buena idea debido a como se está moviendo el mercado entorno a estos dispositivos como se ha podido apreciar en el apartado sobre la domótica.

5.3. Presupuesto

La razón de ser de este apartado es la de dar una pequeña idea del coste del proyecto. No es muy común encontrar un apartado de presupuesto en una memoria que basa gran parte de su desarrollo en software libre. Sin embargo, se ha creído conveniente su incorporación debido a los elementos hardware que también forman parte del proyecto. Además, al dar un presupuesto sobre un producto cerrado se puede discutir sobre una posible visión comercial del proyecto.

Producto	Precio	Cantidad	Total
Placa Arduino Leonardo	10,50 €	1	10,50 €
Protoboard	2,50 €	1	2,50 €
Conjunto de Cables	1,60 €	1	1,60 €
Conjunto de Resistencias	3,60 €	1	3,60€
Sensor LM335	0,81 €	1	0,81 €
Fotorresistencias LDR	0,24 €	3	0,72 €
DS18B20 1-Wire Sensor	1,75 €	1	1,75 €
Interruptores	0,04 €	5	0,20 €
LED RGB	0,24 €	5	1,20 €
Led Amarillo	0,02 €	10	0,20 €
Led Azul	0,05 €	10	0,50 €
Led Rojo	0,02 €	10	0,20 €
Led Verde	0,02€	10	0,20 €
Led Blanco	0,04€	10	0,40€
Total	•	•	24,38 €

Tabla 5.1: *Desglose del precio de todos los elementos utilizados en el proyecto.*

Se ofrecen dos tablas: en la tabla 5.1 se realiza un repaso por todos los componentes comprados para realizar el proyecto, aparte de los que aparecen como seleccionados en el apartado Sensores de esta memoria, se incluyen más elementos. Estos elementos sobrantes, debido a su bajo precio, no repercuten demasiado en el coste total del proyecto, además pueden ser útiles para otro tipo de aplicaciones futuras que no se han podido completar en este trabajo.

La tabla 5.2 muestra el posible precio que podría alcanzar un sistema domótico como el utilizado en las pruebas que consta de sensores de temperatura, luminosidad o varios led para notificar alertas. Con las posibles mejoras futuras, se podría adaptar este presupuesto e incluir sensores digitales o algún actuador más complejo como relés o motores. Es evidente que es un producto muy barato para las posibilidades que ofrece y si bien tiene un largo recorrido en cuanto a mejoras se refiere, este sistema podría funcionar en entornos con pequeñas necesidades o instalaciones caseras.

Producto	Precio
Placa Arduino Leonardo	10,50 €
Protoboard	2,50 €
Sensor LM335	0,81 €
LDR	0,24
Led Rojo	0,02 €
Led Azul	0,05 €
Total	14,12 €

Tabla 5.2: *Desglose del precio de un pequeño sistema domótico*

.

APÉNDICES

APÉNDICE A

ÁRBOL DE FICHEROS

Este apéndice contiene la salida por consola del comando tree, el cual permite mostrar los archivos de un directorio de forma jerarquizada.

```
|-- arduino.sqlite
|-- bootstrap
|   |-- css
|   |   |-- bootstrap.css
|   |   |-- bootstrap.min.css
|   |   |-- bootstrap-responsive.css
|   |   `-- bootstrap-responsive.min.css
|   |-- img
|   |   |-- glyphicons-halflings.png
|   |   |-- glyphicons-halflings-white.png
|   |   |-- LDRHOWTO.png
|   |   |-- LM335HOWTO.png
|   |   `-- LM35HOWTO.jpg
|   `-- js
|       |-- bootstrap.js
|       |-- bootstrap.min.js
|       `-- jquery-1.8.2.min.js
|-- __init__.py
|-- manage.py
|-- script
|-- sensorINO
|   |-- __init__.py
|   |-- settings.py
|   |-- templates
|   |   |-- ActionConf.html
```

```
|   |   |-- addAction.html
|   |   |-- addBoard.html
|   |   |-- addSensor.html
|   |   |-- addTrigger.html
|   |   |-- base.html
|   |   |-- boards.html
|   |   |-- ConfTrigger.html
|   |   |-- finishTriggerAction.html
|   |   |-- howto.html
|   |   |-- index.html
|   |   |-- last.html
|   |   |-- login.html
|   |   |-- messagecontent.html
|   |   |-- register.html
|   |   |-- showBoard.html
|   |   |-- showSensor.html
|   |   |-- showTrigger.html
|   |   |-- tasks.html
|   |   |-- TriggerConf.html
|   |   |-- triggers.html
|   |   `-- visor.html
|   |-- urls.py
|   `-- wsgi.py
|-- sensorsdata
|   |-- CosmConstants.py
|   |-- forms.py
|   |-- __init__.py
|   |-- management
|   |   |-- commands
|   |   |   |-- __init__.py
|   |   |   `-- runboard.py
|   |   `-- __init__.py
|   |-- models.py
|   |-- sensors
|   |   |-- analog.py
|   |   |-- base.py
|   |   |-- __init__.py
|   |   |-- randomSensor.py
|   |   |-- temperatureLM35Sensor.py
|   |   `-- temperatureSensor.py
|   |-- tests.py
|   |-- urls.py
```

```
|   `-- views.py
`-- sensorsdata.png
```

APÉNDICE B

REFENCIAS ONLINE SOBRE ARDUINO

Como información adicional, en este apéndice se hace un recopilatorio con varias páginas web donde encontrar ayuda y referencia acerca de Arduino.

- Arduino oficial

  Página oficial del proyecto Arduino, es la primera página a la que debe acudir cualquier interesado en el mundo Arduino. Dispone de un populoso foro donde la comunidad de desarrolladores participa.

- Arduteka

  Blog relacionado con el hardware libre, en el se pueden encontrar tutoriales de electrónica y noticias relacionadas.

- BricoGeek

  Recomendable para envíos rápidos de componentes ya que se trata de una tienda con ubicación en España.

- Adafruit

  Empresa muy relacionada con el aprendizaje del hardware libre. Además de dispositivos basados en Arduino y productos propios dispone de una sección de tutoriales muy recomendable (*Learning)*.

- Sparkfun

  Muy similar a Adafruit, con un extenso catalogo, es una de las páginas mas importantes dentro del mundo "Hazlo tu mismo" (DYT).

- Fritzing

  Página oficial del software Fritzing que permite una documentar y crear esquemas de circuitos impresos. Es el programa utilizado para crear los diagramas de instalación de los sensores.

- Tayda Electronics

  Tienda de electrónica presente en Tailandia, con unos precios prácticamente imbatibles y un catálogo extensísimo. Además, el envío es particularmente rápido teniendo en cuenta su situación.

- Ebay

  Si existe algún sensor está en Ebay. La mayoría de los vendedores optan por el pago directo en lugar de la típica puja.

- Guía de Usuario en Español

  Recurso online muy completo con las nociones y los primeros pasos para aprender a programar para Arduino.

APÉNDICE C

DIAGRAMA DE PINES: ARDUINO LEONARDO

En este apéndice se presenta el diagrama de pines para la placa Arduino Leonardo (figura C.1), este diagrama se encuentra disponible en la página oficial de Arduino. Además, se ofrece una adaptación de la referencia disponible para cada pin.

Entradas y Salidas

Cada uno de los 20 pines digitales de Arduino Leonardo pueden utilizarse como entrada o salida. Operan a un voltaje de 5V. Cada pin puede ofrecer o recivir un máximo de 40mA y una resistencia de pull-up (desconectada por defecto) de 20-50 KOhms. Además, algunos pines tienen funciones especiales:

- Serial: 0 (RX) y 1 (TX). Se usan para recibir (RX) y transmitir (TX) datos en serie usando la capacidad del serial de ATmega32U4. Notar que para Leonardo, la clase Serial refiere al USB
- TWI: 2 (SDA) y 3 (SCL). Soportan comunicación TWI usando la librería Wire.
- Interrupciones externas: 2 y 3. Estos pines pueden configurarse para lanzar una interrupción en un valor bajo, subida o bajada del escalón o un cambio en un valor.
- PWM: 3, 5, 6, 9, 10, 11, y 13. Ofrecen una salida PWM de 8-bit.
- LED: 13. Hay un led en la placa conectado al pen 13. Cuando el pin tiene valor HIGH , el led se enciende, cuando el pin esta en LOW, se apaga.

- Analog Inputs: A0-A5, A6 – A11 (en los pines digitales 4, 6, 8, 9, 10, y 12). La Leonardo tiene 12 entradas analógicas, nombradas desde A0 hasta A11, todas ellas pueden usarse como digitales i/o. Los pines A0-A5 aparecen en la misma posición que en la Uno; las entradas A6-A11 están en los pines digitales i/o 4, 6, 8, 9, 10, y 12 respectivamente. Cada entrada analógica ofrece 10-bit de resolución(es decir, 1024 valores diferentes). Por defecto las medidas se miden desde tierra a 5V, aunque es posible variar la cota superior usando el pin AREF.

- Reset. Sirve para resetear la placa.

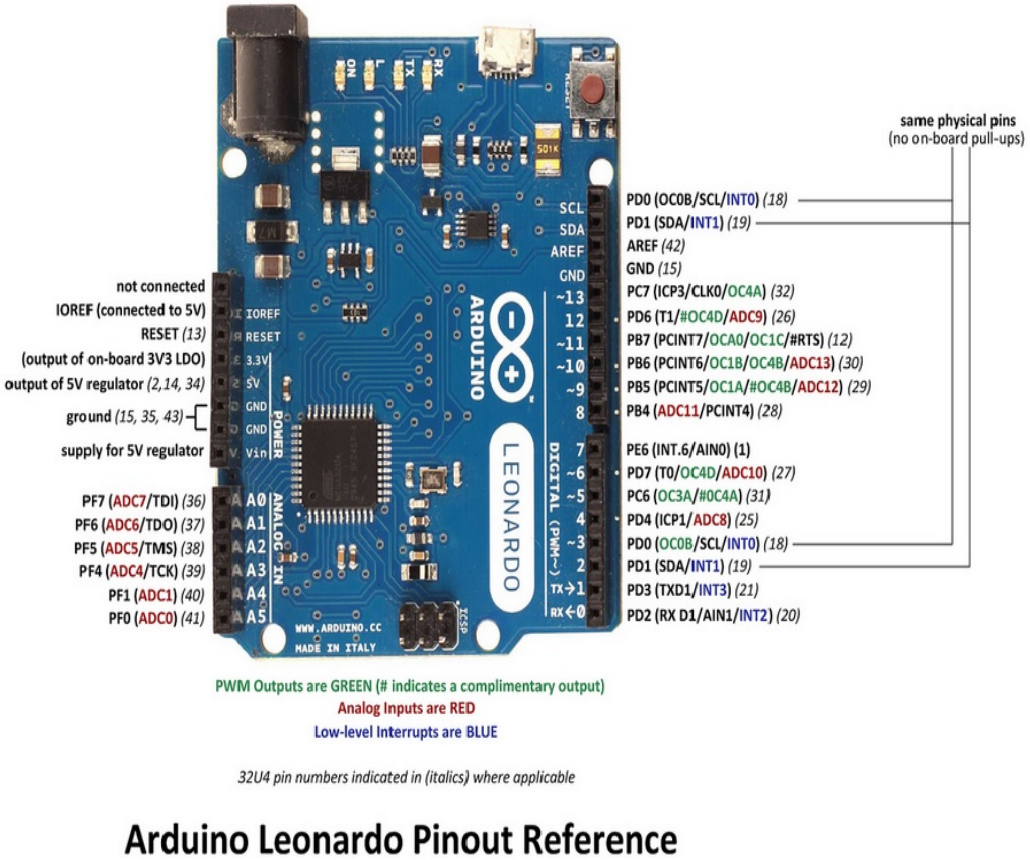
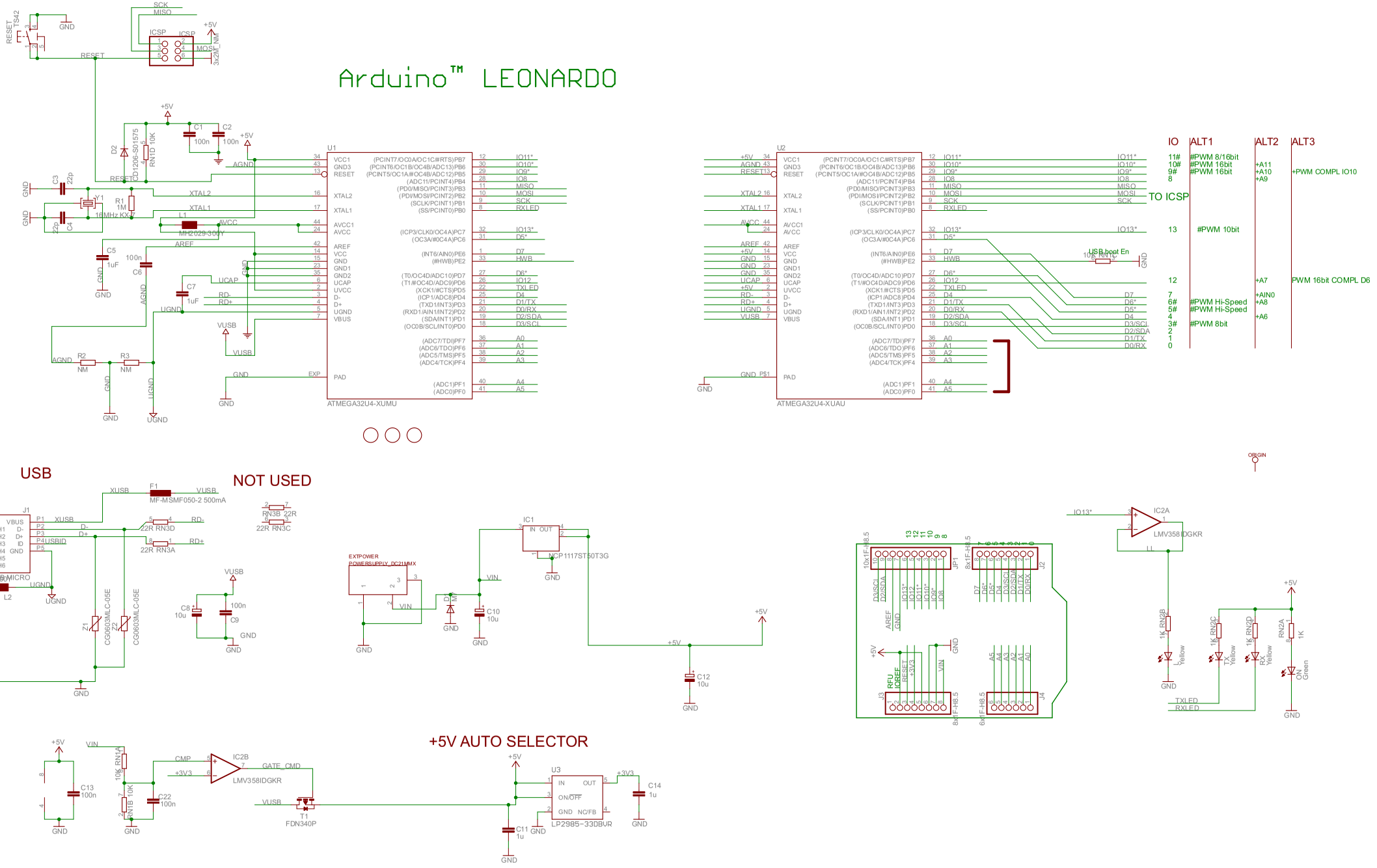


Figura C.1: *Diagrama de pines para la placa Arduino Leonardo*


Arduino™ LEONARDO
ATMEGA32U4-XUMU
ATMEGA32U4-XUAU
TO ICSP
USB
NOT USED
+5V AUTO SELECTOR
LMV358IDGKR
LP2985-33DBVR
FDN340P
MCP1117ST50T3G
MF-MSMF050-2 500mA

Bibliografía

[1] Página web de Django. [Online]. Disponible: https://www.djangoproject.com

[2] Página web de Python. [Online]. Disponible: http://www.python.org

[3] Página web de Flir. [Online]. Disponible: https://www.flir.com

[4] Entrevista a Corpora. [Online]. Disponible: http://blog.bricogeek.com/noticias/robotica/entrevista-a-the-corpora-y-el-robot-qbo

[5] Open-Source Warehouse - The Challenge. [Online]. Disponible: http://www.opensourcewarehouse.org/problem-statement

[6] Introducción oficial Arduino. [Online]. Disponible: http://www.arduino.cc/en/Guide/Introduction

[7] Página web de los creadores de FreeDuino. [Online]. Disponible: http://www.elecfreaks.com/2973.html

[8] Descarga de Arduino IDE. [Online]. Disponible: http://arduino.cc/en/Main/Software

[9] Página web servicio Xively. [Online]. Disponible: https://xively.com/

[10] Página web editor Sublime Text. [Online]. Disponible: http://www.sublimetext.com/

[11] Página web Twitter Boostrap. [Online]. Disponible: http://twitter.github.io/bootstrap/

[12] CEDOM. Asociación española de domótica. ¿Qué es domótica? [Online]. Disponible: http://www.cedom.es/que-es-domotica.php

[13] Github. Resultado de la búsqueda: Arduino + Django [Online]. Disponible: https://github.com/search?q=arduino+django&ref=cmdform

[14] Repositorio RESTDuino [Online]. Disponible: https://github.com/jjg/RESTduino

[15] Grupo M2M Telefónica [Online]. Disponible: https://m2m.telefonica.com

[16] Alianza Zigbee [Online]. Disponible: http://www.zigbee.org

[17] Manual para Cron [Online]. Disponible: http://usemoslinux.blogspot.com/2010/11/cron-crontab-explicados.html

[18] Tutorial de inicio para Django [Online]. Disponible: https://docs.djangoproject.com/en/1.5/intro/tutorial01

[19] Página de referencia del motor SQlite [Online]. Disponible: http://www.sqlite.org