Universidad del País Vasco

Euskal Herriko Unibertsitatea

Facultad de Informática / Informatika Fakultatea

Diseño e implementación de un sistema de seguridad para el hogar con interfaz en Android

Autor: David García Monje
Director: José Miguel Blanco Arbe

Proyecto Fin de Carrera, Junio 2014

Resumen

Este proyecto trata sobre el diseño e implementación un sistema de seguridad y domótico para el hogar u otros establecimientos.

El sistema consta de un detector de intrusos, que detecta la presencia por diferentes métodos combinados para no producir falsas alarmas, como pueden ser el movimiento, la vibración de ventanas y puertas producida por golpear o abrir estas, y finalmente el sonido. También dispone de sistemas de detección de humo y gases tóxicos, producidos por la mala combustión de chimeneas, calderas, calentadores de agua, calefactores y estufas. El usuario también puede acceder a los datos de todos los sensores instalados.

Como sistema disuasorio, dispone de un simulador de presencia automático, que consta del encendido y apagado selectivo de determinadas luces y electrodomésticos, como puede ser una radio. Aunque el usuario, si lo desea puede encender y apagar manualmente desde la interfaz cualquier electrodoméstico o iluminación que esté conectada al sistema.

Para controlar todo el sistema se dispone de una interfaz en Android, que puede ser accedida desde un teléfono con dicho sistema. Desde la interfaz el usuario puede activar o desactivar la alarma, activar la simulación de presencia y activar un botón de pánico, entre otras cosas. Además se puede configurar los medios por los que comunicarse en caso de que se produzca una alarma.

En el caso de producirse una alerta, se activa una señal acústica y se genera una alerta en la aplicación Android, también existe la opción de que se produzca una llamada a un número telefónico especificado por el usuario con un mensaje generado por el sistema gracias a un sintetizador de voz.

Para que todo el sistema funcione, se ha diseñado e implementado una arquitectura, hardware, software y de comunicaciones, desde la programación de los sensores, pasando por el servidor y hasta la interfaz. Para la elección de tecnologías se han utilizado de hardware y software libre en la medida de lo posible.

Para probar el sistema se ha implantado en un entorno real, siendo el prototipo implementado completamente funcional.

Índice

1. Introducción

Este documento describe el trabajo realizado, en el proyecto de fin de carrera por el alumno David García y bajo la dirección de José Miguel Blanco. El proyecto describe el diseño e implementación de un sistema de seguridad y de domótica, partiendo desde cero.

A continuación describiremos la estructura que tiene la memoria y resumiremos el contenido de cada capítulo.

En el segundo capítulo, se describen las ideas e intenciones antes de comenzar el proyecto, y como evolucionaron hasta tener como objetivo el presente proyecto.

En el tercer capítulo, se describen cuáles son los objetivos del proyecto, detallando cual es el alcance e indicando el trabajo que si se realizará y cual no. Además se explican las motivaciones del proyecto.

En el cuarto capítulo se presentan las tecnologías usadas para abordar el proyecto y se justifican las elecciones realizadas.

En los tres capítulos siguientes, quito, sexto y séptimo, se detalla con precisión el diseño de la arquitectura del sistema completo. La arquitectura se puede dividir en cuatro secciones como se puede ver en la ilustración 1. La *adquisición de datos*, el *ordenador central*, el *servidor* y la *aplicación Android*.

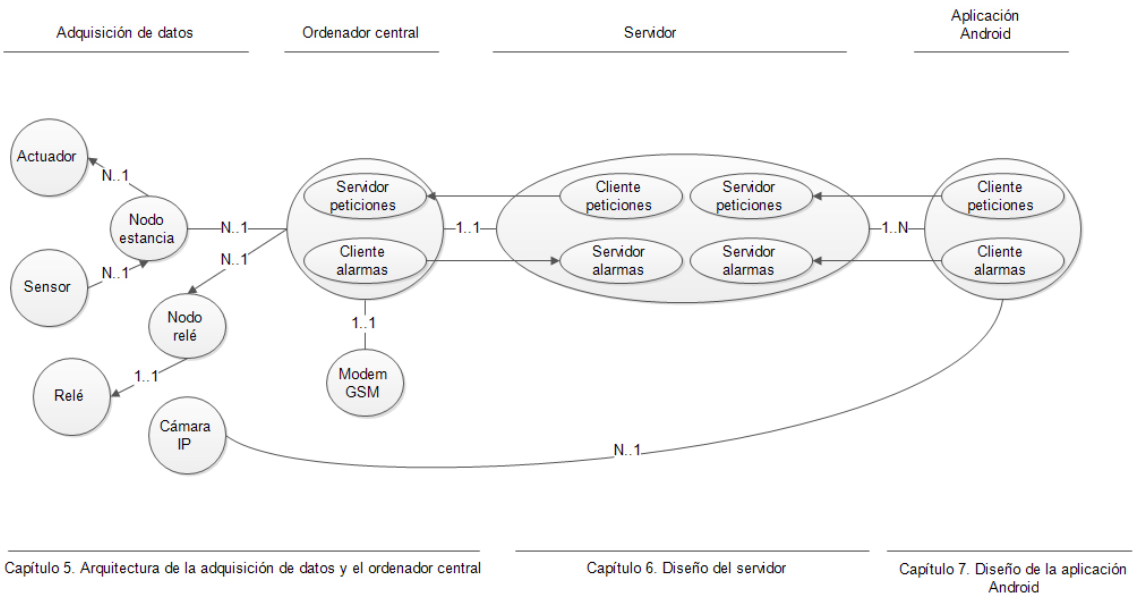

Ilustración 1. Arquitectura del sistema completo.

En la adquisición de datos, en primer lugar veremos cómo interaccionar con todos los elementos hardware desde el más bajo nivel, para después entender la categorización de los diferentes tipos de sensores así como el concepto de nodo y ver cómo puede crecer el sistema en función de los nodos, ya que como vemos en la ilustración 1, el ordenador central puede estar conectado a N nodos. También veremos cómo se conecta cada nodo con el ordenador central y el diseño del software del ordenador central. Todo lo mencionado hasta ahora pertenece a la parte del sistema que se encuentra en el propio establecimiento y se verá en el capítulo quinto, *arquitectura de la adquisición de datos y el ordenador central*.

En el capítulo sexto, *diseño del servidor,* en primer lugar veremos cómo se comunica el ordenador central con el servidor, ya que se utilizan dos canales, uno para peticiones de obtención de datos y órdenes, y por otro lado un canal reservado para el envió de alarmas. También veremos el diseño del software del servidor y como se comunica con las diferentes aplicaciones Android.

En el capítulo séptimo, veremos cómo está diseñada la aplicación Android y todos los elementos necesarios para su funcionamiento, como el uso de un servicio.

En el capítulo octavo, se describe la lógica del sistema, es decir, cómo se comporta ante cada situación.

En el capítulo noveno, se presenta la interfaz de usuario y se justificarán las decisiones tomadas para desarrollarla.

En el capítulo decimo, se describen todos los elementos relacionados con la gestión del proyecto.

Para finalizar, en el capítulo undécimo se exponen las conclusiones obtenidas después de la realización del proyecto.

2. Antecedentes

La idea para realizar este proyecto, surge dos años antes de su inicio, con la pregunta, ¿Cómo se podría medir de forma remota las propiedades físicas de un lugar distante y obtener los datos? Para responder a esta pregunta, se investigan diferentes métodos para obtener datos de sensores, y repetidamente aparece el nombre de Arduino, y el concepto de hardware libre.

El hardware libre consiste en aquellos dispositivos hardware, que se puede acceder a sus esquemáticos o especificaciones de manera gratuita, este concepto de hardware libre tiene sus inicios en la década de los setenta, pero en aquel entonces debido al alto costo de cualquier elemento, era muy laborioso replicar hardware a partir de esquemáticos y no era una práctica muy extendida, pero si que nacieron los clubs como el "Homebrew Computer Club", en el que se reunían entusiastas de la electrónica y la computación, e intercambiaban esquemáticos entre ellos, de este club en concreto, surgieron grandes fundadores de empresas como Apple. En los años noventa, gracias al abaratamiento de los componentes electrónicos y al uso masivo de internet desde cualquier parte del mundo, crecen las comunidades como "Open Design Circuits", donde por primera vez se propone la creación de comunidades de hardware con el mismo espíritu que el software libre. Pero estas comunidades no tienen mucho éxito, ya que no existía ningún software libre para el diseño de circuitos y realmente en estas comunidades no llegaron a subir muchos diseños, pero si que se produjeron grandes discusiones con una gran cantidad de personas, con las que se sentaron las bases para comunidades enteras de hardware libre. Entre finales de los años noventa y principios de siglo, es cuando realmente surge el hardware libre, gracias a que el número de transistores que se pueden integrar en un solo circuito ha crecido exponencialmente y su coste se ha reducido significativamente. Es entonces cuando explota el movimiento, surgiendo múltiples comunidades gracias a la rápida expansión de internet y la creación de múltiple software libre para el desarrollo de circuitos.

En este entorno es cuando surge la plataforma Arduino. Al ser de hardware libre, quien quiera puede descargar los esquemas y construirla paso a paso en su casa, o comprarla a algún proveedor que se encargue de fabricarla.

De forma resumida, lo que permite hacer esta plataforma, es hacer de interfaz entre elementos electrónicos y un equipo, ya sea Windows, Linux o Mac, facilitando enormemente la interacción con elementos como sensores. Aunque Arduino es hardware libre, esta plataforma también puede interaccionar con sensores o actuadores que no sean de hardware libre y no conozcamos sus

diseños internos, ya que de los elementos utilizados nos interesa como comunicarnos con ellos, no como están hechos.

Una vez elegida la plataforma Arduino, antes de comenzar el proyecto y sin saber aún que proyecto de fin de carrera realizar, se adquirieron algunas unidades de Arduino y se realizaron algunos pequeños proyectos utilizando equipos de mesa convencionales con el fin de adquirir una primera experiencia con la plataforma.

Pero controlar estos dispositivos electrónicos desde un PC convencional, resultaría algo limitado en cuanto a movilidad. Por esa razón, surge la idea de utilizar los smartphons para hacer de interfaz con los sensores.

Un Smartphone, es una combinación entre un teléfono y un ordenador convencional, teniendo más conectividad incluso que un teléfono convencional, ya que permite al usuario tener una conexión continuada a Internet, además de disponer de otros métodos de comunicación, como por ejemplo bluetooth o NFC[1]. De estas conexiones, la más destacada es la conexión a Internet, ya que permite que el dispositivo, en cualquier lugar del mundo con cobertura GSM, esté conectado a Internet y así poder recibir información en tiempo real en la palma de una mano de casi cualquier parte del mundo.

Ilustración 2. Mapa global de cobertura GSM. Las regiones en blanco no tienen cobertura.
Fuente: http://www.pocketfinder.com/

Hoy en día existen smartphons de una gran cantidad de marcas, y estas marcas necesariamente tienen que incluir un sistema operativo a sus dispositivos. Existe un conjunto de sistemas operativos diferentes, pero en este caso, hay una

[1] Near field communication (NFC): Es una tecnología de comunicación inalámbrica, de corto alcance que permite el intercambio de datos entre dispositivos.

tendencia clara con el uso del sistema operativo Android, no solo superando a los demás smartphons, si no incluso superando a todos los equipos de escritorio.

Worldwide Device Shipments by Operating System (Thousands of Units)

Operating System	2012	2013	2014	2015
Android	503,690	877,885	1,102,572	1,254,367
Windows	346,272	327,956	359,855	422,726
iOS/Mac OS	213,690	266,769	344,206	397,234
RIM	34,581	24,019	15,416	10,597
Chrome	185	1,841	4,793	8,000
Others	1,117,905	801,932	647,572	528,755
Total	**2,216,322**	**2,300,402**	**2,474,414**	**2,621,678**

Source: Gartner (December 2013)

Ilustración 3. Evolución del uso de los diferentes sistemas operativos.
Fuente: www.gartner.com/

Viendo que el sistema para smartphones más utilizado es Android, y que en el futuro aumentará esa tendencia, esta ha sido la elección para la tecnología a usar en cuanto a la elaboración de la interfaz. Hasta el comienzo del proyecto no se ha experimentado desarrollando aplicaciones para esta plataforma, pero a diferencia de las tecnologías hardware anteriormente mencionadas, no requiere un tiempo de espera hasta la adquisición del producto, por esa razón la tarea de aprender a diseñar aplicaciones para Android, se incluye dentro del proyecto.

Una vez conocida la tecnología básica a usar, era necesario pensar en un problema concreto a resolver. En primer lugar, surgieron muchas ideas diferentes sobre implantar redes de sensores el lugares externos, o distantes, pero por los activos disponibles para un proyecto de fin de carrera, se pensó que una buena manera de desarrollar esta idea era el presente proyecto debido a que uno de los principales objetivos es la implantación del sistema, y este proyecto tiene las características ideales para ello, ya que puede ser fácilmente implantado en un hogar, lo que reduce todos los posibles costes de implantarlo en un lugar diferente al de trabajo. Para centrar el proyecto un tema relacionado, teniendo en cuenta la opción elegida de implantarlo en un hogar, rápidamente surgieron los conceptos de domótica y automatización del hogar. La domótica abarca un conjunto amplio de posibilidades, y este proyecto se centra en cumplir objetivos de seguridad, automatización y comunicación de datos en tiempo real, sobre todo destacando la parte de seguridad en el entorno físico.

Antes de comenzar con el proyecto, se estudiaron los protocolos de domótica existentes para investigar si el presente proyecto se podía adaptar a alguno de

ellos. Ya que existen muchos protocolos de este tipo, como por ejemplo, C-Bus[2], X10[3], Insteon[4] o KNX[5]. Pero estos protocolos, lo que hacen es definir estándares de comunicación, para decir a los sensores y actuadores como se tienen que comunicar con la red, y para que un sensor o actuador utilice estos protocolos, hay que adquirir un sensor o actuador que ya tenga implementado el protocolo en su hardware. Por lo que adquirir un sensor para estos protocolos, es muy costoso y limitado. Por ejemplo no existe ningún sensor de los protocolos mencionados, que detecte la vibración en las ventanas. Además la mayoría de estos protocolos ya tienen implementadas centrales y toda la infraestructura de comunicaciones necesaria, que sería necesaria adquirir y tendría un coste inasumible en este proyecto.

Además de lo mencionado, estos protocolos se utilizan generalmente para para controlar elementos "simples", como luces, motores, otros actuadores y sensores. Si queremos por ejemplo encender un televisor en el canal dos o poner el aire acondicionado con potencia media no podemos hacerlo. Ya que para ello, es necesario que el fabricante del electrodoméstico integre en su producto un protocolo de comunicaciones concreto. Pero Los fabricantes de electrodomésticos, optan por integrar su propia tecnología domótica para promocionar los electrodomésticos de su marca. Ejemplos de esto son, *BDF* (Bus Domótico Fagor) o *LG Thinq*, protocolos solo utilizables por una marca especifica.

Con en animo de poder interaccionar con las funciones avanzadas de los electrodomésticos, se está intentando por parte de los fabricantes orientar el protocolo UPnP[6] a este ámbito. Este protocolo ya se usa en la mayoría de las televisiones, lo que permite tener un control total sobre ellas. Pero en otros electrodomésticos de cocina como hornos o lavadoras, parece que no tiene tanta aceptación. En el protocolo UPnP, también está apareciendo tecnología para controlar elementos domóticas "simples". Pero si lo utilizáramos tendríamos el mismo problema mencionado, una limitación muy grande en cuanto al número de sensores, actuadores y un coste económico inasumible, además de una limitación en cuanto a la arquitectura a desarrollar, por lo que no se podría cumplir el objetivo de implantar el sistema.

Una vez analizadas todas las opciones, se tomó la opción final de realizar el sistema domótica y de seguridad basado en Arduino. De esta manera se ha podido definir un sistema desde su diseño hasta su implantación en un entorno

[2] C-Bus: http://en.wikipedia.org/wiki/C-Bus_(protocol)
[3] X10: http://es.wikipedia.org/wiki/X10
[4] Insteon: http://es.wikipedia.org/wiki/Insteon
[5] KNX: http://es.wikipedia.org/wiki/KNX
[6] UpnP: http://es.wikipedia.org/wiki/Universal_Plug_and_Play

real, con un coste asumible y con una libertad total a la hora de diseñar su arquitectura.

3. Objetivos del proyecto

A continuación se exponen cuáles han sido las motivaciones principales para realizar el proyecto así como su alcance y finalmente de describen las tareas que no se han realizado en este proyecto.

3.1 Motivación

La motivación del proyecto nació con la idea de desarrollar un sistema partiendo de cero, que no se concentre en un área tecnológica específica, si no que permita adquirir experiencia práctica, con una visión global los diferentes campos estudiados durante los estudios de ingeniería Informática, desde la manipulación de datos al más bajo nivel hasta el desarrollo de una aplicación con la plataforma concreta Android, ya que durante el transcurso de la carrera no se había producido la oportunidad de estudiar esta plataforma en detalle.

También, poniendo un especial interés en la adquisición de datos, ya que viendo todos los elementos tecnológicos que existen hoy en día que adquieren datos, desde el sensor de llenado de una lavadora, hasta el sensor de inclinación de un avión, el autor de este proyecto sentía un vacío con respecto al funcionamiento de estos sistemas.

3.2 Alcance

En el alcance del proyecto definiremos los procesos necesarios para asegurarnos que el proyecto incluya todo el trabajo necesario para finalizarlo con éxito, es decir, indicaremos que se incluye y que no se incluye en el proyecto.

Puesto que este proyecto de fin de carrera no ha sido acordado con ninguna empresa, ni con ningún otro interesado con el rol de cliente, los objetivos han sido definidos por el propio autor del proyecto y acordados con el director del proyecto.

- Capacidad de detectar la presencia por tres métodos diferentes.
    - Detectores de presencia piro eléctricos.
    - Detección de ruido.
    - Detección de movimientos o vibraciones de elementos críticos.

- Capacidad de medir las siguientes magnitudes.
    - Cantidad de humo
    - Temperatura
    - Intensidad de la luz
    - Presión atmosférica
    - Humedad
- Capacidad de apagar/encender la iluminación de una estancia.
- Existencia de un dispositivo portátil que permita encender/apagar el electrodoméstico que se enchufe a él.
- Uso de un zumbador o sirena para anunciar las alarmas de fuego.
- Existencia de un dispositivo capaz de realizar llamadas telefónicas automáticas para comunicar alertas y que genere las alertas de texto hasta voz.
- Uso de una cámara IP con el fin de que el usuario pueda confirmar visualmente las alertas.
- El sistema tiene que tener la capacidad de estar en funcionamiento continuo 24 x 7 días.
- El sistema tiene que tener la capacidad de recuperarse por sí mismo después de un error o fallo sin la intervención de personal, en el caso que sea posible.
- El sistema será capaz de operar en un establecimiento sin el requerimiento de una IP estática o el uso de un servicio similar.
- El sistema será controlado desde una aplicación nativa para el sistema operativo Android, que tendrá las siguientes características.
    - Permitirá al usuario configurar los parámetros de configuración del sistema.
    - Permitirá ver los datos actuales de cada sensor.
    - Permitirá ver los datos de cada sensor de las últimas 24 horas.

- Permitirá modificar el estado de los actuadores.
- Permitirá añadir eventos de encendido o apagado por cada actuador en rangos puntuales.
- Dispondrá de un log con las últimas alarmas ocurridas.
- Informará de las alarmas ocurridas en tiempo real, aun si no está la aplicación activa.
- Incluirá integrada una interfaz para ver y mover la cámara IP.
- Permitirá ver el estado de cada parte del sistema en tiempo real, si está funcionando correctamente.

- El sistema tendrá que ser completamente implantado en un entorno de pruebas y tendrá que estar completamente funcional. Se implementará completamente en una estancia de un hogar y se realizará una simulación automática de una segunda estancia para probar la arquitectura con múltiples estancias.

3.3 Esquema de descomposición del trabajo

A continuación se presenta el esquema de descomposición del trabajo junto con la explicación de cada tarea.

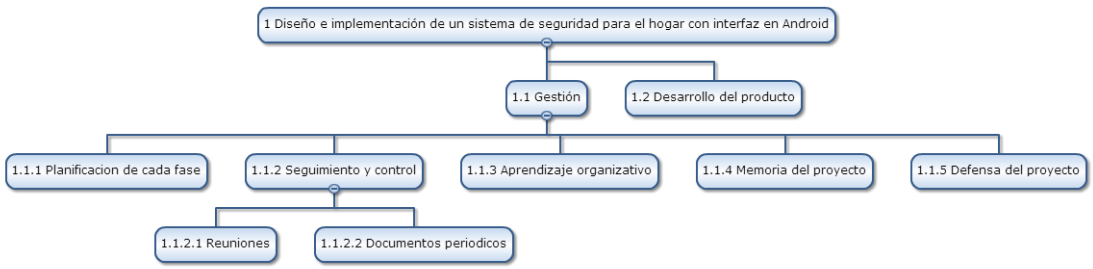

Ilustración 4. EDT del proyecto.

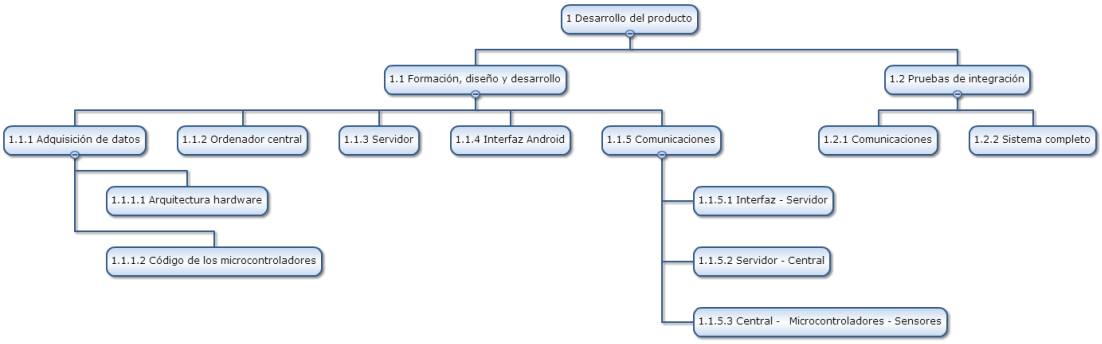

Ilustración 5. EDT del producto

El primer EDT muestra las tareas el proyecto entero, siendo el desarrollo del producto una de sus tareas, que queda descompuesta en el segundo EDT. Las demás tareas pertenecen al grupo de gestión, a continuación se detallan algunas de ellas.

- **Planificación de cada fase**. La planificación del proyecto no se ha realizado al comienzo del mismo, si no que en el comienzo de cada fase se ha planificado los contenidos de la misma. Esta tarea engloba todos los procesos necesarios de la planificación de cada fase.
- **Seguimiento y control**. El seguimiento y control engloba todas las tareas necesarias para controlar que el proyecto se está desarrollando según lo planeado.
- **Aprendizaje organizativo**. El aprendizaje organizativo representa el proceso continuo de adquisición de nuevo conocimiento y habilidades necesarias para la realización del proyecto.

En el EDT del producto se incluyen todas las tareas necesarias para que el producto sea desarrollado con éxito. Se divide en dos categorías, por un lado las pruebas realizadas, y por otro todo el proceso de formación diseño y desarrollo. Esta última categoría se he dividido en función de las diferentes secciones que contiene el producto, la adquisición de datos, el ordenador central, el servidor, la aplicación Android y todas las comunicaciones.

3.4 Exclusiones del trabajo

En esta sección se han listado todas las tareas que no se han realizado en este proyecto, pero que si serían necesarias realizar o estudiar en el caso de que el producto tuviera que salir al mercado.

3.4.1 Aspectos legales

El sistema desarrollado en este proyecto, guarda información de carácter personal en el servidor, ya que el usuario tiene la opción de incluir números de teléfono. Para tratar estos datos adecuadamente es necesario aplicar todos los pasos indicados en la ley orgánica de protección de datos.

El sistema desarrollado puede enviar avisos automáticos a cualquier número de teléfono, pero si entre los números de teléfonos a enviar avisos, incluyéramos el número de cualquier autoridad, como la policía o los bomberos se estaría incumpliendo una ley, ya que la mayoría de los sistemas de seguridad, no tienen

el privilegio de poder contactar directamente con las autoridades, si no que tienen que contactar con una central receptora de alarmas, donde el personal de la central es quien interpreta las alarmas de los sistemas y decide si avisar a las autoridades. Todos estos aspectos se podrían estudiar en la Ley 5/2014, de 4 de abril, de Seguridad Privada[7].

3.4.2 Tareas relacionadas con la seguridad

Estas tareas, representan todo el trabajo necesario a realizar para que el sistema no tenga vulnerabilidades ante un posible atacante o fallo inevitable.

- **Seguridad de las comunicaciones, cliente, servidor y ordenador central.** Cuando entre estas partes se intercambian datos, el destinatario de los mismos no tiene la certeza de que quien se los está enviando es el emisor esperado. Un atacante que se pusiera en el medio de las comunicaciones, podría analizar el método de comunicación utilizado y realizar un ataque de "man in the middle", haciéndose pasar por cualquiera de las partes. Para que el atacante no pueda ver los datos enviados, sería necesario añadir una capa de seguridad a los sockets con SSL. Y para verificar las credenciales de cada parte la utilización de certificados digitales.
- **Seguridad del servidor y el ordenador central**. Ambas máquinas están conectadas a internet, y pueden ser atacadas. Por ello es recomendable aplicar todos los consejos generales de seguridad, como mantener el sistema operativo actualizado. Entre los posibles ataques, el más peligroso y fácil de realizar es un ataque de denegación de servicio, que consiste en realizar múltiples peticiones, ya que al no ser máquinas preparadas para responder muchas peticiones, es especialmente vulnerable. Para evitar este tipo de ataques, bastaría con discriminar con un firewall, todas las conexiones de direcciones no acreditadas. Dependiendo del nivel de seguridad deseado, también se podrían incluir máquinas redundantes, para activarse en el caso de fallar las activas.
- **Seguridad en la adquisición de datos**. Al igual que en los casos anteriores, en la adquisición de datos, se podría realizar un ataque de "man in the middle", ya que las comunicaciones son inalámbricas, para este caso también se podría realizar tareas de cifrado de datos y verificaciones de los datos. Además, en este caso un atacante podría utilizar un inhibidor de frecuencias, para bloquear todas las comunicaciones inalámbricas y así anular el sistema. En

[7] Ley 5/2014, de 4 de abril, de Seguridad Privada: http://www.boe.es/diario_boe/txt.php?id=BOE-A-2014-3649

este caso se podría estudiar la detección de estos dispositivos por la emisión continua de ruido que emiten.

3.4.3 Despliegue del sistema para más de un establecimiento

El diseño actual del sistema sí que se contempla la posibilidad de que el sistema completo esté instalado en N establecimientos diferentes, pero no se ha diseñado para que el posible instalador del sistema de cada establecimiento pueda gestionarlos todos eficientemente.

Por ejemplo, para cada establecimiento u hogar en el que esté instalado el sistema, el diseño actual contempla que es necesario una instancia de un servidor para cada uno. Pero en el caso de que fuera instalado de forma masiva en diferentes hogares, sería recomendable que una estancia gestionara más de un establecimiento, para minimizar costes y centralizar los datos de todos los usuarios.

4. Elecciones y tecnologías a usar

En este capítulo se presentan las tecnologías usadas para cada parte del proyecto, además se estudian y se comparan con otras también disponibles hoy en día. Para empezar veremos las tecnologías relacionadas con el hardware, para después pasar a ver las del servidor y finalmente las de la aplicación Android.

4.1 Tecnologías del sistema de adquisición de datos

El sistema de adquisición de datos es el que interacciona con el entorno, ya sea recibiendo datos de sensores o actuando en el entorno mediante actuadores, se encarga de tomar datos del mundo real y digitalizarlos para que un ordenador sea capaz de entenderlos. En primer lugar veremos la plataforma Arduino y después la tecnología usada para las comunicaciones.

4.1.1 Arduino

Arduino es una plataforma de hardware libre para interaccionar con elementos electrónicos. Consta de un microcontrolador RISC, normalmente de 16 MHz, y una placa que contiene entradas y salidas analógicas. También dispone de un entorno para desarrollar software e introducirlo en los microcontroladores.

El proyecto de Arduino se inició en 2005 y poco a poco ha ido creciendo hasta convertirse hoy en día en la mayor plataforma de este tipo, siendo ampliamente usada.

Una de las razones por la que se ha elegido esta plataforma, es la magnitud con la que se ha extendido, por lo que existe una comunidad muy grande de desarrolladores, en la que hay un ambiente muy colaborativo.

Para la mayoría de sensores y actuadores comerciales que existen, si el fabricante no ha desarrollado una librería de su dispositivo para Arduino, en la comunidad habrá un proyecto similar, por esa razón los conocimientos necesarios de electrónica de circuitos para abordar un proyecto con Arduino, no son muy superiores a los impartidos en los estudios de ingeniería informática.

Dado que la variedad de dispositivos electrónicos que pueden interaccionar con Arduino es casi ilimitada, existe una total libertad a la hora de desarrollar una gran variedad de proyectos hardware con Arduino.

A continuación se incluye una lista con ejemplos de proyectos realizados con Arduino:

- Ardupilot[8]: software y hardware de aviones no tripulados.
- ArduSat[9]: satélite basado en Arduino de código abierto.
- Myspectral[10]: espectrómetro hecho con Arduino.
- Openbci[11]: interfaz cerebro-computador para Arduino.

Para programar el microcontrolador de las tarjetas, existen múltiples entornos de desarrollo, pero en este proyecto se utilizara el IDE oficial de Arduino. El lenguaje que utilizaremos es el propio de Arduino, siendo este una combinación de los lenguajes C y C++ con algunas características añadidas y otras reducidas.

Aunque a la hora de programar para esta plataforma hay que tener en cuenta sus características, ya que los microcontroladores que utiliza Arduino, no funcionan bajo ningún sistema operativo, por lo tanto todo el código que se programe será el único que esté funcionando adentro del microcontrolador. Además, estos microcontroladores, no admiten excepciones, es decir no existe la estructura *try – catch,* por lo que cuando se produce una posible excepción pueden ocurrir dos cosas. Si la excepción puede permitir que el sistema siga funcionando, lo hará. Por ejemplo, cuando accedemos a un *array* fuera de su índice, no inicializamos una variable y accedemos a su valor, o cuando incrementamos el valor de una variable fuera de sus límites, el programa continuará funcionando con valores erróneos. Por otro lado cuando ocurre un error que no permite seguir con el funcionamiento, el hardware y software se reinicia automáticamente. También hay que tener en cuenta que estos procesadores, no admiten *hilos*, si hay que hacer más de una tarea simultáneamente, la labor de gestionarlas es del programador.

4.1.2 Comunicaciones inalámbricas y XBee

Para comunicar las diferentes partes del sistema de adquisición de datos es necesario hacerlo por medio inalámbrico, ya que si lo hiciéramos por medio cableado estaríamos desarrollando un producto menos atractivo al requerir una instalación más compleja y poco estética.

Existen muchas alternativas para que un proyecto adquiera capacidad inalámbrica, en el mercado hay múltiples dispositivos de bajo coste que por radio frecuencia permiten intercambiar información, normalmente, estos dispositivos de bajo coste, no suelen tener una documentación buena y al existir tantos tipos

[8] Ardupilot: http://en.wikipedia.org/wiki/Ardupilot
[9] ArduSat: http://en.wikipedia.org/wiki/ArduSat
[10] MySpectral: http://myspectral.com/
[11] Openbci: http://www.openbci.com/

diferentes cada uno funciona de un método diferente. Para no tener que asumir riesgos innecesarios, se ha optado por utilizar dispositivos XBee[12], ya que es una plataforma muy conocida.

Existe una amplia gama dispositivos XBee, se pueden clasificar, por muchas características diferentes, entre las más importantes se encuentran, la topología de red, velocidad de transmisión y la distancia de conexión.

Los dispositivos XBee utilizan el protocolo de comunicaciones Zigbee[13]. Zigbee es un protocolo de comunicaciones inalámbricas creado por Zigbee Alliance, una organización sin ánimo de lucro. Este protocolo es ampliamente recomendado para sistemas como el que se presenta en este proyecto, ya que tienen un consumo bajo y cada nodo no requiere mucha electrónica, en contra posición, no permiten velocidades de transmisión superiores a 250 kbit/s, es decir no pueden transportar información de forma masiva como lo hacen los protocolos Wi-Fi o bluetooth, pero son muy eficientes trabajando con datos de sensores.

4.2 Tecnologías del ordenador central

El ordenador central es quien se encargará de gestionar toda la adquisición de datos, y a su vez hacer de interfaz de comunicaciones con el servidor, por esa razón necesitamos una tarjeta algo más potente que los 16 MHz que pueden ofrecer Arduino.

Para este proyecto se ha elegido el ordenador de placa reducida Raspberry pi [14]de modelo B, en el mercado existen muchos ordenadores de este tipo, pero la elección ha sido esta, ya que por su coste es el que mejores prestaciones ofrece.

- SoC: Broadcom BCM2835
- CPU: ARM 1176JZFS a 700 MHz
- GPU: Videocore 4
- RAM: 256 MB
- Video: HDMI y RCA
- Resolución: 1080p
- Audio: HDMI y 3.5 mm
- USB: 2 x USB 2.0
- Redes: Ethernet 10/100
- Alimentación: Micro USB

[12] Dispositivos Xbee: http://en.wikipedia.org/wiki/XBee
[13] Protocolo Zigbee: http://en.wikipedia.org/wiki/ZigBee
[14] Raspberry Pi http://es.wikipedia.org/wiki/Raspberry_Pi

- Sistema operativo: GNU/Linux

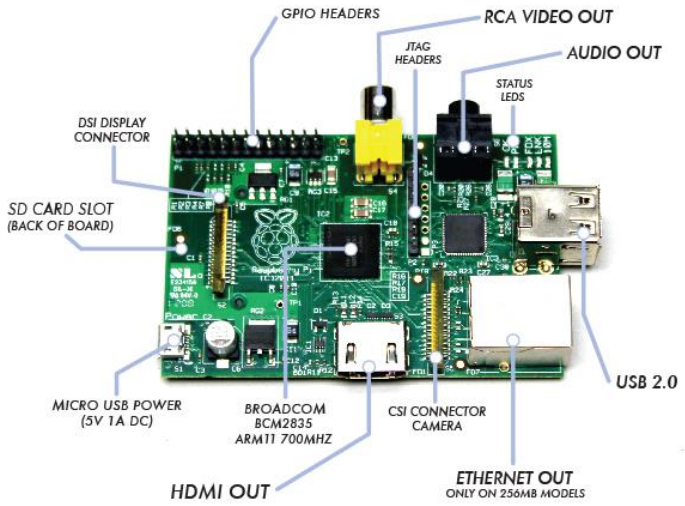

Ilustración 6. Ordenador de placa reducida, Raspberri Pi.
Fuente: http://www.pcmag.com/

Como se puede ver por las especificaciones técnicas, hemos pasado de los 16 MHz de Arduino hasta 700 MHz y con sistema operativo Linux. Esto nos permite disponer de un amplio abanico de alternativas a la hora de escoger la tecnología con la que desarrollaremos el software del ordenador central.

Para esta tarea se ha elegido el lenguaje Python, esta elección puede llamar la atención, ya que es un lenguaje de alto nivel, y necesitaremos manejar estructuras bit a bit. Pero este lenguaje, también posee algunas herramientas para tareas de bajo nivel, además, la distribución Linux que está especialmente diseñada para este ordenador, Raspbian[15], contiene todas las herramientas necesarias de Python para desarrollar con este lenguaje, in situ. Es decir, en el mismo hardware que hayamos desarrollado el software, será donde lo probaremos. Esta manera de programar es una gran ventaja, ya que agiliza mucho la velocidad de desarrollo. Esto es debido a que una de las finalidades de Raspberri Pi, es incentivar la programación en Python. Al utilizar este lenguaje tendremos que tener en cuenta que es en lenguaje interpretado y la mayoría de los errores aparecen cuando se ejecuta el código en vez de en tiempo de "compilación".

Como IDE utilizaremos Geany[16], ya que es extremadamente ligero, aunque como inconvenientes, no tiene tecnologías como la corrección de la sintaxis del código

[15] Raspbian: http://es.wikipedia.org/wiki/Raspbian
[16] Geany: http://en.wikipedia.org/wiki/Geany

en tiempo real o ayudas para acceder directamente a la documentación de las funciones utilizadas en el código, como podría tenerlas el entorno Eclipse más el pluging para Python. Las únicas ayudas que tiene en cuanto a programación, es el coloreado de la sintaxis y una lista de las posibles llamadas a las APIs, pero muchas veces esta ayuda incluso puede generar problemas, ya que en un módulo concreto puede indicar llamadas a funciones que no son visibles para ese módulo o indicar llamadas a funciones cuyas librerías no se han importado.

Para conectarnos al ordenador central para prográmalo, utilizaremos la tecnología de escritorio remoto, lo que nos permitirá acceder a él desde cualquier lugar con conexión a Internet.

Además, el ordenador central se encargará de generar las llamadas de voz, para generar un audio narrado con la información necesaria, se utilizará el sintetizador de voz festival[17], software libre únicamente disponible en Linux.

4.3 Tecnologías del servidor

Para desarrollar el servidor se ha utilizado una plataforma de computación en la nube, ya que es el encargado de la lógica central sistema y de gestionar las comunicaciones entre clientes. Para ello se ha elegido Amazon EC2.

Amazon EC2, es un servicio que ofrece Amazon de computación en la nube, esto nos permite alquilar una máquina por horas y disponer de ella con un control total de la misma mediante permisos de Administrador.

Para este trabajo se ha optado por una máquina Linux, ya que de esta forma se logra tener una homogeneidad en todo el proyecto con respecto al mismo sistema operativo. Por la misma razón, para programar el servidor se ha utilizado el lenguaje Python. Esta vez la programación no se ha realizado en el propio servidor, si no desarrollando el código en una máquina local con sistema Windows y subiéndolo al servidor Linux cuando es necesario. Esto es debido a que las condiciones de la máquina en Amazon, son fácilmente simulables en una máquina local. Además como el lenguaje Python es multiplataforma, no hay ningún problema con este método de desarrollo.

4.4 Tecnologías de la aplicación Android

Para desarrollar la aplicación, se ha decidido hacerlo de forma nativa para los dispositivos Android. Se ha optado de esta manera, en primer lugar, debido a que

[17] Festival: http://en.wikipedia.org/wiki/Festival_Speech_Synthesis_System

las necesidades del proyecto requieren disponer de funcionalidades que solo se pueden acceder con una aplicación nativa y además uno de los objetivos del proyecto es formarse en el desarrollo de aplicaciones para este sistema operativo.

Como entorno de desarrollo se han barajado dos alternativas. Primero se ha estudiado la posibilidad de desarrollar la aplicación en el nuevo entorno, *Android Studio*, este IDE está exclusivamente desarrollado para implementar aplicaciones nativas para Android y ha sido desarrollado por Google, se anunció en mayo de 2013 y aunque está disponible para descargar de forma gratuita, aun es una versión "early acces". La principal razón por la que se ha descartado este entorno, es la escasa comunidad que tiene comparada con la de su principal competidor, ya que en caso de usar esta alternativa, hubiera ralentizado la velocidad de aprendizaje debido al número reducido de manuales y otros documentos.

En cambio el IDE eclipse, con el pluging para desarrollar para Android, ADT[18], es un entorno maduro, con muchos años de recorrido que contiene una gran cantidad de documentación. Por esa razón ha sido esta la elección tomada.

[18] ADT: http://developer.android.com/tools/sdk/eclipse-adt.html

5. Arquitectura de la adquisición de datos y el ordenador central

En este capítulo se detalla la arquitectura del sistema existente al más bajo nivel. En el apartado de la adquisición de datos se muestre el funcionamiento de todos los elementos hardware y como se comunican, así como algunos de los problemas de estos elementos que han afectado al diseño hardware de los nodos. Y en el apartado del ordenador central se detalla el hardware del mismo y el software desarrollado en Python.

5.1 Arquitectura de la adquisición de datos

Este sistema, es el encargado de leer los datos de los sensores y enviárselos al ordenador central, también interacciona con los actuadores, como los relés.

El sistema está diseñado por nodos, definiremos un nodo como un conjunto de sensores y actuadores que dispone la capacidad de comunicarse con el ordenador central. Cada nodo dispone de una tarjeta de adquisición de datos, que empaqueta la información de todos sus sensores y la envía. La comunicación se produce en las dos direcciones, así que también puede recibir órdenes del ordenador central.

Existen dos tipos de nodo, el *nodo de estancias* es un nodo que tiene conectados tantos sensores y actuadores como se necesiten y el *nodo relé portátil*, que solo tiene conectado un elemento.

Se ha tomado esta decisión de arquitectura por nodos, como se puede ver en la ilustración 7. Ya que un nodo necesita una fuente de alimentación y un dispositivo de comunicaciones, si no se hubiera diseñado así, cada sensor tendría que tener estos dos elementos, como los sensores de los protocolos mencionados en el capítulo de antecedentes, lo que supondría que cada sensor para su alimentación tuviera pilas o alimentación por cable y un módulo de comunicaciones de coste alto.

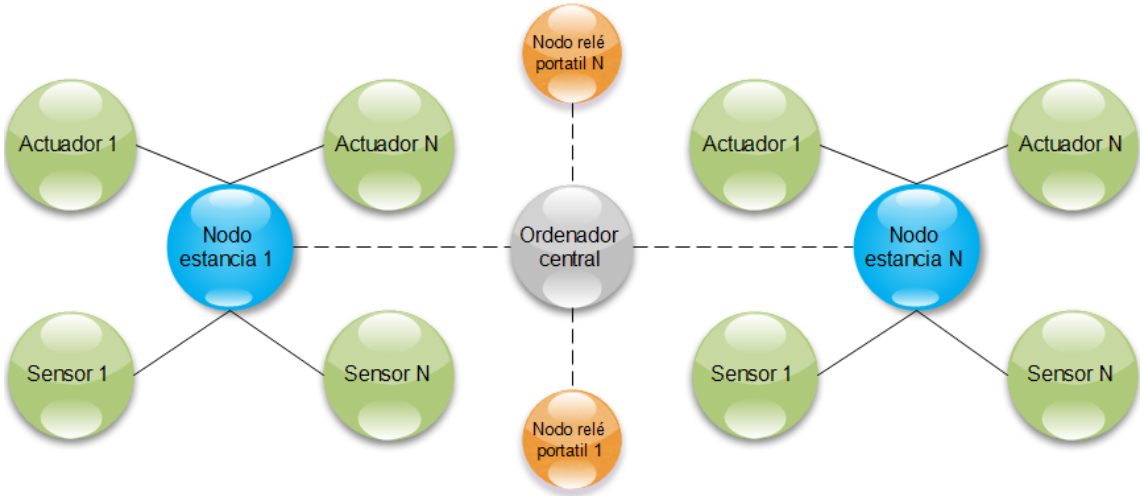

Ilustración 7. Diseño de las comunicaciones en la adquisición de datos. Los enlaces discontinuos representan conexiones inalámbricas y los continuos conexiones cableadas.

En este apartado veremos como funcionan todos los elementos del sistema de adquisición de datos y como interaccionan entre si y con el ordenador central. Empezando por los sensores y actuadores, para continuar con las tarjetas de adquisición de datos y para finalizar con la comunicación entre estas y el ordenador central.

5.1.1 Sensores y actuadores

Los sensores y actuadores son los elementos finales del sistema de adquisición de datos, son quienes se encargan de medir las magnitudes del entorno y actuar sobre él. Ya sea transformando magnitudes físicas en voltajes o transformando voltajes en funciones de control sobre el entorno.

Si consideramos estrictamente lo que es un sensor o actuador, únicamente es el elemento necesario para producir esta transformación. Pero si consideramos solo a estos elementos, la comunicación requeriría mucha electrónica adicional. Por esa razón, todos los sensores y actuadores adquiridos para este proyecto, incluirán integrada toda la electrónica necesaria, y a partir de ahora, llamaremos sensor o actuador, al elemento más su módulo electrónico. Ya que el diseño electrónico de los sensores, no está entre los objetivos de este proyecto, si no utilizarlos adecuadamente.

En la clasificación siguiente, no estamos incluyendo de qué tipo es el elemento del sensor. Si no el módulo del sensor completo, a que tipo pertenece en función de cómo nos comunicamos con él y en que formato se transmiten los datos.

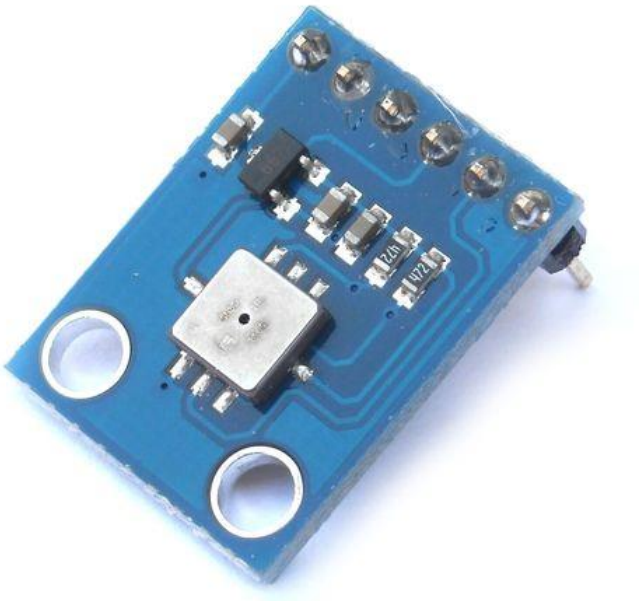

Ilustración 8. Módulo de sensor, para medir temperatura y presión.
Fuente: www.emartee.com

Ilustración 9. Elemento sensor,
para medir temperatura y presión
Fuente: www.digikey.com

5.1.1.1 Sensores analógicos

Un sensor es considerado analógico, cuando las magnitudes físicas que mide las convierte en voltajes que están comprendidos entre dos valores. También se puede dar el caso en vez de tomar como salida voltajes, lo haga en función de la intensidad de la corriente, pero nosotros no trabajaremos con este último tipo.

Considerando que los sensores analógicos emiten un voltaje entre dos valores, el valor será un número entero y la precisión de este valor viene definida por la resolución del sensor. Cuanta más resolución tenga el sensor, más preciso será.

Por otra parte necesitamos leer el valor del sensor, al igual que el sensor tiene un resolución, la tarjeta con la que leemos el dato también tiene una resolución de lectura. Para todo el proyecto, este valor será una constante, ya que utilizaremos el mismo tipo de tarjeta. El conversor analógico de Arduino es de 10 bits, esto significa que convertirá tensiones entre un rango de 0 y 5 voltios a un valor entre 0 y 1023, por lo que tiene una resolución de 4.9 mV, es decir, para que Arduino note que la salida del sensor ha cambiado, al menos tiene que variar en 4.9 mV.

No es una resolución alta, ya que en el mercado existen tarjetas con mayor resolución, pero es suficiente para el tipo de sensores que utilizaremos en este proyecto.

Del mismo modo, el rango de voltajes que admite la tarjeta es una constante, como hemos mencionado, de cero hasta cinco voltios. Por lo que tendremos que utilizar también sensores que emitan un voltaje entre este rango, si quisiéramos adaptar un sensor que emite un voltaje entre [0,12]v hasta nuestra tarjeta con [0,5]v podríamos hacerlo simplemente añadiendo circuitos adiciones.

Una de las razones por la que se fabrican sensores con diferentes rangos de salidas en cuanto al voltaje, es el ruido. En un entorno industrial, el ruido electromagnético es alto y al ampliar el rango de salida, los datos son menos afectados por el ruido. Normalmente se utilizan de [0,24]v. Pero para nuestro caso, todos los sensores que utilicemos en el proyecto pertenecerán al rango de [0,5]v.

Para leer una salida analógica de un sensor con Arduino, tenemos que conectar la salida del sensor a una de las entradas analógicas de Arduino, dependiendo del tipo de tarjeta, tendrá desde 5 entradas analógicas en adelante. Leer un valor analógico en Arduino es muy sencillo. Únicamente hay que llamar a la función analogRead que está incorporada en la API de Arduino.

```
val = analogRead(analogPin);
```

Si en el microprocesador de la tarjeta, solo hiciéramos la operación anterior, podríamos realizarla 10.000 veces por segundo, ya que el conversor analógico tarda aproximadamente 100 µs en leer un valor.

5.1.1.2 Sensores digitales

Un sensor lo consideraremos digital cuando las magnitudes que mide las convierte en un valor discreto. Por ejemplo, en el caso de los sensores analógicos, para una misma salida de un sensor concreto, tarjetas diferentes pueden hacer interpretaciones diferentes, dependiendo de su resolución. Pero en el caso de sensores digitales, puesto que normalmente funcionan bajo un protocolo de comunicaciones, un valor concreto emitido por un sensor digital solo tiene una posible interpretación.

Normalmente para obtener un dato de un sensor digital, haremos una petición mediante el bus de datos al circuito integrado del sensor. Al igual que los procesadores tienen registros, este tipo de sensores también tienen registros, donde guardan el ultimo valor adquirido más los datos de configuración,

entonces lo único que tenemos que hacer es pedir este dato mediante un protocolo establecido.

Este tipo de peticiones, varían completamente en función de la arquitectura del sensor y son complejas de realizar, ya que requieren un estudio profundo de la documentación del sensor especifico. Como ese no es el objetivo de este proyecto, utilizaremos librerías ya implementadas para comunicarnos con estos sensores. Normalmente estas librerías las implementa el propio fabricante del sensor, pero en el caso de que no sea así, en la comunidad de Arduino suele existir un proyecto de utilidad para esa labor.

5.1.1.3 Sensores binarios

Como su nombre indica, este tipo de sensor solo produce un valor con dos estados diferentes, bajo o cero voltios cuando no ocurre nada y alto cuando se produzca un determinado evento. Leer el estado de estos sensores es trivial, podemos hacerlo de la siguiente manera llamando a la función digitalRead.

```
val = digitalRead(inPin)
```

5.1.1.4 Sensor de luz

El sensor de luz tiene la función de medir la intensidad de la luz de una estancia. Siendo cero una ausencia total de luz y cien una intensidad muy fuerte.

Para medir esta magnitud utilizaremos una foto-resistencia, es decir un componente que varía su resistencia eléctrica en función de la intensidad de la luz que recibe. Para poder medir con Arduino el valor de esta resistencia utilizaremos el siguiente circuito.

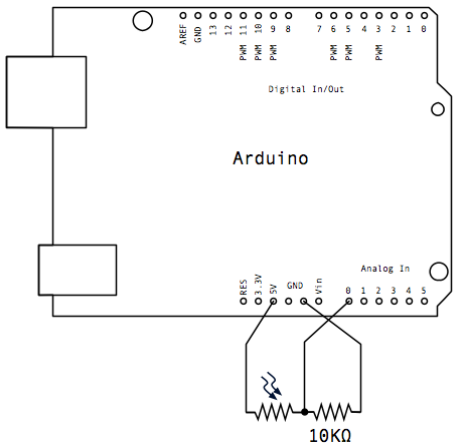

Ilustración 10. Esquema para leer los valores de una fotorresistencia.

El método para recibir los datos es el descrito para los sensores analógicos.

Para probar este sensor se ha variado la iluminación de una estancia desde una oscuridad completa, hasta una intensidad muy fuerte, obteniendo unos valores entre [0,250], donde la escala de posibles valores está situada entre [0,1023]. Como solo queremos obtener un valor en el rango [0,100] se ha "remapeado" el rango de valores de entrada a el rango deseado con la función map[19].

5.1.1.6 Medición de temperatura

Para medir la temperatura se ha utilizado el sensor BMP085. Es un sensor digital muy preciso, que permite detectar temperaturas entre -40 y 85 grados Celsius con una precisión de 0.01 grados. El sensor se puede ver en la ilustración 8.

El dato de la temperatura se almacena en los registros del sensor, por eso, para acceder a los datos del sensor utilizaremos una librería. En este caso, el propio fabricante, Bosch, pone a disposición de la librería bmp085driver[20].

Para comunicarse el sensor con la tarjeta, es necesario utilizar el protocolo I^2C, esto nos es totalmente indiferente a la hora de utilizar la librería. Solo necesitamos saberlo para conectar el sensor, ya que este bus utiliza dos cables para la comunicación, uno para transmitir datos (SDA) y otro para sincronizarse (SLC).

Para probar el sensor de temperatura se ha variado la temperatura del sensor, junto a un termómetro convencional y los resultados producidos han sido los esperados.

5.1.1.7 Medición de presión

La presión atmosférica es la fuerza con la que el aire ejerce presión sobre la superficie terrestre. Simplificando, si tenemos una presión que es la normal para la altura que se encuentre el sensor, significa que en la atmosfera, por encima del sensor no hay ninguna anomalía. En cambio, sí tenemos una presión más baja de lo normal, significa que hay algún fenómeno en la atmosfera que está reduciendo la presión, normalmente frentes nubosos. También se puede interpretar una tendencia de descenso de presión como posible mal tiempo y una tendencia de aumento de presión como buen tiempo.

[19] Función map: http://arduino.cc/es/Reference/map
[20] Librería bmp 085: https://code.google.com/p/bmp085driver/

Para medir la presión atmosférica se ha utilizado el mismo sensor que para la temperatura, es decir el sensor BMP085 puede medir dos magnitudes al mismo tiempo.

Para comunicarnos con el sensor, es exactamente igual que en el caso anterior, pero llamando a funciones diferentes.

Para probar el sensor se han comparado los datos suministrados con los datos de presión proporcionados por la estación meteorológica Igueldo[21], que está a cinco kilómetros del entorno de pruebas y a la misma altura sobre el nivel del mar. Los resultados han sido satisfactorios.

5.1.1.8 Medición de humedad

La humedad es la cantidad de vapor de agua en el aire. Para medir esta magnitud se ha utilizado el sensor digital DHT22, es un sensor de coste muy bajo y de precisión muy baja también, de ±1 %.

Para acceder a estos datos se ha utilizado una librería creada por Adafruit[22], que es la organización que se encarga de distribuir este sensor.

Para probar el sensor se ha variado la humedad del ambiente y el sensor se ha comportado de la manera prevista.

[21] Estación meteorológica Igueldo: http://es.windfinder.com/forecast/san_sebastian_igueldo
[22] Librería DHT 22: https://github.com/adafruit/DHT-sensor-library

5.1.1.9 Detección de movimiento

Para detectar movimiento utilizaremos un detector de movimiento piroléctrico HC-SR501 . Este tipo de detectores utilizan la radiación infrarroja para detectar personas, ya que cualquier objeto por encima del cero absoluto, es decir, más caliente que cero kelvin, emite radiación infrarroja, esta diferencia de temperatura que tienen las personas con respecto al ambiente, es la que se utiliza para la detección.

Ilustración 11. Sensor HC-SR501. Fuente: http://www.vetco.net/

El sensor utilizado es un sensor binario, emite un voltaje cercano a cero voltios cuando no hay presencia y cinco cuando se detecta.

El sensor tiene dos paramentos que se pueden ajustar físicamente con dos potenciómetros, uno es la distancia de detección y otro es un parámetro que indica cuantos segundos el sensor estará indicando presencia cuando la detecte, ya que es un sensor "inteligente" que sigue manteniendo el estado de cinco voltios unos segundos después de la detección para que a la tarjeta le dé tiempo a procesar el dato.

Para probar el sensor se han hecho pruebas de presencia, y se ha podido comprobar que la detección es satisfactoria hasta una distancia de 8 metros.

5.1.1.10 Detección de sonido

Para detectar ruidos usaremos un micrófono *electrec* integrado en un circuito. Este es un sensor analógico de muy bajo coste que devolverá un valor entre cero y cinco voltios en función de la magnitud del valor detectado.

Para acceder a los datos que nos proporciona este sensor lo haremos por el método tradicional de los sensores analógicos.

Las primeras pruebas hechas con este sensor han sido para comprobar que niveles de ruidos detecta y como reacciona ante ellos, para ello se ha conectado el sensor a una placa Arduino y está a un PC para poder ver los datos. Para que la TAD remita los datos obtenidos desde una entrada analógica, al puerto USB solo necesitamos el siguiente código.

```
void loop() {

  Serial.println(analogRead(A0));

}
```

Ilustración 12. Ejemplo de redirección de entrada analógica a puerto serial.

Ahora podremos ver en cualquier ordenador, con cualquier programa que nos permita acceder a el puerto serial, toda la información que emite el sensor. Veremos números comprendidos entre 0 y 1023.

Como resultados de las primeras pruebas, se ha observado que en un ambiente sin ruidos, no siempre se estabiliza la salida del sensor, ya que produce una salida de x ± 5 donde x es el valor medio de los datos. Además, al reiniciar el sensor, el valor de x cambia entre ± 10.

Después de estos desconcertantes resultados se han hecho pruebas con ruidos de diferentes magnitudes. Para ruidos medios, altos y muy altos se han obtenido datos muy relevantes. Ya que dependiendo del nivel del ruido se obtiene una salida de hasta x±50. Aunque pueda llamar la atención, este sensor en concreto, también produce resultados de un voltaje más bajo que el actual al detectar un ruido fuerte. En el siguiente ejemplo se pueden ver los datos recibidos al detectar un ruido fuerte.
{80, 80, 81, 80, 79, 80, 82, 79, 84,108, 62, 73, 88, 85, 79, 81, 80, 82, 82, 80, 80, 81}

Visto que no vamos a tener problemas con los ruidos de gran intensidad se han hecho pruebas con ruidos débiles y medios, por ejemplo, a esta categoría pertenecería el ruido que se produce al abrir o cerrar una ventana.

Debido a que no se estabiliza la salida del sensor, no se han podido obtener datos relevantes ante ruidos débiles por ahora. Pero cambiando o moviendo la localización de los cables entre el sensor y la TAD, sin hacer ruido claro, se ha percibido una variación en la salida del sensor, ¿Pero cómo puede ser esto posible si los cables están estañados en ambos extremos? Después de investigar, se ha llegado a la conclusión, de que los cables utilizados son tan finos que son muy

sensibles ante interferencias de otros cables o cualquier otro aparato electrónico cercano, después inspeccionar los cables, se ha comprobado que efectivamente solo tienen 2 o 3 pelos de cobre por cable. Se han cambiado los cables por otros más gruesos y se ha conseguido que el sensor se estabilice en un valor concreto cuando no hay ruido con un error de ± 1.

Ahora sí que podemos distinguir ruidos medios con claridad, que producen una salida de x ± 3, pero los bajos no producen ninguna salida ¿Hemos llegado a los límites del sensor o queda algún otro detalle? Como los datos recibidos en la TAD están siendo enviados por el puerto USB ¿Esto podría hacer que se estuvieran perdiendo datos debido a la lentitud de una operación de escritura en un puerto USB? Ya que el código que se ejecuta en una tarjeta Arduino es todo secuencial, si escribimos en un puerto, solo hacemos eso hasta que terminemos de escribir y mientras tanto no podemos atender al sensor. Para comprobar si el comportamiento del sensor está siendo alterado por la operación de escritura en el puerto USB, se ha escrito un programa, para que se encienda un led si se detecta algún ruido por minúsculo que sea, ya que el tiempo necesario para encender un led es despreciable. Como resultado de esta prueba, se ha llegado a la conclusión de que el comportamiento del sensor mejora muy ampliamente si no envía los datos por el puerto USB, llegando a detectar ruidos muy bajos.

En la arquitectura diseñada inicialmente, constaba muchos sensores y actuadores conectados a la misma TAD Arduino, pero viendo que solo una operación de salida al puerto USB, decrementa el rendimiento de este tipo de sensor, es impensable esta arquitectura. Aun así, se ha probado utilizar el sensor de sonido más otro sensor digital que requiere muchos cálculos en la misma TAD y se ha comprobado que casi no detecta ni los ruidos fuertes. Por lo que se puede confirmar que el rendimiento del sensor de sonido es muchísimo mayor cuantas menos operaciones tenga que hacer la TAD. Esto es debido a que este sensor, solo emite una salida en el momento exacto que la recibe del entorno físico, al igual que todos los sensores analógicos, y si en ese momento no tomamos una muestra de la señal, estamos perdiendo información. Como al aumentar el número de operaciones de la TAD, estamos reduciendo la frecuencia de muestreo, cuantas más operaciones le asignemos a la TAD más información perderemos.

Después de estas pruebas se ha llegado a la conclusión de que no van a poder estar todos los sensores analógicos conectados a la misma TAD. Los sensores digitales estarán conectados a la TAD central, ya que no necesitamos leer los valores tan rápido como en los sensores analógicos. Los sensores analógicos estarán conectados cada uno a una TAD dedicada exclusivamente a ese sensor que a su vez estará conectada a una TAD central, cuando la TAD de cada sensor detecte un dato relevante lo mantendrá y lo emitirá a la TAD central durante un

tiempo para que le dé tiempo a verlo. Para transmitir la información entre las TAD de cada sensor y la TAD centran se hará en voltajes analógicos igual que los sensores analógicos, por esa razón para la TAD central las TADs de cada sensor son completamente invisibles, además se utilizaran exactamente los mismos cables para la conexión que en un sensor analógico. Las TADs que se utilizarán para cada sensor analógico serán las más pequeñas y baratas que tiene Arduino con la versión *pro mini,* con un coste de 2.63€ por placa es una solución que nos podemos permitir en el proyecto. Además al dedicar exclusivamente una TAD a un sensor, según la especificación de la función de lectura analógica, podemos leer casi 10.000 valores por segundo.

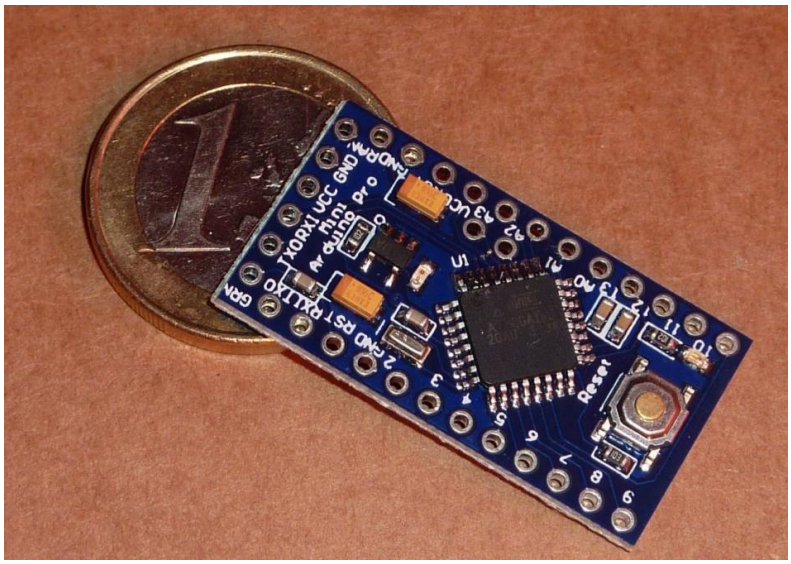

Ilustración 13. Arduino pro mini.
Fuente: http://giltesa.com/

Una vez implementada la arquitectura del nuevo sistema, se ha procedido a realizar pruebas de nuevo, esta vez también recibiendo los datos por cable USB mediante un PC. Las pruebas han sido satisfactorias pudiendo distinguir ruidos de hasta cuatro magnitudes diferentes.

El sensor de sonido quedará implementado como se ha descrito hasta ahora, pero se han investigado métodos para mejorar los resultados que quedan detallados a continuación. Para entender mejor el funcionamiento, si el voltaje sin ruido es de 0.3 voltios, como tenemos una resolución de 0.0049 voltios, estamos interpretando del sensor de sonido:

- 0.30098 voltios para ruidos bajos
- 0.30147 voltios para ruidos medios
- 0.30196 voltios para ruidos altos
- 0.30245 o más para ruidos muy altos

Por ejemplo para un voltaje de 0.3021274896 voltios se tomara el valor más cercano que es 0.30196 voltios, es decir un ruido alto.

Es decir, estamos interpretando solo cuatro valores, pero entre estos valores hay muchos más, que vienen cuantificados por la resolución del sensor. Como el sensor no tiene ningún tipo de documentación, que quizás de se puede justificar por su bajo precio, no podemos saber cuál es su resolución.
Para obtener más de los cuatro valores actuales existen tres soluciones

- Utilizar una TAD con más precisión de medida, con Arduino Due, pasaríamos de 10 bits a 12 bits de resolución.
- Comprar un sensor que este pre amplificado. Lo normal es que todos los sensores con circuito estén amplificados. Pero en este caso no es así.
- Amplificar la salida del sensor con un amplificador

La solución más factible es utilizar un amplificador, ya que es la de menor coste. Pero no se implementará esta solución, debido a los resultados aceptables obtenidos por la solución anterior y por la falta de tiempo, ya que el funcionamiento incorrecto de este sensor ha producido una desviación significativa respecto al tiempo previsto.

5.1.1.11 Detección de humo

Para detectar el humo y otros gases se usará el sensor MQ-2. Es un sensor electro-químico, que es sensible a un rango de gases determinado.

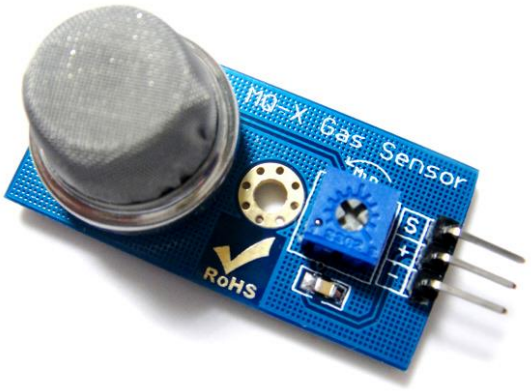

Ilustración 14. Sensor de gas MQ-2
Fuente: http://www.yidian.net.cn/

Este sensor tiene una salida de tipo binario, cuando el los gases sobrepasan un determinado umbral, el sensor se dispara emitiendo un voltaje de cinco voltios. Este umbral se puede establecer manualmente utilizando un potenciómetro del sensor.

Este sensor puede detectar un conjunto amplio de gases, entre los más significativos, se encuentran los siguientes:

- **Liquefied petroleum gas:** (LPG) es la mezcla de gases licuados, entre ellos se encuentran el butano y el propano. Se utiliza, entre otros usos, como combustibles de estufas.
- **Metano:** (CH4) el gas natural está compuesto en más de un 90% por metano. También tiene un uso amplio en los hogares.
- **Monóxido de carbono:** (CO) "El monóxido de carbono es un gas inodoro, incoloro y altamente tóxico. Puede causar la muerte cuando se respira en niveles elevados. Se produce por la combustión deficiente de sustancias como gas, gasolina, keroseno, carbón, petróleo, tabaco o madera. Las chimeneas, las calderas, los calentadores de agua o calefactores y los aparatos domésticos que queman combustible, como las estufas u hornallas de la cocina o los calentadores a queroseno, también pueden producirlo si no están funcionando bien".

Fuente: http://es.wikipedia.org/wiki/Mon%C3%B3xido_de_carbono

- **Humo: "**El humo es una suspensión en el aire de pequeñas partículas sólidas que resultan de la combustión incompleta de un combustible. Es un subproducto no deseado de la combustión"
  Fuente: http://es.wikipedia.org/wiki/Humo

A continuación se puede apreciar como varia la salida del sensor en función de los gases detectados. Cuando el sensor sobrepasa en determinado umbral definido entre 200 partes por millón (ppm) o 1000 ppm se dispara.

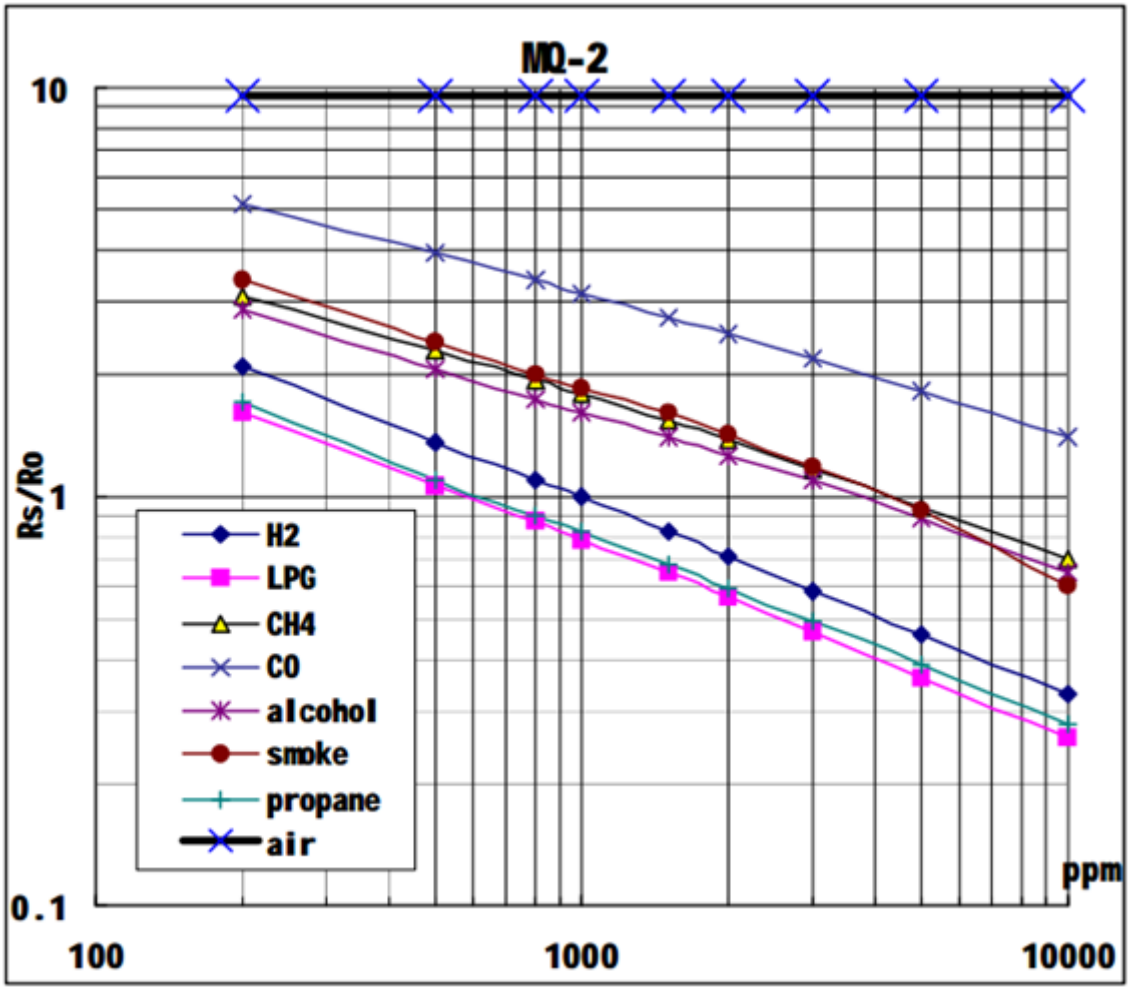

Ilustración 15. Comportamiento de la salida del sensor MQ-2 ante diferentes gases.
Fuente: http://www.seeedstudio.com/depot/datasheet/MQ-2.pdf

Como en casi todos los sensores, la salida del sensor varía en función de la humedad y la temperatura. En este sensor concreto, el cambio de humedad puede suponer un gran problema porque puede producir un error muy alto por sus características. Para solucionar este problema, el sensor incorpora integrado

un calentador, para mantenerse constante en temperatura y humedad. El calentador encuentra su estado óptimo en 50 grados Celsius, y en función de la temperatura del ambiente, tardará más tiempo o menos en alcanzarlos, por esa razón, en los primeros 5 minutos del arrancado del sistema tendremos que ignorar los datos del sensor o en caso contrario se producirían falsos positivos.

Este sensor no tiene la funcionalidad de estar emitiendo un voltaje de cinco voltios en cuanto se alcanza el umbral durante unos segundos, como lo hace el detector de movimiento, por esa razón si el sensor genera una señal muy corta, no la podremos detectar por el problema de tener una frecuencia de muestreo baja, como explicamos en el apartado del sensor de sonido. Por esa razón se ha optado por dedicar una tarjeta exclusiva para este sensor, al igual que ocurrió con el sensor de sonido. Será esta tarjeta quien se encargue de esperar los 5 primeros minutos sin validar los datos del sensor y cuando detecte una señal la seguirá emitiendo durante entre dos o tres segundos, para que le dé tiempo a la tarjeta principal detectarla. También se ha incorporado a esta tarjeta un pequeño zumbador, para emitir una señal acústica en cuanto detecte una señal.

Para probar este sensor en primer lugar se ha expuesto a un ambiente con metano y propano, dejando la válvula de gas de un mechero abierta durante cuatro segundos, situándose el sensor a 50 centímetros por encima de la fuente de gas, la detección ha sido satisfactoria.

En las mismas condiciones, como resultado de la combustión de una hoja de papel de 5cm x 5cm, también se han producido los resultados esperados.

5.1.1.12 Detección de vibración

Para detectar la vibración, en puertas y ventanas, se utilizara el acelerómetro digital ADXL345.

Ilustración 16. Acelerómetro ADXL345.
Fuente: http://www.china-electronics-gadgets.com/

Un acelerómetro es un sensor capaz de medir las fuerzas gravitatorias que se aplican a su superficie, éste en concreto, es un acelerómetro de tres ejes x, y, z. Entonces nos indicara para cada uno de sus ejes, si tiene una fuerza G positiva o negativa. Para entender mejor este concepto basta con pensar cómo se comportaría una bola introducida en un cuerpo cubico como el siguiente.

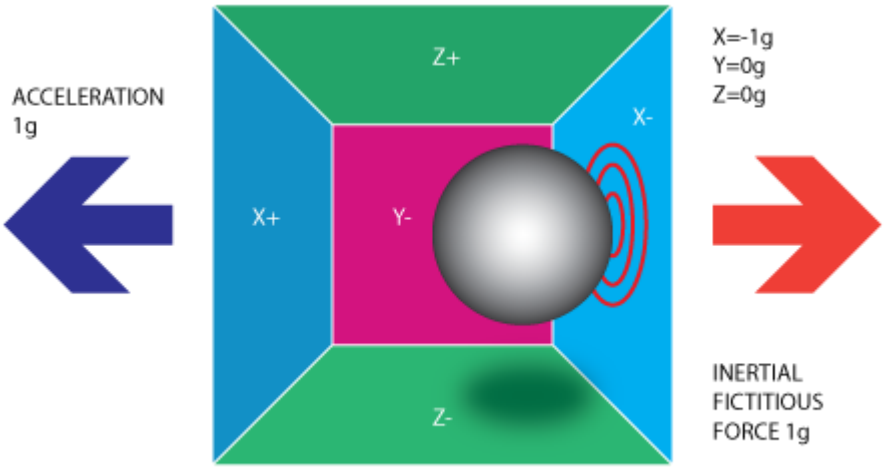

Ilustración 17. Comportamiento de un acelerómetro.
Fuente: http://www.starlino.com/

En nuestro caso, no necesitamos medir con precisión la fuerza medida en cada eje, si no si el acelerómetro está siendo sometido a fuerzas en general. Entonces, en el algoritmo diseñado para leer los datos del acelerómetro, no tiene en cuenta cual es el eje en el que se han detectado las fuerzas.

Se toman muestras por pares de cada eje, una muestra en un instante t y otra en t+1. Si para cada eje, la diferencia entre la muestra t y la t+1 supera un determinado umbral, se suman todas las fuerzas de la siguiente manera.

```
diferencia = abs(x-xAnt) + abs(y-yAnt) + abs(z-zAnt);
```

Si la diferencia supera un determinado umbral, la tendremos en cuenta como una vibración positiva.

La razón para que comparen las fuerzas de cada eje con un instante anterior, es para omitir la posición del sensor, si no lo hiciésemos así y utilizásemos una constante para la comparación, tendríamos que localizar el sensor en una orientación muy precisa.

Para definir los umbrales del algoritmo, se han realizado pruebas, para que sea suficientemente preciso como para detectar vibraciones de intensidad baja y no genere falsos positivos.

Para este sensor también se ha optado por dedicarle exclusivamente una tarjeta, ya que la reducción de la frecuencia de muestreo reduce la posibilidad de la detección.

5.1.1.13 Detección de mal funcionamiento de sensores

Uno de los objetivos del proyecto, es que el sistema tenga la capacidad de detectar cuando un sensor no está funcionando de la manera esperada, ya sea porque no está conectado o por presentar anomalías.

Para detectar un comportamiento extraño en los sensores analógicos y digitales, en este proyecto adoptaremos una solución sencilla. Si no están leyendo datos en un rango determinado, es que están desconectados o se están comportando de manera anómala y entonces se enviará una señal de error. En cambio, para detectar anomalías en sensores binarios no lo podemos hacer así, ya que cuando un sensor binario está funcionando correctamente puede estar escribiendo un voltaje de cero voltios. Además para leer los sensores de este tipo hemos estado utilizando una función que devuelve simplemente alto o bajo, y si leemos las notas de esta función, encontramos lo siguiente. “Si el pin no está conectado a algo, digitalRead() puede regresar HIGH o LOW (y esto puede cambiar

aleatoriamente).” Por lo tanto queda claro que no vamos a poder detectar la ausencia de señal mediante los pines digitales de Arduino.

Ahora vamos a estudiar la posibilidad de detectar señales digitales mediante los pines analógicos de Arduino, para ello utilizamos la función analogRead. Si leemos la documentación de esta función, encontraremos información muy útil para este caso. “Si la entrada analógica que vamos a leer no está conectada a nada, el valor que devolverá la función analogRead() fluctuará dependiendo de muchos valores (los valores de otras entradas analógicas, que tan cerca está tu mano de la entrada en cuestión, etc.)”. Es decir, si conectamos una salida de un sensor binario a un pin analógico de entrada de Arduino ocurrirá lo siguiente.

- Si el sensor está conectado y emitiendo un voltaje de alto, recibiremos un valor constante en el rango [0,1023]
- Si el sensor está conectado y emitiendo un voltaje de cero, recibiremos un valor constante en el rango [0,1023]
- Si el sensor esta desconectado o está funcionando mal, recibiremos un voltaje aleatorio en el rango [0,1023]

Entonces lo único que necesitamos hacer es detectar cuando estamos ante cada una de estas tres situaciones. Para distinguir las tres posibles situaciones se compara el valor actual con nueve muestras anteriores, de estas diez muestras se toman dos parámetros, el valor medio y la máxima diferencia para cada valor con todos los demás. Si la media está entre un rango de valores que es considerado de alto o bajo y además la máxima diferencia es menor que un determinado valor, entonces se considera un valor de alto o bajo del sensor, en otro caso se considera error. Además de esto, solo enviamos la señal de error cuando estamos en error más de tres segundos, esto es debido a que la transición del estado bajo al estado alto puede causar un estado de error debido a la transición de voltajes de un estado a otro. Para transportar la información haremos las siguientes definiciones.

- 0: Sensor en estado bajo.
- 1: Sensor en estado alto.
- 2: Error.

5.1.1.14 Relés

Un relé es un dispositivo electromecánico que tiene la función de interruptor.

Al igual que para los sensores, en este proyecto no utilizaremos únicamente el elemento relé, si no que utilizaremos un módulo de relé, que contiene todos los elementos necesarios para su fácil utilización. El elemento rectangular y grande que se puede apreciar en la imagen es el elemento relé, y el conjunto es el módulo que utilizaremos.

Ilustración 18. Módulo de relé usado para el proyecto.
Fuente: http://www.elecrow.com/

Por el extremo izquierdo se pueden ver las conexiones para la tarjeta Arduino, dos pines para la alimentación y uno para la señal, cuando enviemos una señal de voltaje alto por este pin, la corriente alterna fluirá entre dos de las conexiones de la derecha y cuando enviemos una señal de bajo fluirá entre otras dos. Eso significa que hace función de interruptor de tipo conmutador, pero nosotros solo lo utilizaremos como un interruptor simple, utilizando únicamente dos, de las tres conexiones del lado izquierdo. Para encenderlo desde Arduino lo haremos de la siguiente manera.

```
digitalWrite(relePin, HIGH);
```

5.1.2 Tarjeta de adquisición de central de cada nodo

La tarjeta central de cada nodo es la encargada interaccionar con todos los elementos a los que está conectada y comunicarse con el ordenador central.

Al igual que todos los programas hechos para un microcontrolador, el programa de esta tarjeta tiene que tener un bucle o lazo principal, que es el que se estará

repitiendo continuamente desde que se enchufe el sistema hasta que se apague. Todos los programas de Arduino tienen que tener una función de inicialización, donde se inician los sensores, variables y un bucle principal. En la ilustración 19, se muestra el bucle principal tal y como está en el código.

- **LeerDatosSensores**: Se encarga de leer los datos de todos los sensores y guardarlos en sus respectivas variables.
- **RecibirDatosCentra**l: Se encarga de recibir los comandos enviados por el ordenador central por el puerto serial. Como el puerto serial está conectado a un módulo XBee esta comunicación es por medio inalámbrico.
- **ActualizarRele:** Actualiza el estado del relé en función de los comandos recibidos en la función anterior.
- **EnviarDatosCentral**: Envía los datos de todos los sensores ya leídos por medio inalámbrico al ordenador central.

Como podemos ver, los datos de los sensores están siendo enviados en tiempo real al ordenador central. Para los sensores que generan alarmas, la alarma producida es enviada durante tres segundos repetidamente.

```
void loop()
{
   leerDatosSensores();
   recibirDatosCentral();
   actualizarRele();
   enviarDatosCentral();

}
```

Ilustración 19. Lazo principal de la TAD de cada nodo.

5.1.3 Diseño del nodo de cada estancia

Como ya sabemos, un nodo está formado por todos los elementos necesarios para que una estancia interaccione con los sensores y actuadores correctamente. A continuación, se muestra como están conectados todos los elementos de cada nodo.

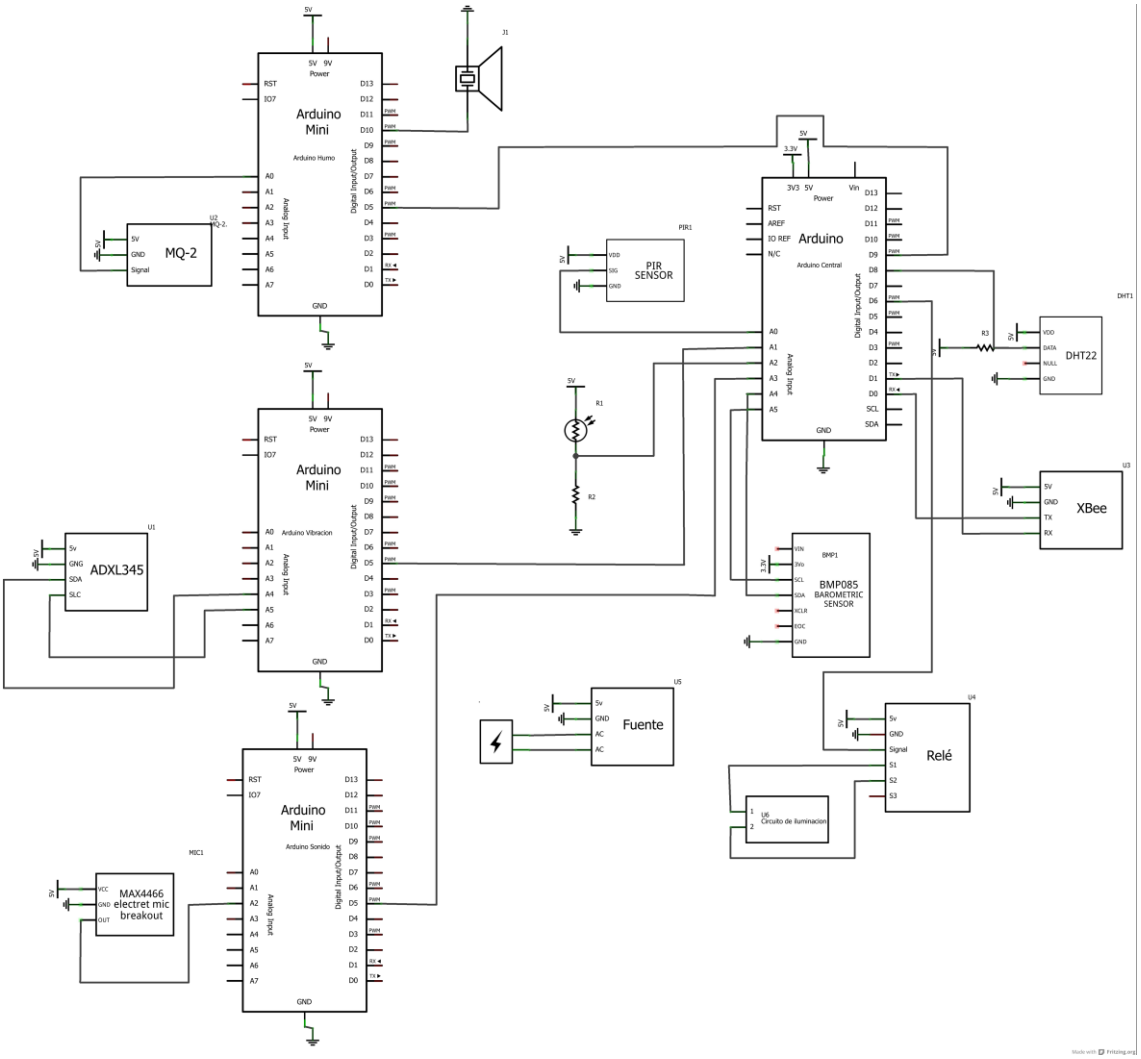

Ilustración 20. Diseño electrónico del nodo

En este esquema, únicamente se utiliza una fuente de alimentación. Pero en el proyecto se han utilizado dos fuentes, ya que una no proporcionaba la suficiente potencia. Las dos fuentes utilizadas, han sido de 5 voltios y 1.5 amperios, debido a disponibilidad, por lo tanto, con una única fuente de 5 voltios y 3 amperios es suficiente.

5.1.4 Diseño del relé portátil

Este relé, no forma parte de ningún nodo, si no que el mismo es un nodo ya que tiene la capacidad de comunicarse por sí mismo con el ordenador central. Tiene la finalidad de poder controlar cualquier aparato electrónico al que se conecte, siempre y cuando este en el rango de comunicaciones del ordenador central. El esquema queda detallado a continuación.

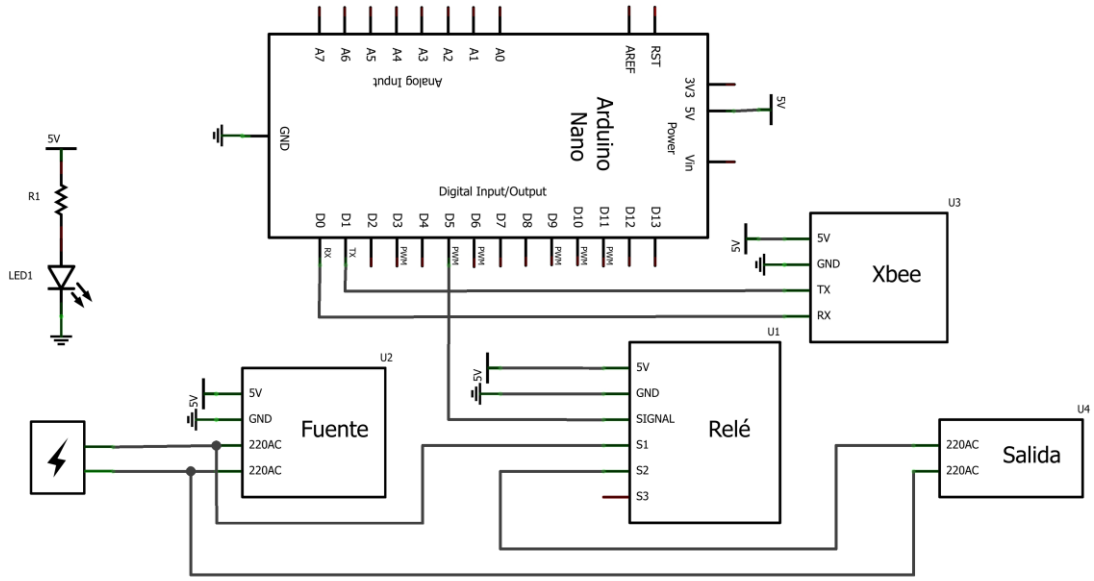

Ilustración 21. Diseño electrónico del relé portátil.

5.1.5 Comunicación entre cada nodo y el ordenador central

En este capítulo describiremos los dispositivos que utilizamos para que se produzca la comunicación y en que formato se intercambian los datos.

5.1.5.1 XBee

Para comunicarnos entre cada nodo y ordenador central lo haremos por medio inalámbrico utilizando módulos *Xbee pro series 1*. Sus características se pueden ver en la ilustración 22. Para que la comunicación se produzca únicamente hay que conectar el módulo a un puerto serial, y toda la información que le llegue por el puerto será transmitida por medio inalámbrico. Este dispositivo funciona bajo el estándar IEEE_802.15.4[23], que es un subconjunto del protocolo ZigBee[24].

Este es un dispositivo que forma redes de comunicación *punto a multipunto,* lo que significa que un dispositivo puede comunicarse con más de otro simultáneamente siempre que estén a su alcance. Pero para el caso de este proyecto, lo realmente apropiado es utilizar una red de tipo *mesh*[25]. Esta red tiene las características de una red *punto a multipunto,* pero con el añadido de que

[23] IEEE_802.15.4: http://es.wikipedia.org/wiki/IEEE_802.15.4
[24] ZigBee: http://es.wikipedia.org/wiki/ZigBee
[25] Red mesh: http://es.wikipedia.org/wiki/Red_inal%C3%A1mbrica_mallada

puede comunicarse con un dispositivo aunque no esté en su rango, pero si en el rango de otros nodos de la red. Esta red la implementan automáticamente los dispositivos *Xbee pro series 2,* que sí que implementan completamente el protocolo ZigBee, pero no se han adquirido porque su coste es más elevado que los de tipo uno, y además habría que adquirir un dispositivo más de los necesarios para que actué como coordinador de la red.

Platform	XBee-PRO® 802.15.4 (Series 1)
Performance	
RF Data Rate	250 kbps
Indor/Urban Range	300 ft (100 m)
Outdoor/RF Line-of-Sight Range	1 mi (1.6 km)
Transmit Power	60 mW (+18 dBm)*
Receiver Sensitivity (1% PER)	-100 dBm
Features	
Serial Data Interface	3.3V CMOS UART
Configuration Method	API or AT Commands, local or over-the-air
Frequency Band	2.4 GHz
Interference Immunity	DSSS (Direct Sequence Spread Spectrum)
Serial Data Rate	1200 bps - 250 kbps
ADC Inputs	(6) 10-bit ADC inputs
Digital I/O	8
Antenna Options	Chip, Wire Whip, U.FL, & RPSMA
Networking & Security	
Encryption	128-bit AES
Reliable Packet Delivery	Retries/Acknowledgments
IDs and Channels	PAN ID, 64-bit IEEE MAC, 12 Channels

Ilustración 22. Características del módulo Xbee pro series 1 adquirido
Fuente: http://www.digi.com/

Cuando un módulo XBee le llega una trama para enviar o la recibe, actúa en función de la configuración que tenga el módulo. A continuación de estudian los parámetros de configuración que nos afectan.

- **PAN ID (ID)**: Es un identificador de dos bytes que define en que red está el nodo. En nuestro caso todos los nodos pertenecen a la misma red, por lo tanto tendrán el mismo ID. Este parámetro se utiliza para evitar interferir con redes XBee cercanas.
- **Source Address (SA):** Es la dirección de identificación de cada nodo. No es único, más de un nodo pueden tener la misma Souce Address.
- **Destination Address (DA):** Es la dirección a la que enviara los paquetes cada nodo.

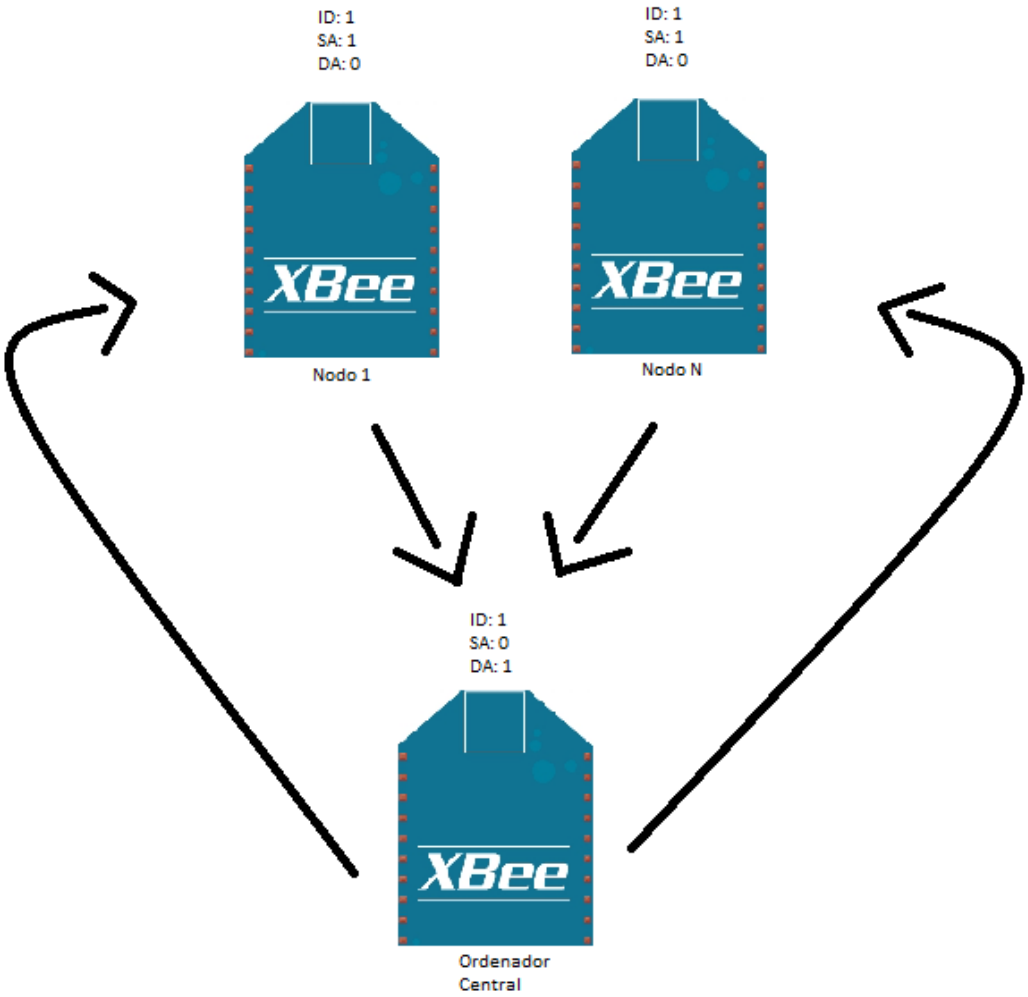

Ilustración 23. Relación entre los módulos Xbee de los nodos y el ordenador central.

- **Retries and acknowledgments:** Es un parámetro que permite activar/desactivar este modo. Cuando está activado, después de enviar una trama, espera a su confirmación, si no la recibe, vuelve a enviar la trama hasta un número concreto de intentos que se especifican en otro parámetro. Como se puede ver en el capítulo 5.1.3, nosotros estamos enviando de forma continuada la información de todos los sensores y alarmas repetidamente, para nuestro caso, no tiene interés activar este modo ya lo único que podría hacer es ralentizar las comunicaciones.
- **Parity**: Este parámetro permite desactivar o activar este modo. Sirve para verificar que una trama es correcta utilizando bit de paridad. Como veremos más adelante, no utilizaremos este método ya que es demasiado débil para nuestras necesidades. El bit de paridad es el único método que tienen esta versión de Xbee ya implementado.

5.1.5.2 Diseño de la trama

Para que los datos sean interpretados correctamente por el ordenador central y por los nodos, necesitamos definir un formato de trama, que queda detallado a continuación y se puede ver en la ilustración 24.

- **Sensores binarios**: Como hemos visto, estos sensores solo tienen tres posibles estados, *si*, *no* o *error*. Utilizaremos un Byte, que podría definir hasta 255 estados, pero debido a que el Byte es la unidad mínima transportable, hay que enviar al menos uno en cada comunicación.
- **Sensores analógicos**: Estos sensores producen una salida entre [0,1023]. Utilizaremos dos bytes ya que es más que suficiente para representar los posibles valores. Para algunos sensores analógicos, como el de luz, hemos *remapeado* la salida para que el rango de [0,1023] lo conviertan en otro rango más reducido. Aunque en estos sensores no necesitemos utilizar los dos bytes, los utilizaremos, para mantener un estándar para cada tipo de sensor y que la posible ampliación de sensores no modifique el tamaño.
- **Sensores digitales**: Los sensores digitales generan números reales, para transportarlos se ha optado por utilizar dos bytes para la parte entera y uno para la parte decimal, pudiendo representar números en el rango [-32767.99, 32767.99]. El primer bit se usa para el signo. Existen métodos más complejos y eficientes que este para representar números reales, como utilizar mantisa y exponente, pero no se ha realizado así, para simplificar la comunicación.
- **Inicio de trama**: Para que se pueda detectar el inicio de la trama se ha optado por utilizar dos bytes con el valor 255. Se ha elegido este valor ya que es muy difícil que en medio de la trama se produzcan estos valores, ya que supondría el rango máximo de lectura de un sensor, tanto para el caso de los sensores analógicos como de los digitales.
- **Código TAD**: después de la constante del inicio de la trama, se envía un byte con un código que identifica al nodo que envía la trama. En función de este código, quien reciba la trama sabrá que estructura tiene en cuanto a número de sensores.
- **Fin de trama**: ya que el módulo de Xbee adquirido no nos puede proporcionar la certeza de que la información enviada llegue al destino sin alteraciones, se ha decidido utilizar un código detector de errores. Si no lo utilizáramos, hay que tener en cuenta que la diferencia entre la detección de fuego y la no detección es un bit a cero o a uno, y debido a interferencias esta información puede ser fácilmente alterada. Por esa razón se ha

decidido implementar CRC32, este algoritmo creara un resumen de la trama entera en 4 bytes, que serán enviados como últimos bytes de trama.

A continuación se muestra un ejemplo de trama, tal y como sería para el caso de los sensores utilizados en este proyecto.

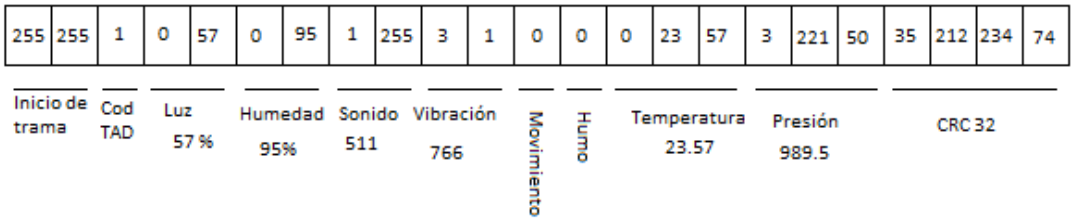

Ilustración 24. Ejemplo de una trama enviada desde un nodo hasta el ordenador central.

Se ha probado la comunicación, y se ha conseguido una velocidad efectiva de 300 Bps. De media, tenemos una trama corrupta cada once segundos, diremos que una trama es corrupta cuando el CRC no corresponde con el contenido de la trama. Con esos datos podemos calcular que de cada 143 tramas tenemos una corrupta, con lo que el 0.69% de las tramas son corruptas. Después de ver estos datos podemos concluir que ha sido una opción muy acertada incluir un código detector de errores.

Para enviar la información en sentido contrario, es decir del ordenador central hasta los nodos, no se hace de forma continuada, si no que en momentos muy concretos, cuando el sistema pida encender algún relé. Por ello se ha ideado otra solución diferente que para el caso anterior. En este caso, como la información a enviar es muy reducida, un número de identificación del nodo, más el estado, es decir dos bytes, no tiene sentido utilizar un código detector de errores dos veces más largo que la propia trama, por esa razón, para comprobar la veracidad de los datos, se ha optado por enviar la misma orden repetida durante tres segundos.

Cuando se recibe una trama, se toman las diez siguientes tramas, si son exactamente iguales se acepta, si no, se repite la operación. Como se envían del orden de cien tramas por segundo, hay muchas oportunidades para verificar.

Para las conexiones de ambos sentidos, los dispositivos Xbee, permiten fijar una tasa en baudios, para modificar la velocidad en la que se envían los datos. Este número normalmente puede variar entre 480 y 115.000 baudios, cuanto más alto sea el número, más bits se recibirán por segundo, pero existen más posibilidades de error, la tasa de baudios elegida es de 9600 bauds.

5.2 Arquitectura del ordenador central

En este capítulo describiremos cual es la arquitectura tanto hardware como software del ordenador central.

5.2.1 Arquitectura Hardware del ordenador central

El ordenador central no tiene una interfaz para interaccionar con el módulo Xbee y con el modem GSM, si no que necesita de una tarjeta que haga de intermediario, a continuación queda ilustrado el esquema del ordenador central.

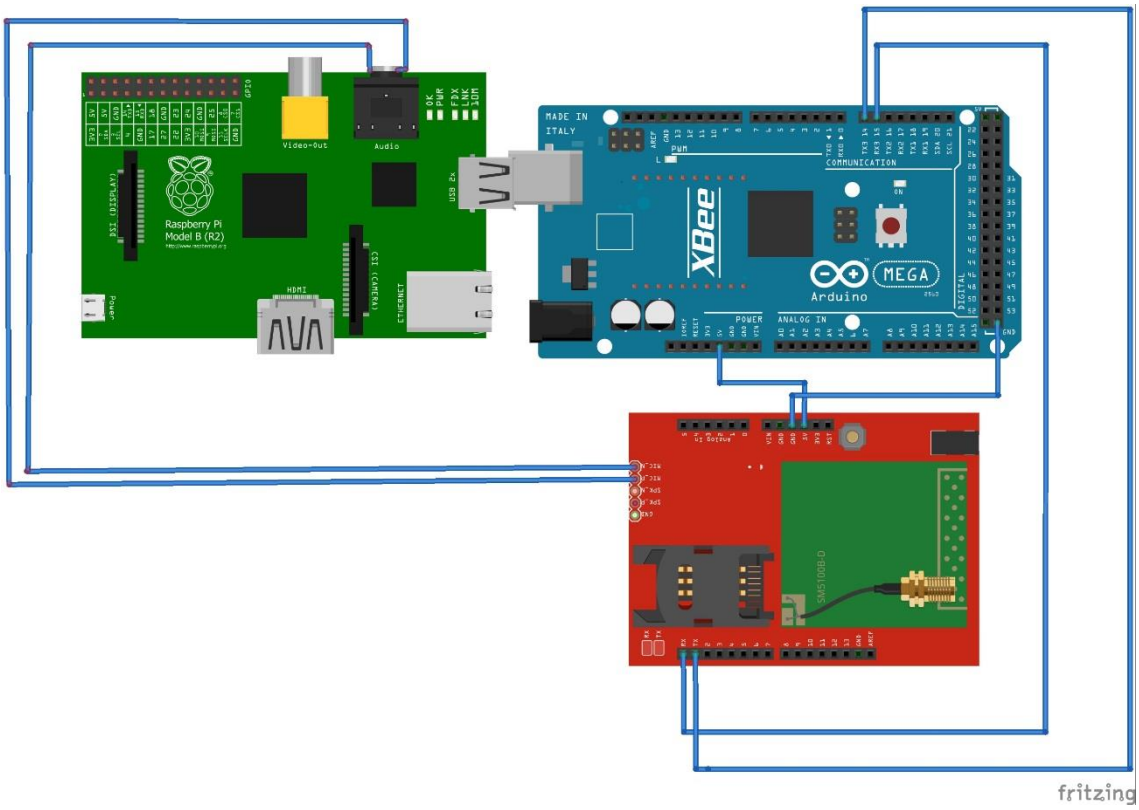

Ilustración 25. Diseño hardware del ordenador central.

Puede llamar la atención el uso de una tarjeta tan grande como Arduino Mega, pero, esta tarjeta es la única que tiene más de un puerto serial, que en este caso son necesarios ya en esta tarjeta se ponen en común 3 canales de comunicación, Xbee el ordenador central y el modem GSM.

Es verdad que para este esquema, podríamos haber omitido el uso de la tarjeta Arduino y conectar tanto el modem GSM como el Xbee al ordenador central, pero de esta manera es más sencillo ya que la tarjeta tiene mecanismos más eficientes para este caso.

Como se puede ver, también está conectada la salida de audio del ordenador central, con el micrófono del modem GSM.

5.2.1.1 Modem GSM

Un modem GSM es un dispositivo capaz de conectarse a una red GSM. Esta red, es un sistema estándar de telefonía digital que describe el funcionamiento de una red 2G. Cubre aproximadamente un 90% de todas las comunicaciones móviles. Un cliente conectado a través de este modem, puede realizar todas las tareas habituales de los teléfonos móviles, desde conectarse a internet, hasta hacer llamadas. En este proyecto, solo utilizaremos este modem para hacer llamadas y utilizaremos una tarjeta SIM con servicio activo para su funcionamiento.

Este dispositivo no se parece en nada respecto a los elementos trabajados anteriormente, para comunicarnos con él, utilizaremos un puerto serial, TX para transmitir y RX para recibir, enviaremos comandos en formato ASCII. Los comandos utilizados son el conjunto de comandos Hayes[26], es un lenguaje desarrollado por la compañía Hayes Communications, que se convirtió en estándar para comunicarse con los modems.

Cada modem tiene un módulo SIM diferente y en función la versión de este, los comandos pueden variar. La versión para nuestro caso es SIM900[27].

Los comandos que utilizaremos para poder realizar llamadas se describen a continuación.

- **AT**: Es el comando más básico de todos. Sirve para comprobar que la comunicación con el modem es correcta. Devuelve “OK”. Se utilizara en el inicio del sistema, en caso negativo se enviara una señal para encenderlo.
- **ATD**: Inicia una llamada de voz al número indicado. Por ejemplo “ATD 943000000;”. Si se establece la conexión no devuelve nada, en caso contrario muestra el error.
- **ATH**: Termina la conexión actual. Devuelve “OK”.
- **AT+CLCC**: Devuelve el estado de la llamada actual. El formato es un vector en código ASCII, donde cada elemento del vector es un parámetro en formato entero indicando un estado.

[26] Comandos de Hayes: http://es.wikipedia.org/wiki/Conjunto_de_comandos_Hayes
[27] Manual sim 900: http://www.datasheet-pdf.com/datasheetdownload.php?id=726721

Parameters

<idx>	1..7	Call identification number
This number can be used in +CHLD command operations		
<dir>	0	Mobile originated (MO) call
	1	Mobile terminated (MT) call
<stat>	State of the call:	
	0	active
	1	held
	2	dialing (MO call)
	3	alerting (MO call)
	4	incoming (MT call)
	5	waiting (MT call)
	6	disconnect
<mode>	Bearer/tele service:	
	0	voice
	1	data
	2	fax
<mpty>	0	Call is not one of multiparty (conference) call parties
	1	Call is one of multiparty (conference) call parties
<number>	String type(string should be included in quotation marks) phone number in format specified by <type>.	
<type>	Type of address	
<alphaId>	String type(string should be included in quotation marks) alphanumeric representation of <number> corresponding to the entry found in phone book.	

Ilustración 26. Formato de respuesta para el comando AT+CLCC.
Fuente: Manual sim 900

Para las pruebas realizadas con el mando AT+CLCC, el modem se ha comportado de la siguiente manera:

- Envío del comando antes de realizar una llamada. Respuesta nula, es decir un string de longitud cero.
- Envió del comando después de hacer una llamada y sin descolgarla. El parámetro *stat* en modo 2.
- Envió del comando mientras la llamada de voz esta activa. El parámetro *stat* en modo 0.

Con estos elementos ya tenemos toda la información necesaria para poder gestionar una llamada.

A continuación se describe en dos esquemas el sistema, en primer lugar entre que pares se producen las comunicaciones y después la lógica del sistema.

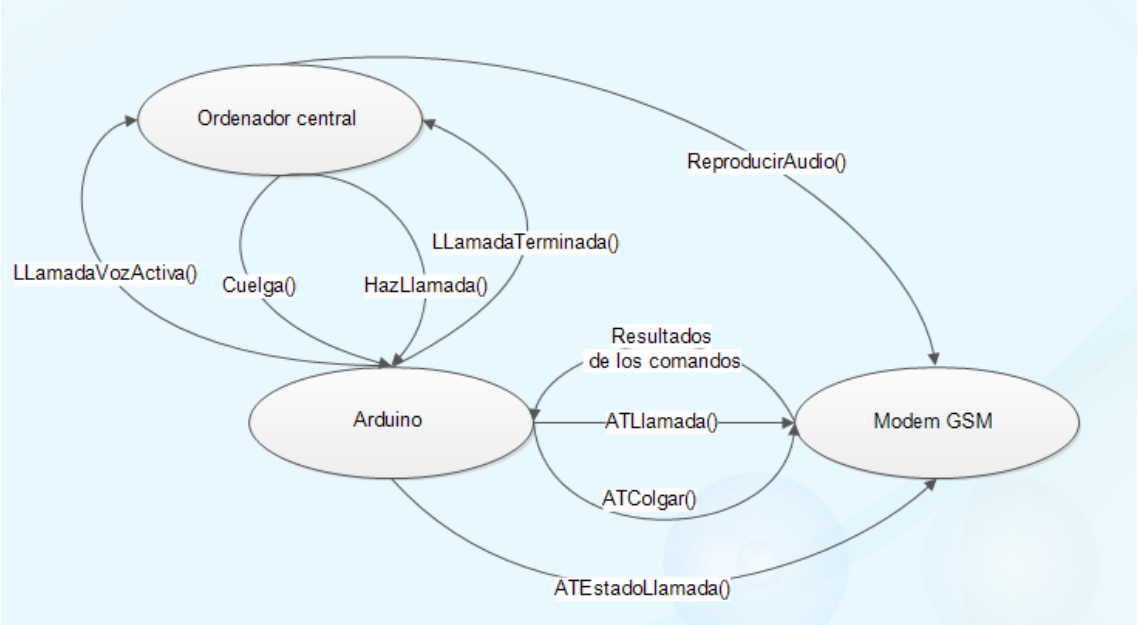

Ilustración 27. Descripción de las comunicaciones para realizar llamadas.

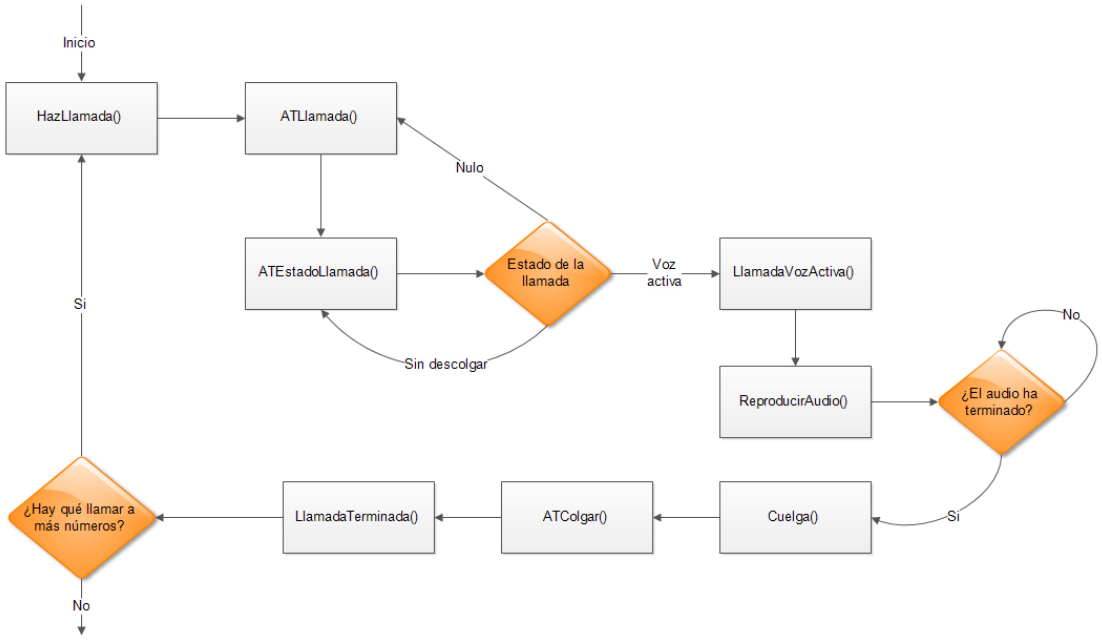

Ilustración 28. Descripción de la lógica para realizar llamadas.

El esquema que describe la lógica, únicamente muestra los pasos básicos y fundamentales, ya que se han excluido algunos comportamientos para simplificarlo, por ejemplo se ha definido en la tarjeta, una constante en segundos, para no dejar bajo ninguna circunstancia la llamada activa más de 30 segundos, ya que si no se detecta la señal de colgar por algún fallo, la llamada permanecería

activa indefinidamente. También en el caso de que se detecten más de tres llamadas nulas consecutivas, se reinicia el modem.

Todas las funciones indicadas en la ilustración 27, son funciones que se comunican por el envió de comandos, ya sea el envió de comandos constantes, por ejemplo, para el caso de entre el ordenador central y Arduino se envía dos bytes con el valor 64 para colgar la llamada, o comandos de Hayes desde Arduino hasta el modem GSM. Pero la función de ReproducirAudio(), tiene diferencias significativas, en primer lugar genera un audio utilizando el software festival, para ello se hace una llamada mediante la consola del sistema de la siguiente manera. *Filename* es la ruta del fichero que contiene el texto a convertir a voz.

```
subprocess.call('festival --language spanish --tts ' + filename, shell=True )
```

Cuando se ejecuta este comando, automáticamente se emite el audio por la salida de sonido de Raspberri Pi. El retardo para generar la voz es de unos dos segundos, por esa razón cuando la persona que reciba la llamada descuelgue el teléfono, no oirá nada hasta pasados dos segundos.

5.2.2 Diseño del software del ordenador central

El ordenador central obtiene los datos de los sensores en tiempo real, su labor es tenerlos en todo momento disponibles, para cuando reciba una petición. Además de transmitir órdenes a los relés.

Como ya sabemos, el ordenador central utiliza Python como lenguaje, a continuación se describe cual es la funcionalidad de cada clase y como interaccionan entre sí.

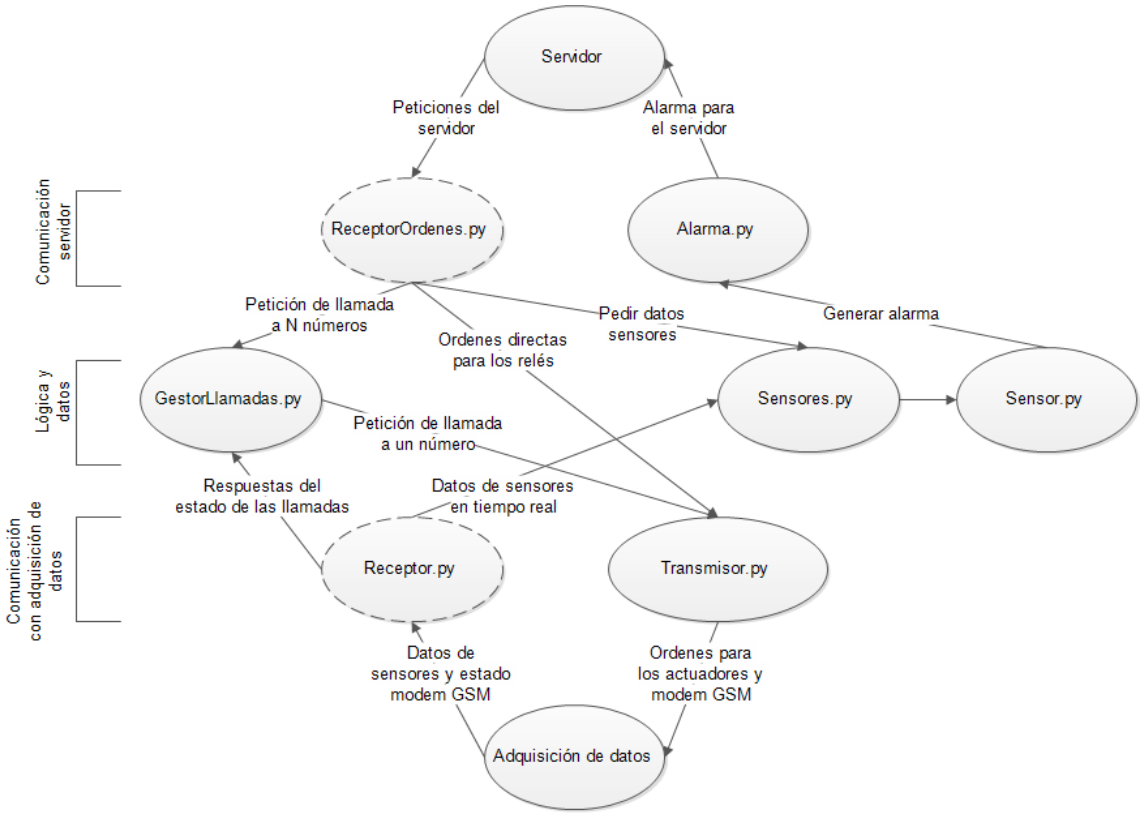

Ilustración 29. Interacción entre las clases para el ordenador central. Las clases con líneas discontinuas representan hilos de ejecución diferentes.

- **Sensores.py**: Se encarga de verificar, clasificar y guardar toda la información de sensores recibida. Utiliza para ello N instancias de la clase Sensor.py y las estructura en una lista. Se crea una única instancia de esta clase al comienzo del sistema que tienen en su posesión conjuntamente ReceptorOrdenes.py y Receptor.py.

  La función más significativa de esta clase es la función *clasificaDatos*, que se encarga de decodificar una trama, comprobar su integridad y clasificar sus datos. Para comprobar el crc32 se utiliza la librería *binascii*[28] de la API de Python.

[28] Python binascii: https://docs.python.org/2/library/binascii.html

```
def clasificarDatos(self,datos,CodTAD):
    #Dada una trama y el codigo de la TAD a la que pertenece la trama,
    #se almacena cada sensor de la trama en el objeto sensores siempre
    # y cuando el crc sea correcto

    tamTrama = self.caclularLong(CodTAD)

    crcPython=binascii.crc32(datos[0:tamTrama-4]) & 0xFFFFFFFF

    crcArduino=self.__decodCRC(datos[tamTrama-4:tamTrama])

    if crcArduino == crcPython:

        apuntador = 0

        for sensor in self.sensores:

            if CodTAD == sensor.getCodTAD():

                tamDato = sensor.getTamDato()

                if(sensor.getTipo() == 'Digital'):

                    sensor.setDato(self.__decodFloat(datos[(apuntador):(apuntador + tamDato)]))

                else:

                    sensor.setDato(self.__decodInt(datos[(apuntador):(apuntador + tamDato)],tamDato))

                apuntador = apuntador + tamDato

    else:

        msg('Se ha recibido una trama corrupta desde la TAD')
        return -1

    return 0
```

Ilustración 30. Función para clasificar los datos de los sensores del ordenador central.

Las funciones de decodificación que se pueden ver, como *decodFloat*, *dedodInt, y decodCRC,* lo que hacen es, dado un buffer de bytes devuelven el valor codificado en ese buffer. Están implementadas en la misma clase sensores.

- **Sensor.py:** Encargada de guardar la información de un sensor. Contiene todas las funciones necesarias para la manipulación de los datos de un sensor. Cuando el dato de un sensor se detecta como alarma, crea una nueva instancia de Alarma.py. Como las diferentes clases que acceden a los métodos de esta clase, lo hacen a través de una misma estancia compartida pero desde hilos diferentes, se utilizan secciones críticas para que la información no sea manipulada al mismo tiempo, para ello se utiliza la clase Lock[29].

- **Alarma.py:** Envía alarmas al servidor con un cliente socket, únicamente se instancia cuando se necesita enviar una alarma. Para no bloquear la adquisición de datos debido a la latencia de las comunicaciones, se ejecuta en un hilo de ejecución diferente.

[29] Mecanismo Lock de Python: https://docs.python.org/2/library/threading.html#lock-objects

- **ReceptorOrdenes.py**: Es un servidor socket, que está continuamente a la escucha para recibir peticiones del servidor, por esa razón se ejecuta en un hilo de ejecución nuevo. Para acceder a GestorLlamadas.py y Transmisor.py crea una nueva instancia de estos, para acceder a Sensores.py utiliza la instancia existente que dispone desde el inicio del sistema. Este servidor dispone de cinco comandos diferentes que quedan detallados en el capítulo *6.1 comunicaciones entre el ordenador central y el servidor.*

- **Receptor.py:** Es una clase que está continuamente en ejecución, se ejecuta en nuevo hilo. Lee de forma continuada toda la información entrante de la adquisición de datos. Para acceder a Sensores.py utiliza una instancia ya existente que dispone desde el inicio del sistema. Para acceder a GestorLlamadas.py crea una nueva instancia de esta clase.

- **GestorLlamadas.py:** Se encarga de gestionar todo el proceso para realizar llamadas telefónicas. Únicamente se crea una instancia cuando se necesita la clase, es decir existe una instancia de esta clase cuando se está en el proceso de realizar una o varias llamadas. Utiliza una cola para guardar las llamadas que están en espera y cuando es el turno de una llamada realiza todos los procesos necesarios para iniciarla como generar la voz.

- **Transmisor.py:** Se encarga de enviar órdenes a los relés y al modem GSM. Únicamente se crea una instancia de esta clase cuando hay que enviar una orden.

- **Constantes.py:** Fichero auxiliar para guardar constantes que afectan a más de una clase.

- **Msg.py:** Funciones auxiliares que se encargan de escribir información en el log del sistema.

5.2.3 Robustez en el ordenador central

El ordenador central tiene que tener la capacidad de recuperarse por si mismo y seguir funcionando en caso de sufrir un error. Por esa razón se han analizado cuidadosamente las *excepciones*.

Las excepciones más importantes son las generadas por las clases que tienen comunicación con la adquisición de datos o con el servidor.

Para la comunicación con la adquisición de datos, los problemas más comunes de excepciones surgen al abrir el puerto serial, cuando el ordenador central abre una comunicación por USB con la TAD. Ya que los nombres de los puertos USB de Raspberry pi, no se mantienen constantes siempre. Pueden variar desde "dev/ttyACM0" hasta "dev/ttyACM4". Por esa razón, cuando se detecta una excepción que indica que en el puerto serial no hay un dispositivo conectado, se busca entre todos los posibles nombres disponibles. De esta manera, en caso de que se desconecte la tarjeta Arduino accidentalmente y se conecte en otro puerto diferente al actual, también seguirá funcionando.

Además de eso, continuamente se mide la velocidad de los datos de los sensores, que también se puede visualizar de la interfaz Android. Cuando la velocidad es cero se reinicia el puerto serial, lo que provoca que se reinicie el software de la tarjeta Arduino.

Para la comunicación con el servidor, los problemas más comunes encontrados han sido cuando las comunicaciones se han cortado, por falta momentánea de conexión a internet o por algún otro error relacionado con exceder el tiempo límite de espera de la conexión. Cuando se produce una excepción de este tipo, para los clientes, se intenta la misma conexión con un máximo de tres intentos, para el servidor, se reinicia siempre y cuando se pueda controlar la excepción. Como posiblemente el servidor después de reinícialo dejaría el puerto ocupado hasta que se cierre automáticamente pasado un tiempo, la nueva conexión de servidor se abre reusando el puerto anterior de la siguiente manera.

```
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

Cuando en el ordenador central se produce una excepción no controlada, sea del tipo que sea, se ha decidido reiniciar el software completo. Es decir, que si se produce un error en una clase concreta y no se puede controlar, se reinician todas las clases. Se ha optado por esta opción, por la concurrencia del sistema, y por la manera que están inicializadas las clases (diferentes clases tienen la instancia de una misma clase), ya que reiniciar solo una clase no sería una tarea trivial.

Para reiniciar el software, se terminan todos los procesos Python, para asegurar la finalización de todos los hilos. A continuación, un *shell script* que se inicia con el sistema y se mantiene siempre en ejecución, es decir un *daemon*, se da cuenta de que el programa no está funcionando y lo ejecuta. Además de servir para casos de emergencia, este script es el encargado de poner en marcha el programa en el inicio del sistema.

```
while true
do

        if !(ps aux | grep "[p]ython" > /dev/null)
        then

                python /home/pi/projects/Central/main.py &

        fi

        sleep 5

done
```

Ilustración 31. Script ejecutado de forma continuada en el ordenador central.

También cabe mencionar que cada vez que se abre una nueva conexión serial con la tarjeta Arduino, el software de Arduino se reinicia. Este es un comportamiento que se produce en todas las tarjetas Arduino por defecto. Eso significa que cuando se reinicie el software del ordenador central, el de la tarjeta también se reiniciará y todos sus componentes conectados también.

Todos los eventos comunes, como posibles errores que se producen en el servidor, se guardan en un fichero de texto. Gracias a ello, después de un fallo del sistema se pueden analizar sus posibles causas. También es de mucha utilidad, para ver el comportamiento del sistema después de realizar un cambio.

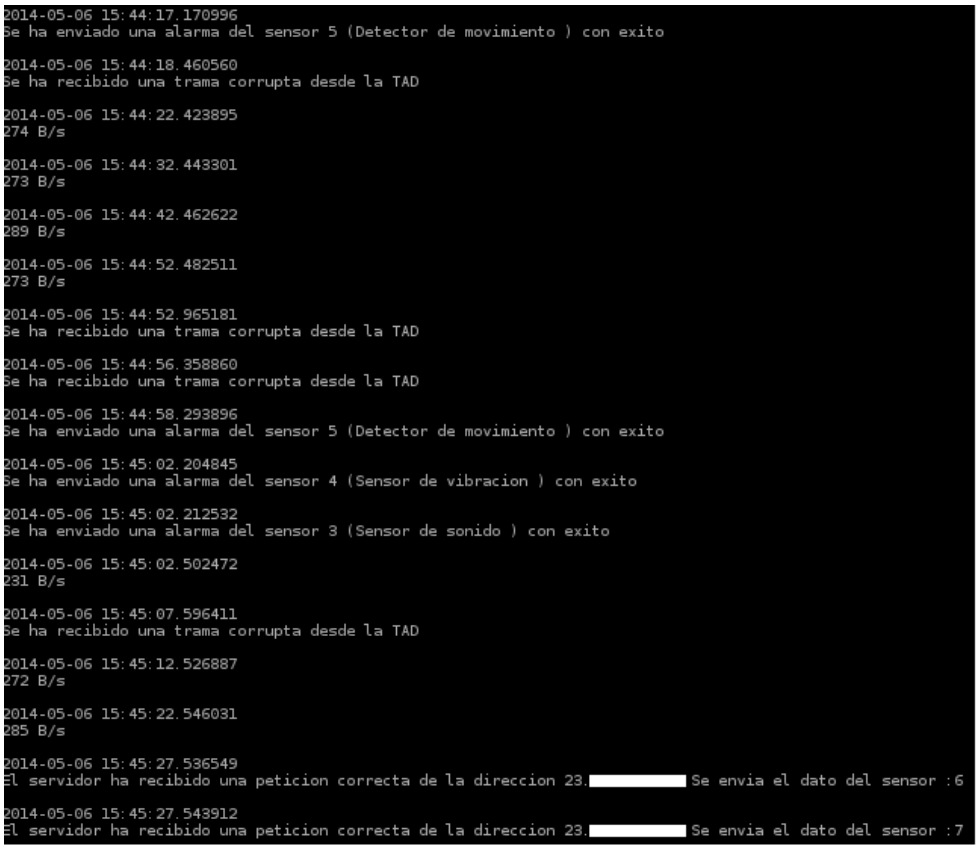

Ilustración 32. Ejemplo del log del ordenador central.

En cuanto a la alimentación eléctrica, en cualquier lugar es posible que ocurran cortes del suministro ya sea por accidentes o provocados. Para contrarrestar estas situaciones, existen sistemas de alimentación ininterrumpida (UPS), estos sistemas constan principalmente de un conjunto de baterías, que se activan instantáneamente y el elemento conectado a ellos, no es consciente de ninguna alteración cuando se corta el suministro. El tiempo de duración del suministro es limitado, y varía en función del dispositivo. En el mercado existen múltiples dispositivos, generalmente cuanto más tiempo aguanten y más potencia otorguen durante el corte, su precio es más elevado. Únicamente hay que conectar el UPS a una toma eléctrica, y el dispositivo que tenga que funcionar en el corte, conectarlo al UPS. Debido a que el coste de estos dispositivos es elevado, y el proceso para implantarlos no es significativo, no se ha adquirido ninguno para este proyecto.

5.3 Implantación y Simulación de estancia

Todo el sistema descrito hasta ahora ha sido implantado en el hogar del autor de este proyecto. La implantación se ha realizado con un nodo completo de tipo estancia, junto con todos los sensores y actuadores vistos. También se ha implantado el nodo de relé portátil, que puede controlar el encendido y apagado de cualquier electrodoméstico situado en el hogar.

Pero el sistema implantado solo puede generar datos de una estancia, y para probar toda la arquitectura del sistema completo, es muy conveniente hacerlo con al menos dos nodos de tipo estancia. Como se indica en el alcance, uno de los objetivos es la simulación de una estancia con datos aleatorios.

El trabajo de realizar esta simulación lo tiene el nodo de la estancia ya implementada. Se ha hecho de esta manera, para que los datos de la estancia simulada tengan que recorrer el máximo camino posible, desde un nodo, hasta la interfaz como lo haría un nodo real. Para realizar la simulación de los sensores que no generan alarmas, se ha tomado el valor del mismo sensor real, y se ha aplicado un factor aleatorio para modificar ese valor. Para los sensores que general alarmas, se ha aplicado una probabilidad de generar una alarma dependiendo del tipo del sensor, por ejemplo el sensor de fuego de la estancia simulada genera una alarma de fuego aproximadamente una vez cada 24 horas. Para enviar la trama de la estancia simulada, se ha utilizado un identificador de trama específico para este nodo simulado.

6 Diseño del servidor

En este capítulo, veremos cómo se comunica el servidor tanto con el ordenador central como con los dispositivos Android y cual es su diseño.

6.1 Comunicaciones entre el ordenador central y el servidor

La comunicación entre el servidor y el ordenador central solo se produce cuando ocurren determinados eventos. Se pueden distinguir dos tipos de eventos diferentes, cuando el ordenador central detecta una alarma y cuando el servidor envía comandos al ordenador. Se ha decidido diferenciar estas dos comunicaciones por dos canales diferentes, para que cuando sea necesario uno u otro esté completamente libre. Además al utilizar la arquitectura cliente-servidor en los dos canales, cuando sea necesario transmitir datos se enviara y recibirá de forma directa, ya que no utilizaremos *comunicaciones de tipo encuesta*, es decir preguntar repetidamente si hay nuevos eventos.

La tecnología que se ha usado han sido *sockets*[30], ya que utilizan el método cliente-servidor. El protocolo de comunicaciones elegido ha sido TCP, ya que posee todas las características necesarias para esta situación frente a UDP. Necesitamos un protocolo que nos garantice que la información llegue al destinatario en el orden enviado, y no necesitamos que sea eficiente en cuanto velocidad de transmisión ya que la información no es transmitida de forma continuada, si no por eventos.

A continuación, en la ilustración 33, se puede ver la arquitectura descrita, la comunicación con la adquisición de datos queda simplificada debido a que ya la hemos visto, y con los clientes Android también, debido a que se verá en un capítulo siguiente.

[30] Socket: http://es.wikipedia.org/wiki/Socket_de_Internet

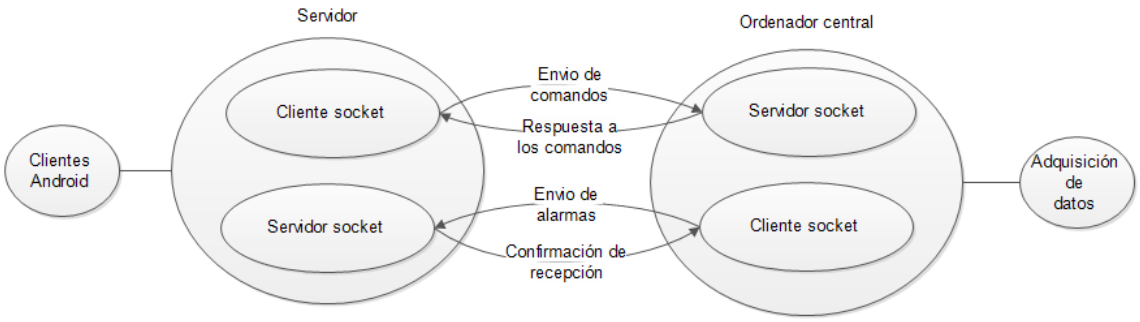

Ilustración 33. Comunicaciones entre el servidor y el ordenador central.

Como ya sabemos, para que un cliente socket se conecte a un servidor necesita una dirección IP. El cliente socket del ordenador central lo tiene fácil, ya que el servidor tiene una dirección estática que nunca cambiará, gracias al servicio de *IP elástica* de Amazon EC2, pero para que el cliente socket del servidor se conecte al ordenador central, también necesitará una dirección que no necesariamente será siempre la misma.

Para adquirir la dirección IP en la que está establecido el servidor del ordenador central, cada vez que se recibe una petición del cliente del ordenador central se guarda su dirección de la siguiente manera.

```
con, addr = server.accept()
```

De esta manera, en "con", tenemos la conexión aceptada en el servidor y en "addr" la dirección ip del cliente. Al inicio del sistema el ordenador central lanza una petición de prueba para compartir su dirección, por si no se generan alarmas para compartirla automáticamente.

En cuando al envió de comandos del servidor, al ordenador central hay cinco tipos diferentes.

- Petición del dato actual del sensor N, siendo N el identificador de un sensor. Se recibe el dato actual de ese sensor.
- Petición de cambio de estado del actuador N. Si el estado ha cambiado correctamente, se devuelve una confirmación.
- Petición para hacer una llamada a un número de teléfono. Incluye el número y un identificador con el tipo de alerta que se transmitirá.
- Petición del estado de la adquisición de datos. Devuelve la velocidad con la que se están recibiendo datos de la adquisición de datos.
- Test de conexión. Devuelve "OK" si se ha recibido la petición.

Para el envió de alarmas solo hay un comando. Cuando se produce una alarma, se envía el identificador del sensor que la ha producido y el dato generado por el sensor.

Para que la información sea interpretada por las dos partes que participan en la comunicación se ha adoptado una solución sencilla. Ya que las dos partes de la comunicación tienen en común el mismo lenguaje de programación se empaquetarán los datos en estructuras de Python que podrán ser fácilmente interpretadas por las dos partes.

```
def getDatoSensor(self, idSensor):
    sensor = self.getSensor(idSensor)
    if sensor.esDigital():
        return struct.pack('!d',sensor.getUltimoDato())
    else:
        return struct.pack('!I',sensor.getUltimoDato())
```

Ilustración 34.Función del ordenador central para empaquetar el dato de un sensor.

```
def actualizarSensor(self, idSensor, Etiqueta):
    peticion = chr(CodPedirDato) + chr(idSensor)
    if Etiqueta == 'Digital':
        estructura = struct.Struct('!d')
    else:
        estructura = struct.Struct('!I')

    cliente = ClienteOrdenadorCentral()
    dato = cliente.peticion(peticion,estructura.size)
    if dato!="":
        dato = estructura.unpack(dato)[0]
        return dato
    else:
        return CodError
```

Ilustración 35. Función del servidor para obtener el dato de un sensor del ordenador central y desempaquetarlo.

Los tipos de datos elegidos, han sido ‘d’, para los sensores que guardan números en coma flotante, que corresponde a un float de ocho bytes, e ‘I’ para todos los demás valores, que son enteros sin signo de cuatro bytes.

En esta comunicación no se ha buscado minimizar el tamaño que ocupa la información, como se ha hecho en la adquisición de datos, ya que también podríamos haber codificado la información por tramas y enviarla de forma empaquetada con una codificación propia. Se ha hecho de esta manera, dado que la información que se puede enviar por segundo entre en servidor y el ordenador central no es crítica, siendo lo fundamental que cuando se envié llegue correctamente y gracias a que se utiliza el protocolo TCP se puede garantizar.

6.2 Arquitectura del servidor

El servidor es el encargado de gestionar toda la lógica y los datos del sistema, así como parte fundamental para comunicar todos los elementos del sistema. En este capítulo se describe su diseño y se detallan los elementos fundamentales de este.

6.2.1 Infraestructura del servidor, Amazon EC2

Para alojar el servidor utilizaremos la plataforma de computación en la nube Amazon EC2. Este servicio permite el control total de una máquina online con permisos de *root*.

Amazon EC2 ofrece un conjunto amplio de diferentes instancias. La instancia elegida es una máquina Ubuntu con Python ya instalado y configurado.

Una vez elegida la instancia, se puede asignar sus características, como el número de procesadores, tamaño del disco duro y tamaño de la memoria principal. Para este proyecto se ha elegido la de características más bajas posibles, para entrar en el plan de un año gratuito. Este plan permite tener usar el servicio con un costo nulo mientras no se excedan unos determinados parámetros que se detallan a continuación.

El plan cuenta con una o dos CPUs, dependiendo de la demanda del momento, 615MB de RAM, 30GB de utilización en el disco duro y dos millones de escrituras/lecturas en el disco duro.

A continuación se muestra la factura obtenida por un uso de la instancia durante 720 horas continuadas durante un mes, es decir, el mes completo con el sistema en funcionamiento.

Elastic Compute Cloud		$0.00
US East (Northern Virginia) Region	Usage	
Amazon CloudWatch		
$0.00 per alarm-month - first 10 alarms	1.000 Alarms	$0.00
Total:		$0.00
Amazon Elastic Compute Cloud running Linux/UNIX		
$0.00 per Linux t1.micro instance-hour (or partial hour) under monthly free tier	720 Hrs	$0.00
Total:		$0.00
EBS		
$0.00 for the first 2 million I/O requests under monthly free tier	180,244 IOs	$0.00
$0.00 per GB-month of provisioned storage under monthly free tier	8.000 GB-Mo	$0.00
Total:		$0.00
Region Total:		$0.00

Ilustración 35. Costes de instancia Micro Amazon Ec2

Como se puede ver, hemos utilizado 180.000 IOs de dos millones, lo que representa el número de operaciones en el disco duro. Y ocho gigas de treinta, esto no significa que tengamos ocupados ocho gigas en el disco duro, si no que se han producido movimientos que en total suponen ocho gigas, por ejemplo al iniciar la instancia, el sistema operativo se carga en la memoria principal, lo que representa al menos una lectura del disco de 250 mb.

6.2.2 Diseño

A continuación se detallan clases que forman parte del servidor y como interaccionan entre sí.

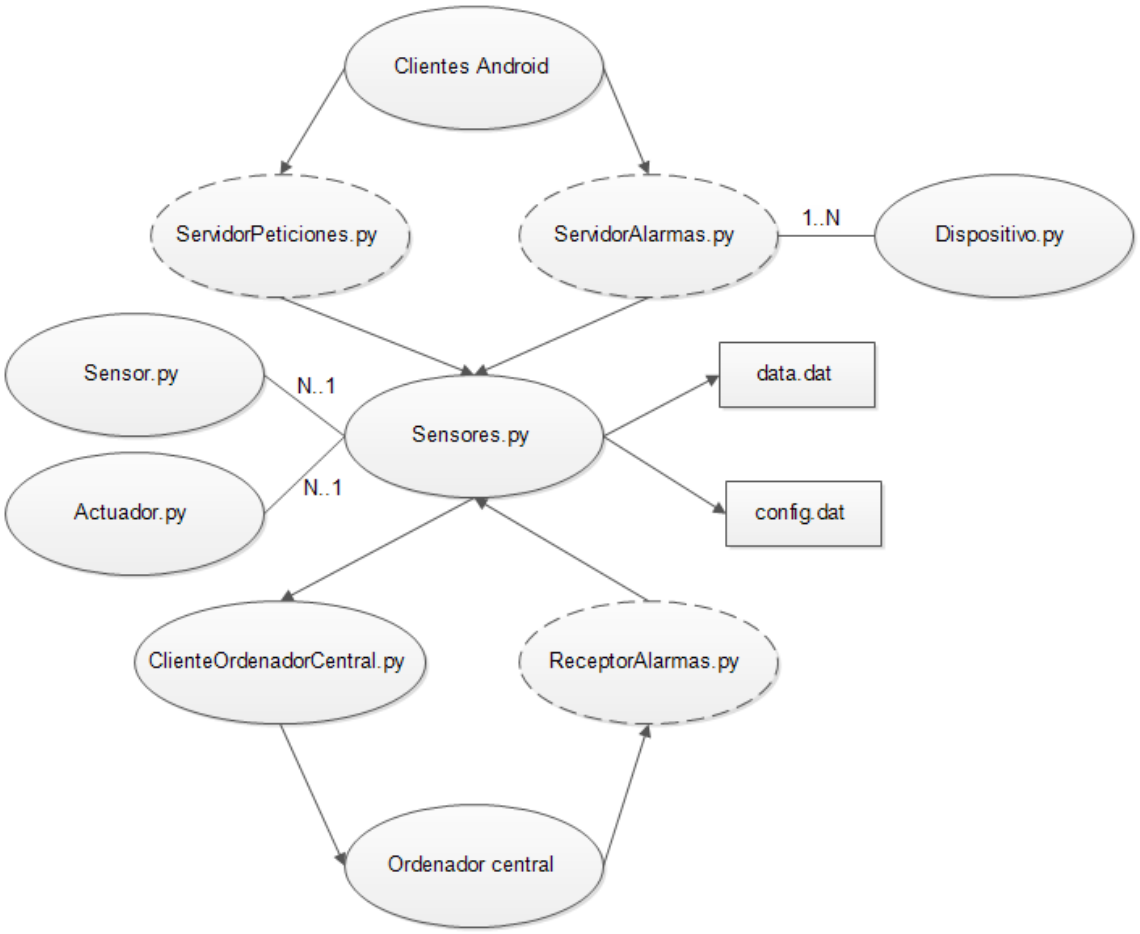

Ilustración 36. Interacción de las clases en el servidor.

- **ClienteOrdenadorCentral.py:** Envía las peticiones al ordenador central, las más comunes son las peticiones de datos de sensores. Esta clase solo se instancia cuando es necesario su uso.
- **ReceptorAlarmas.py:** Es un *servidor socket* que está continuamente a la escucha a la espera de alarmas. La instancia se crea una única vez al inicio del sistema.
- **ServidorPeticiones.py:** Es un servidor socket que está continuamente a la escucha de peticiones por parte de los clientes Android. La instancia se crea una única vez al inicio del sistema.
- **ServidorAlarmas.py:** Es un *servidor socket* que se encarga de gestionar el envió de alarmas a todos los clientes Android. La instancia se crea una única vez al inicio del sistema. Su funcionamiento detallado se describe en el apartado “6.3 Comunicaciones entre el servidor y los clientes”.
- **Dispositivo.py:** Representa a un cliente Android, se crean tantas instancias como dispositivos estén conectados.

- **Sensores.py:** Es la clase que gestiona toda la información y contiene la lógica de cómo se comporta el sistema en cada situación. Se crea una única instancia al comienzo del sistema que utilizan las clases que acceden a ella.
- **Sensor.py:** Clase encargada de guardar la información de un sensor, se instancias tantas como sensores existan.
- **Actuador.py:** Clase encargada de guardar la información de un actuador, se instancias tantas como actuadores existan.
- **Constantes.py:** Fichero auxiliar. Constantes que afectan a más de una clase.
- **Timer.py**: Clase auxiliar que se utiliza para ejecutar un determinado método estático cada tiempo indicado, por ejemplo para actualizar los datos de los sensores, cada hora llama a un método estático de la calase sensores.
- **Ficheros de datos**: quedan detallados en el capítulo siguiente.

6.2.3 Almacenamiento de datos

En primer lugar distinguiremos los tipos diferentes de datos que se necesitan almacenar.

- Datos de sensores por hora: En el minuto cero de cada hora, se piden datos actuales de todos los sensores al ordenador central y se almacenan en el objeto sensor. Estos datos solo son útiles para el usuario, solo se utilizan para hacer estadísticas de las últimas 24 horas.
- Datos de alarmas de sensores: Estos datos muestran en qué fecha un sensor ha producido una alarma, son útiles para el usuario y para determinar el nivel de alerta de una alarma, ya que si se produce una alerta de movimiento justo después de una de sonido se categoriza con una probabilidad más alta. También se guardan en el objeto sensor, con un máximo de 120 alarmas por sensor, es un parámetro ampliable.
- Datos de configuración: Todos los datos de configuración por parte del usuario, como los teléfonos a los que llamar o el comportamiento del sistema en casos concretos.
- Datos de estado: Muestran el estado del sistema, como si está activada o desactivada la alarma o si un relé está encendido o apagado.

El servidor está diseñado para estar en todo momento en funcionamiento, es decir, desde que se instale el sistema en un establecimiento, hasta que termine su uso. Esto quiere decir, que si no ocurriera ningún incidente, bastaría con almacenar los datos en la memoria principal. Pero como el proveedor de computación en la nube no puede ofrecer una disponibilidad del 100% y el

servidor puede que por alguna incidencia se reinicie, necesitamos idear un método que guarde esta información.

El método más común para guardar la información en estos casos suele ser una base de datos, pero el uso de una base de datos, requiere la implantación de un gestor, utilizar un SGBD e instalar en el servidor los medios necesarios para su funcionamiento, como un conector. Como en este proyecto no se ha incluido dentro del alcance realizar estas tareas, se ha buscado una solución más sencilla.

El lenguaje Python, posee métodos para empaquetar y desempaquetar objetos, la librería que lo permite, se llama "pickle"[31]. Lo que hace es, dado un objeto, que puede constar de muchos sub-objetos lo empaqueta y permite escribirlo en un fichero. Esta parece una buena solución para los *datos de estado* y *los datos de configuración*, ya que no varían con mucha frecuencia y son muy pocos. Pero para los datos de los sensores, tendríamos que estar guardando todos los datos para cada nueva alarma o dato, esto no se puede permitir debido a la tasa tan elevada de escrituras/lecturas que requeriría en el disco duro.

Para abordar este problema se ha utilizado un proyecto de Python que permite utilizar la librería pickle haciendo lecturas y escrituras incrementales. Es decir, si por ejemplo tenemos en un fichero guardado la estructura sensores, que son los datos de todos los sensores, y necesitamos solo actualizar un sensor o leer el dato de un sensor, podremos hacerlo con esta librería. El proyecto mencionado se llama "Streaming-pickle[32]".

Para guardar los datos con este método se han utilizado dos ficheros, config.dat, donde se guaran todos los datos de configuración y data.dat, donde se guardaran los datos de sensores y de estado.

```
def guardarDatosSensores(self):
    s_dump(self.sensores, open('data.dat', 'w'))
```

Ilustración 37. Función que almacena los datos de los sensores modificados.

[31] Librería pickle Python: https://docs.python.org/2/library/pickle.html
[32] Librería streaming-pickle Python: https://code.google.com/p/streaming-pickle/

```
def cargarDatosSensores(self):

    try:

        i = 0
        for sensor in s_load(open('data.dat')):

            self.sensores[i] = sensor
            i = i + 1

    except IOError:

        msg('No existe el fichero de datos, se creara uno.')
```

Ilustración 38. Función que lee los datos de los sensores desde el fichero.

6.2.4 Robustez del servidor

Al igual que en ordenador central, en el servidor también existen medios para asegurar el continuo funcionamiento.

El entorno del servidor es el mismo que el del ordenador central, es decir Python bajo Linux, por esa razón se ha utilizado el mismo script que en el ordenador central, para asegurar el inicio del sistema y el reinicio en excepciones no controladas.

El sistema de excepciones también es el mismo. Si se produce una de carácter bajo, como el fallo de una comunicación se intenta tres veces más. Si se produce una excepción no controlada se terminan todos los procesos de Python y el script *daemon* los ejecuta de nuevo.

Además de eso, como el servidor tiene que tener activos tres servidores socket, se han tomado metidas exclusivas para estos. Ya que se podría dar la circunstancia de que un servidor se quedara bloqueado y no generara ninguna excepción. Este caso concreto se ha dado cuatro veces durante 720 horas de uso continuado de todo el sistema, y está relacionado con la implementación de la clase socket de Python. Para prevenir este problema, se he implementado una clase en el mismo servidor, que cada cierto tiempo hace una petición a todos los servidores desde local. En caso de respuesta negativa se procede a su reinicio.

Por último, al igual que en el ordenador central, también existe un log, en el que se puede ver todos los eventos producidos, así como los errores.

6.3 Comunicaciones entre el servidor y los clientes

La comunicación en este caso requiere peticiones por parte de los dispositivos al servidor y el informe de alarmas producidas del servidor a los clientes. Por esa razón se ha decidido utilizar la misma arquitectura que para el caso anterior de los pares servidor - ordenador central.

También es un requerimiento del proyecto responder a peticiones, e informar de las alarmas producidas a todos los dispositivos que lo requieran simultáneamente.

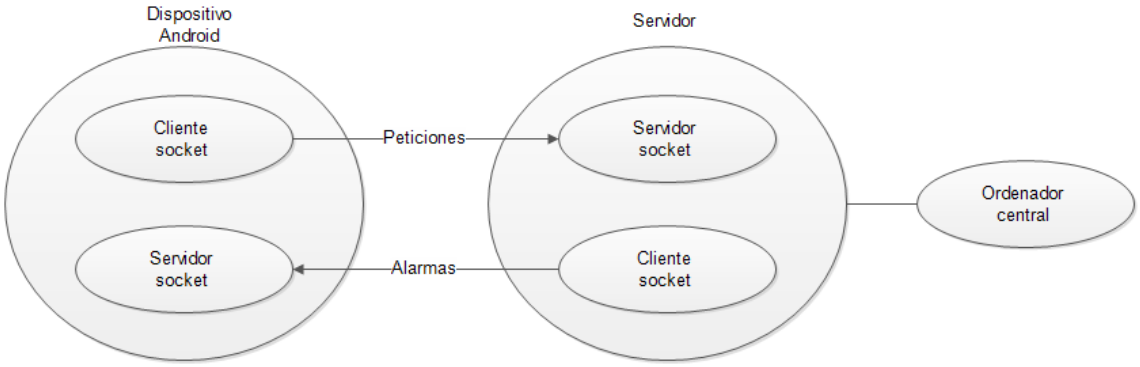

Ilustración 39. Primer diseño de las comunicaciones entre el servidor y los dispositivos Android.

En la ilustración 40, se puede ver el primer diseño de comunicaciones realizado. Este diseño es útil siempre que los dispositivos Android tengan los puertos de entrada abiertos, para permitirles ejecutar un servidor, pero esta situación solo se da cuando el dispositivo está conectado a una red Wi-Fi, ya que la mayoría de las redes de datos no lo permiten.

Para solucionar este problema se ha invertido el canal de comunicación para las alarmas, situando el cliente en el dispositivo y el servidor socket en el servidor, a pesar de que ahora tener dos clientes en el dispositivo Android pueda parecer redundante, se he continuado con esta arquitectura ya que es importante distinguir entre un canal para recibir alarmas, que no pueda ser bloqueado por otras comunicaciones.

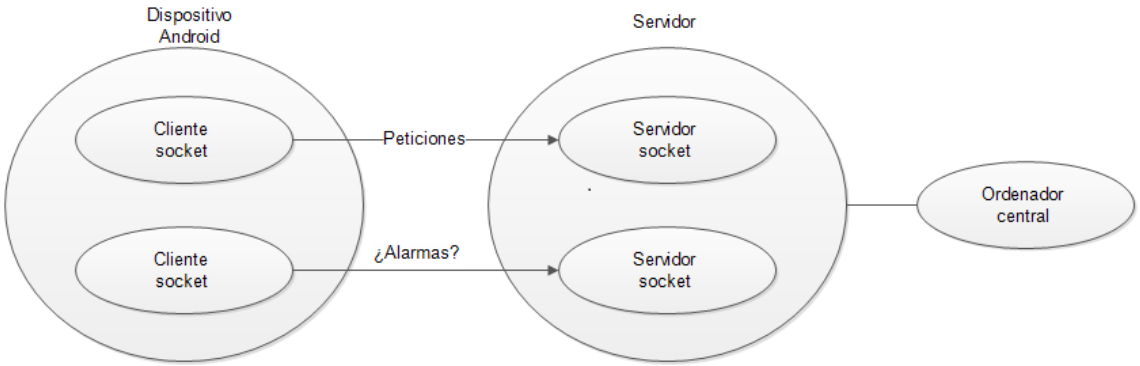

Ilustración 40. Diseño de las comunicaciones entre el servidor y los dispositivos Android.

Para atender a múltiples peticiones con el servidor de peticiones, se han paralelizado las peticiones que requieren un tiempo de espera largo. Por ejemplo, las peticiones que requieren acceder a datos del ordenador central, tienen un tiempo de espera largo porque requiere que el ordenador central responda, un ejemplo de esta petición es el acceso a un dato del sensor en tiempo real. En cambio si se requiere acceder o modificar un paramento que se encuentra en el servidor, la petición no se paraleliza.

En cambio, para enviar las alarmas, cuando se produzca una hay que enviar la misma alarma a tantos dispositivos que estén conectados desde el servidor. Para ello se utiliza el ID único de 16 bytes que tienen todos los dispositivos Android. El servidor recibe una petición y si a ese dispositivo aún no se le he enviado la alarma actual, se envía.

Las peticiones de cada dispositivo para comprobar si hay alarmas se producen cada 1.5 segundos por parte de los clientes. Es verdad que podríamos dejar todos los clientes a la espera y atenderlos cuando se produzca una alarma, pero se ha preferido hacerlo por *encuesta,* preguntando continuamente si hay alarmas para comprobar de esta manera la correcta conexión al servidor. Las alarmas son enviadas a clientes nuevos o desconectados con un tiempo de hasta una hora después de que se produzca la alarma.

A continuación se muestra el código del servidor encargado de responder a las peticiones de alarmas tal y como está implementado.

```
while 1:

    con, addr = server.accept()

    deviceId = con.recv(16)


    if(self.__esNuevo(deviceId)):

        self.__addNew(deviceId)
        con.send(chr(0))

    else:

        if self.__hayAvisosPendientes(deviceId):

            avisos = self.__getAvisosPendientes(deviceId)

            con.send(avisos)


        else:

            con.send(chr(0))

        con.close()
```

Ilustración 41. Código del bucle principal de ejecución del servidor encargado de comunicar las alarmas.

En cuento a la codificación para enviar los datos, esta vez no tenemos el mismo lenguaje en las dos partes, sino que, será entre Python y Java. Por esa razón se ha decidido enviar los datos en formato ascii, ya que es el método que menos tiempo de implementación requiere. Por ejemplo para enviar un dato de temperatura, se enviara el string “24.03”.

El servidor de alarmas, solo acepta un comando, que contesta si se ha producido una alarma y guarda la identificación del cliente que la solicita, pero el servidor que atiende peticiones, tiene 24 comandos diferentes.

- C0: Se envía la ID de un sensor, y se recibe una lista con todos los datos de ese sensor por hora, de las últimas 24 horas.
- C1: Se envía la ID de un sensor y se recibe el dato actual de ese sensor.
- C2: Se envía la ID de un relé más un estado y cambia ese relé al estado indicado.
- C3: Apaga o enciende la alarma general del sistema.
- C4: Obtiene el estado de la alarma general.
- C5: Envía el código de un actuador, más una hora de inicio y de fin y establece en ese actuador un evento diario.

- C6: Envía el ID de un actuador, más una fecha de inicio y de fin y establece en ese actuador un evento puntual.
- C7: Obtiene el estado de un actuador, encendido o apagado.
- C8: Envía el ID de un actuador, y anula su evento puntual.
- C9: Envía el ID de un actuador, y anula su evento diario.
- C10: Envía el ID de un actuador y recibe la hora de inicio y de fin de su evento periódico.
- C11: Envía el ID de un actuador y recibe la fecha de inicio y de fin de su evento puntual.
- C12: Envía un número de teléfono más un índice y lo establece como teléfono para hacer llamadas.
- C13: Envía un índice y recibe el número de teléfono asociado.
- C14: Envía un índice y borra el número de teléfono asociado.
- C15: Envía un índice y se hace una llamada de test al teléfono asociado.
- C16: Devuelve el log completo de todas las alarmas generadas, incluyendo la hora de cada alarma.
- C17: Establece el estado de la simulación de presencia activado/desactivado.
- C18: Obtiene el estado de la simulación de presencia.
- C19: Activa el botón de pánico.
- C20: Activa/desactiva el sistema preventivo.
- C21: Obtiene el estado del sistema preventivo.
- C22: Obtiene la tasa en bps de la adquisición de datos.
- C23: Comprueba la conexión con el ordenador central.
- C24: Comprueba la conexión con el servidor.

Debido a las múltiples peticiones que existen, en la carga de determinadas pantallas de la aplicación se puede dar la circunstancia que se requieran cinco o más peticiones al mismo tiempo.

Si las peticiones se realizan de manera secuencial, es decir, si no se empieza una petición hasta que se recibe la anterior, para calcular el tiempo de realización de todas las peticiones, hay que multiplicar la latencia del servidor por el número de conexiones, teniendo en cuenta que el servidor está alojado en la costa Este de EEUU, la latencia es alta, en torno a un segundo para hacer una petición y que llegue de vuelta.

Para evitar estos problemas de latencia, cuando se requiere una acción con múltiples peticiones se abren N peticiones simultáneas en diferentes hilos de ejecución. Con esto conseguimos que la latencia para N peticiones sea el

equivalente al tiempo necesario para realizar una, ya que el tiempo de procesamiento del servidor es despreciable comparado con la latencia.

Para esperar a que los N clientes tengan sus respuestas, se utiliza un contador atómico, con la clase AtomicInteger de java, para evitar problemas en el caso de que dos clientes escriban en el en el mismo instante y provoquen un problema. Una vez que la interfaz vea que el contador asciende al número de peticiones esperado, sabrá que tiene todas las respuestas y podrá continuar.

7 Diseño de la aplicación Android

Para cumplir con los requerimientos de la aplicación, se han utilizado diferentes herramientas de Android. A continuación se detalla su funcionamiento y como interaccionan entre si.

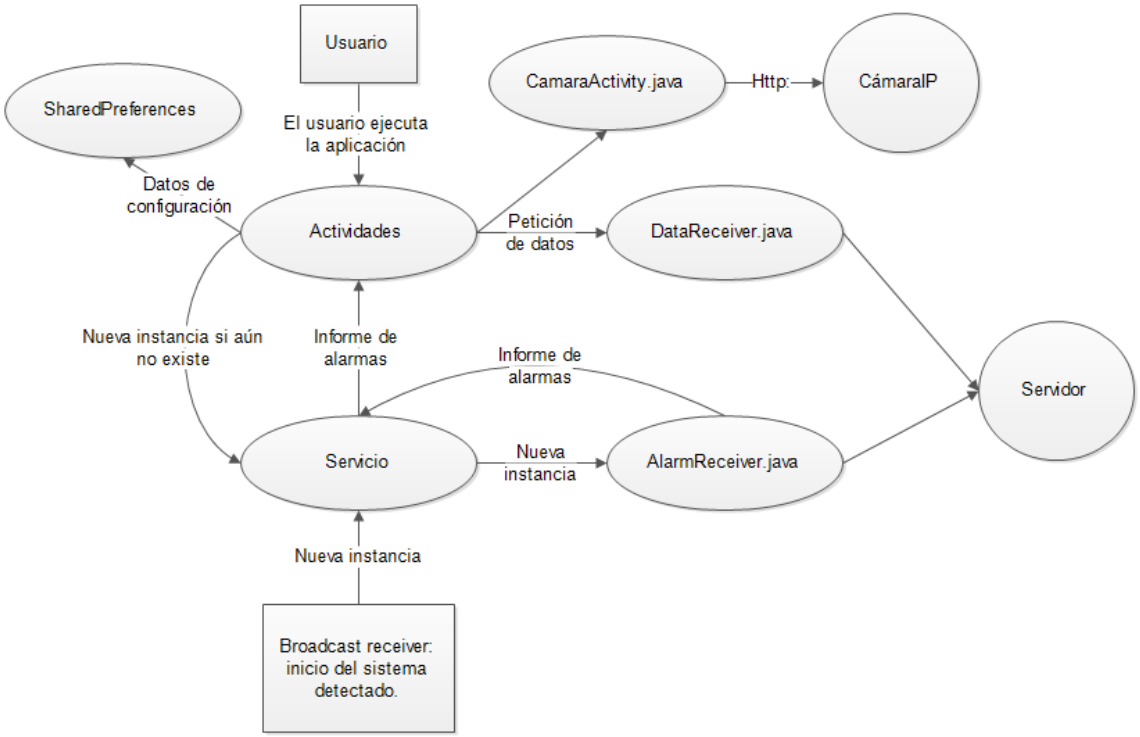

Ilustración 42. Diseño del software de los dispositivos Android.

En los siguientes apartados, se describe cada parte del sistema ilustrada en este esquema.

7.1 Actividades

En Android las ventanas o *layout* se describen en formato xml, y toda la lógica de la ventana se describe en una clase. Por buenos hábitos, si en *main.xml* se describe la interfaz, llamaremos *MainActivity.java* a la clase encargada de la lógica.

Las actividades desde que se crean hasta que se destruyen tienen un ciclo de vida, como se puede ver en la ilustración 44. Por cada estado por el que pasa la

actividad, genera una llamada a una función o genera un evento, la única función que tienen que usar obligatoriamente todas las actividades es la inicial, "onCreate()", que es en la que se debe inicializar todo el contenido. Pero para todas las actividades que muestran datos actualizándose, utilizaremos los eventos "onResume()" y "onPause()" para que cuando el usuario no este visualizando la actividad, por ejemplo al apagar la pantalla o al ponerla en segundo plano, no estén haciéndose peticiones de datos actualizados al servidor.

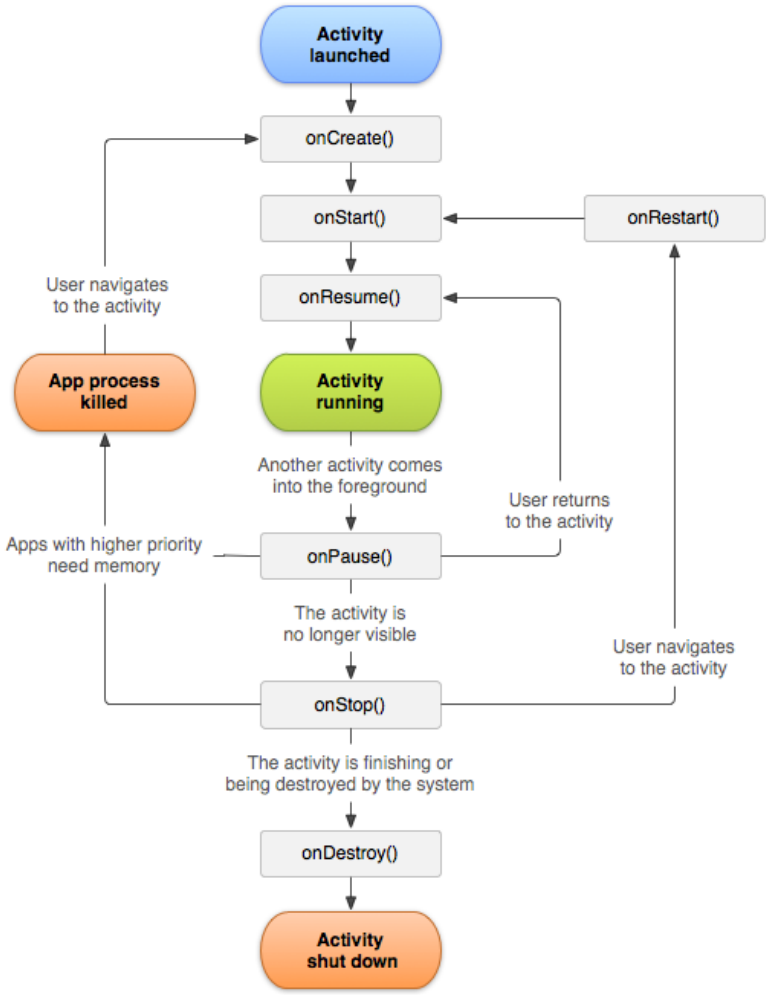

Ilustración 43. Ciclo de vida de la actividad en Android.
Fuente: http://developer.android.com/

7.1.1 Actividad de cámara IP

A diferencia de las actividades del apartado anterior, es necesario destacar esta ya que establece comunicación con una cámara IP por medio del envió de comandos http.

La cámara IP adquirida no permite el envió de video en streaming en un formato válido para Android, por lo que se ha optado por obtener capturas de pantalla que genera la cámara, y estar descargándolas continuamente mientras este la actividad de la cámara activa.

Para acceder a las capturas de pantalla se ha utilizado un enlace directo que posee la cámara de la siguiente manera:

http://[DireccionIp]/snapshot.cgi?user=[Usuario]&pwd=[Contraseña]

Para descargar la imagen se ha utilizado la siguiente función de la clase BitMap de java:

Bitmap image = getBitmapFromURL(URL);

Para visualizarla se ha utilizado el objeto ImageView de Android.

En cuanto al envió de comandos a la cámara, se han utilizado peticiones http al servidor que tiene empotrado la cámara IP, utilizando la siguiente cadena:

http://[DireccionIp]/decoder_control.cgi?user=[Usuario]&pwd=[Contraseña]&command=[Comando]

Aunque la lista de comandos que dispone la cámara es muy larga, solo utilizaremos comandos para mover la cámara.

A continuación se muestra como ejemplo, la función implementada para detener el movimiento de la cámara enviando el comando número uno.

```
private void parar()
{
    try
    {
        HttpClient httpclient = new DefaultHttpClient();
        HttpResponse response = httpclient.execute(new HttpGet(URL_COMMAND + "1"));
        response.getEntity().getContent().close();
    }
    catch(Exception e)
    {
        Log.e("Error","No se pudo operar la camera");

    }

}
```

Ilustración 44. Función encargada de parar el movimiento de los servos de la cámara.

7.2 Shared preferences

Es una clase de Android que simplifica el almacenamiento de la información de configuración de la aplicación. Se utiliza para guardar los parámetros de la aplicación que no afectan al sistema, sino a una sola aplicación, por ejemplo el tipo de tono para una alarma concreta o la frecuencia con la que se actualizan los datos en la interfaz.

En las siguientes instrucciones estamos leyendo del archivo file, la preferencia con identificador “F_A” y con valor por defecto 10, el valor por defecto se utiliza cuando el campo no tiene ningún valor asignado.

```
SharedPreferences settings = getSharedPreferences(file, 0);
int fa = settings.getInt("F_A", 10);
```

Escritura:

```
SharedPreferences settings = getSharedPreferences(file, 0);
SharedPreferences.Editor editor = settings.edit();
editor.putInt("F_A",value);
editor.commit();
```

7.3 DataReceiver.java

Clase encargada de procesar todas las peticiones al servidor, es un cliente socket, contiene métodos para cada comando del servidor, se crea una instancia para cada petición.

Esta clase es usada por múltiples actividades, tanto para recibir datos como enviar comandos. Por ejemplo, para recibir una lista con todos los datos de un sensor se haría de la siguiente manera.

```
DataReceiver dataReceiver = new DataReceiver(context);
List<Data> listaDatos = dataReceiver.getListData(idSensor);
```

O para establecer el estado de un relé:

```
DataReceiver dataReceiver = new DataReceiver(context);
dataReceiver.setRelay(codTad0, (byte) 0);
```

7.4 Servicio

Los servicios en Android no se pueden iniciar solos, si no que tienen que estar enlazados a una actividad por medio del fichero *manifest*. Gracias a este enlace, tanto el servicio como la clase pueden instanciarse mutuamente. Por ejemplo, si el usuario inicia la aplicación y el servicio no está instanciado la aplicación creará una instancia de éste, y si se inicia el servicio y requiere mostrar la aplicación para informar de una alerta el servicio podrá instanciar la aplicación.

Cuando un usuario cierra una aplicación, la actividad termina, pero su servicio asociado sigue en funcionamiento. El único método de parar un servicio es desde la actividad que se creó o con el apagado del dispositivo.

A la hora de diseñar un servicio, hay que tener en cuenta, que en Android los servicios utilizan el hilo de la actividad principal, entonces, si en el servicio tenemos una tarea bloqueante, la interfaz gráfica de la aplicación no responderá. Por esa razón conviene crear los servicios en hilos de ejecución diferentes mediante la clase *Thread* de java.

En Android existen dos métodos diferentes de iniciar un servicio, uno método es realizando una llamada a “bindService()”, este tipo de servicios están orientados a tareas que se tengan que realizar en segundo plano, pero en un periodo concreto de tiempo, también para cuando es necesario una comunicación continuada entre el servicio y la actividad, en ambas direcciones. Y por otro lado el método “startServeice()”, está pensado para servicios en los que se espera que realicen una tarea en segundo plano de forma continuada, este último es el tipo de servicio utilizado. Para los dos tipos de servicios diferentes, existe un ciclo de vida propio como se puede ver en la ilustración 46.

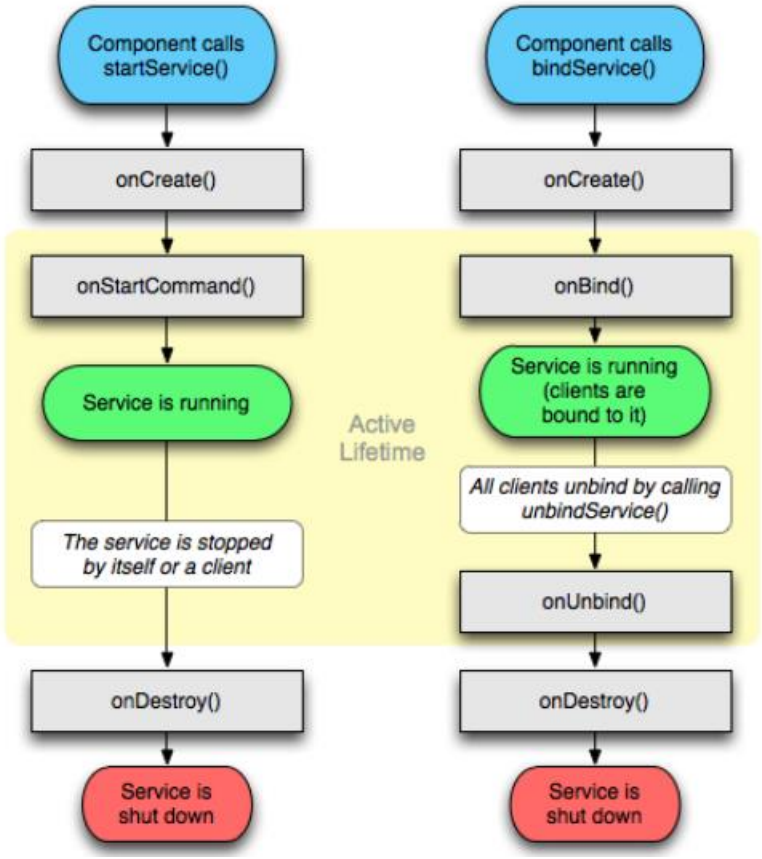

Ilustración 45. Ciclo de vida de los servicios en Android.
Fuente: http://developer.android.com/

En este proyecto, el servicio se encarga que en todo momento una parte de la aplicación este activa a la escucha de alarmas, para garantizar que la recepción siga funcionando mientras la aplicación no esté activa. La primera vez que se ejecuta la aplicación, el servicio es iniciado por la actividad principal y desde entonces, en el inicio del sistema por un *broadcast receiver*.

Además, hay que tener en cuenta que algunos dispositivos paran el procesador cuando el dispositivo no se usa durante un tiempo, esto supone que el servicio se quede pausado y no podría realizar ninguna acción, ignorando las posibles alarmas.

Para que el servicio no se pause nunca, se ha utilizado el mecanismo *WakeLock*[33] de Android, de esta manera hasta que no lo indiquemos no se pausara el procesador.

[33] WakeLock: http://developer.android.com/reference/android/os/PowerManager.WakeLock.html

7.5 AlarmReceiver

AlarmReceiver.java, es la clase que utiliza el servicio para recibir las alarmas. Es un cliente socket, que está pidiendo el estado de las alarmas de forma continuada con una determinada espera, también se encarga de decodificar las alarmas y generar la alerta adecuada dependiendo del tipo de alarma. Abre la actividad principal de forma automática si la alarma lo requiere.

También cabe mencionar, que en cada petición que hace al servidor, envía un identificador único del dispositivo Android, para que el servidor sepa a qué dispositivos ya ha enviado una alarma concreta. El identificador se obtiene de la siguiente manera.

```
byte[] android_id = Secure.getString(context.getContentResolver(), Secure.ANDROID_ID).getBytes();
```

7.6 BroadcastReceiver

Es un mecanismo de los dispositivos Android para ejecutar una determinada acción ante eventos determinados. Se utiliza para ejecutar el servicio en el inicio del sistema.

Para activar un BroadcastReceiver primero hay que registrarlo, se puede hacer desde el archivo manifest de la siguiente manera.

```
<receiver android:name="paquetePrincipal.main.Starter">
    <intent-filter>

        <action    android:name="android.intent.action.BOOT_COMPLETED" />

    </intent-filter>

</receiver>
```

En la acción estamos indicando ante qué tipo de evento queremos que reaccione y en el nombre indicamos qué clase se llamará. La clase llamada tiene que extender la clase BroadcastReceiver.

```
public class Starter extends BroadcastReceiver{


    @Override
    public void onReceive(Context context, Intent intent) {

        Intent serviceIntent = new Intent(context,AlarmService.class);
        if(!AlarmService.isRunning())context.startService(serviceIntent);

    }
}
```

En la clase que es llamada con el evento, podemos indicar que tipo de acción iniciar, en nuestro caso, iniciar el servicio.

7.7 Permisos de la aplicación Android

Cuanto una aplicación de Android usa recursos o características del sistema que pueden afectar al comportamiento del dispositivo, la aplicación tiene que indicar en el fichero *AndroidManifest.xml* que el sistema usará estos recursos. De esta manera cuando el usuario instale la aplicación en su dispositivo, podrá ver que recursos utilizará la aplicación. Los permisos utilizados en la aplicación quedan detallados a continuación.

- **INTERNET**. Este permiso es usado para que la aplicación se pueda comunicar con el servidor.
- **ACCESS_NETWORK_STATE**. Este permiso se utiliza para comprobar si el dispositivo tiene conexión a internet. En caso negativo se advierte al usuario, ya que la aplicación no tiene ninguna funcionalidad sin conexión. De esta manera se puede diferenciar cuando una conexión falla por no tener conexión a Internet o al servidor.
- **VIBRATE**. El control de vibración se utiliza después de que el usuario pulsa el botón de pánico, para notificar que la alerta ha sido enviada correctamente. Se ha optado por no escribir un mensaje de confirmación, ya que ese tipo de mensaje, visto por la persona no indicada podría crear una situación indeseable.
- **RECEIVE_BOOT_COMPLETED.** Se utiliza para detectar el encendido del dispositivo.
- **WAKE_LOCK.** Utilizado para evitar que el dispositivo entre en modo de suspensión.

8. Lógica del sistema

En este capítulo se detalla cómo se comporta el sistema cuando se produce una alarma. En primer lugar veremos que sucede cuando se origina una alarma y después las opciones de configuración.

8.1 Alarmas

Cuando un sensor activa una alarma, esta adquiere una probabilidad base dependiendo del tipo que sea. A continuación se toman datos de sensores actuales y también se comprueban las alarmas anteriores existentes, después, se guarda la alarma en el servidor, y se envía una alarma a todos los dispositivos Android, indicando el tipo de alarma y la probabilidad. Y en paralelo se inicia el proceso necesario para informar por llamadas telefónicas de la alarma. Las posibles probabilidades de una alarma son cinco; muy baja, baja, media, alta y muy alta.

Cuando el sensor de humo se activa, la alarma se categoriza de base como probabilidad alta, a continuación se accede al dato de temperatura de la última hora y se compara con el dato de la temperatura actual si la temperatura es superior en más de un grado, se sube la probabilidad de la alarma.

Cuando se genera una alarma por sonido o por vibración, de base se categoriza como muy baja, ya que estas alarmas se pueden producir por eventos diferentes de los que queremos detectar, como vibración por una ráfaga de viento en la ventana o sonido por un ruido intenso proveniente del exterior. Este tipo de alarmas solo serán mostradas en el cliente Android si así está configurado por el usuario en su terminal, y además la alarma del sistema está activada. Estas alarmas nunca generarán llamadas telefónicas.

Cuando el sensor de presencia se activa, se le asigna una probabilidad base media, a continuación se comprueba si en los últimos minutos se han producido alarmas de probabilidad muy baja por vibración o sonido, por cada una se aumenta la probabilidad, estas alarmas por norma serán mostradas en los clientes Android y generarán llamadas siempre y cuando la alarma general del sistema este activada.

Cuando el usuario activa la alarma de pánico desde su dispositivo Android, esta alarma es comunicada a todos los demás dispositivos, excepto el que la ha mandado, y también genera llamadas telefónicas.

8.2 Configuración y opciones

- **Simular presencia**. Permite desactivar o activar este estado. Si se encuentra activado, la iluminación de una estancia se puede encender con una probabilidad baja y seguidamente apagarse. Cada minuto se genera un número aleatorio para encender la iluminación, o apagarla en caso de que este encendida. Para que la simulación sea lo más real posible, cuando se apaga o enciende la luz no se hace en el minuto que se genera el evento, si no pasados un número de segundos entre uno y veinte, número también aleatorio.
- **Evento diario**. Permite activar un evento que se repetiría diariamente para la iluminación o relé seleccionado. Permite indicar una hora de inicio y otra de fin.
- **Evento puntual**. Permite establecer un evento único para la iluminación o un relé. Permite indicar una fecha de inicio y de fin.
- **Comportamiento preventivo.** Activa o desactiva este modo, cuando está activado. Si se detecta presencia por sonido o por vibración, sea activará la iluminación de dicha instancia con la finalidad de disuadir a los posibles intrusos que estén intentando entrar.
- **Frecuencia de actualización**. Establece cada cuantos segundos se actualizan los datos en tiempo real del dispositivo Android, el valor por defecto es 10. La modificación de este valor no afectará al tiempo de reacción de la recepción de las alarmas.
- **Forzar sonido**. Si esta opción está activada, cuando se produce una alarma emitirá el sonido que le corresponda aunque el volumen del dispositivo este desactivado.
- **Alarmas de probabilidad baja**. Permite establecer el tipo de sonido en alarmas de probabilidad baja, no hacer nada, sonido simple o sonido continuado. El sonido continuado seguirá emitiéndose hasta que el usuario lea el mensaje de alarma.

9. Interfaz de usuario y pruebas

En este capítulo, veremos los distintos tipos de interfaz creados para el proyecto, cuáles han sido las decisiones para su diseño y los resultados obtenidos. Para finalizar veremos cómo se ha probado el sistema completo.

Más del 90% de los dispositivos Android que están actualmente en el mercado, están entre las versiones 2.2 y 4.3, que es el rango de dispositivos compatibles para la aplicación. Para cada versión de Android, los dispositivos solo admiten un conjunto de temas, cuanto más alta sea la versión de Android, el dispositivo admitirá más temas, los propios de su versión más todos los anteriores.

Para no tener que usar el mismo tema en todos los dispositivos, se han elegido dos temas diferentes, uno para dispositivos antiguos y otro para dispositivos más actuales. Dependiendo del dispositivo que ejecute la aplicación, el tema cambiará dinámicamente. Para ello, se han utilizado "configuration qualifiers". Es un mecanismo Android, que permite crear diferentes directorios y a cada directorio asignar un rango de versiones, los recursos que estén en estos directorios, como la definición de temas, se activarán en función de la versión Android.

El tema elegido para dispositivos anteriores a la versión 4.0 es "Theme.Black.NoTitleBar.Fullscreen" y para dispositivos superiores "Theme.Holo.NoActionBar.Fullscreen". Para todos los ejemplos que veremos a continuación el tema mostrado será Holo y el dispositivo usado es LG-P710 con la versión 4.1 y pantalla de 4.3". La aplicación también ha sido probada con dos dispositivos más de 3,3"

9.1 Interfaz de cada estancia

La finalidad de esta pantalla es presentar todos los datos de cada estancia en tiempo real, sus estadísticas, así como el control de los relés y el acceso a la cámara. Existe una para cada nodo que tenga instalado el sistema.

En esta pantalla, existen una gran cantidad de datos a mostrar, por eso se ha utilizado el elemento gridView. Es un elemento de tipo tabla, que permite en cada fila o columna incluir los elementos deseados, además permite modificar el número de filas o columnas de forma dinámica, se ha utilizado está propiedad, para establecer el número de columnas a dos, cuando cambia la orientación del teléfono.

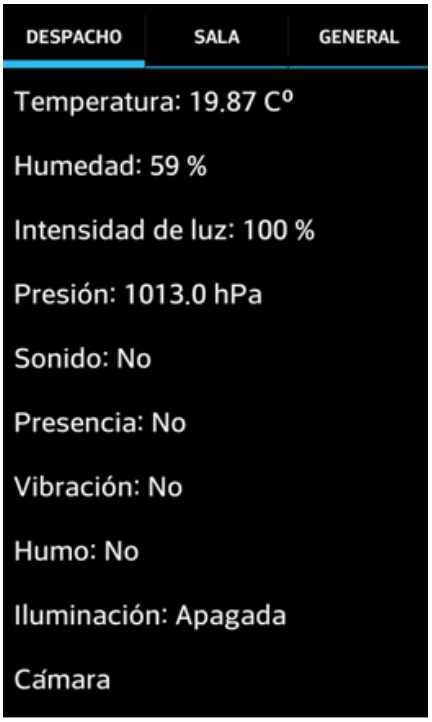

Ilustración 47. Pantalla principal de la aplicación vertical.

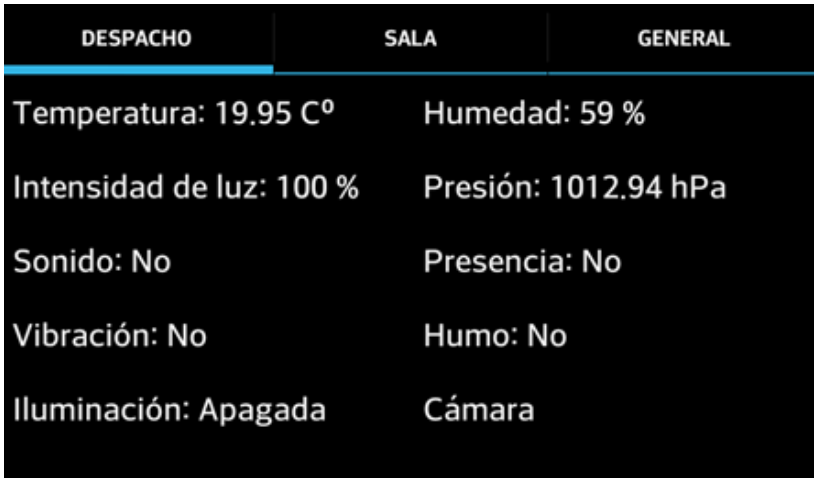

Ilustración 46. Pantalla principal de la aplicación horizontal.

9.2 Gráficas

Las gráficas muestran para cada sensor, los datos del sensor de las últimas 24 horas, para los sensores que generan alarmas, se muestran el número de alarmas generadas por hora. Para mostrar las gráficas, se ha utilizado la librería

aChartEngine. Una vez que se muestra la gráfica, permite ajustarla por medio de la pantalla táctil.

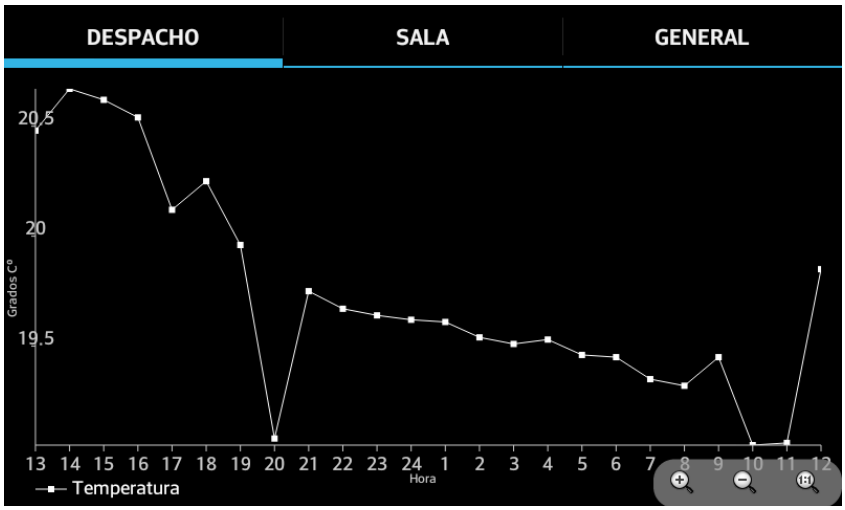

Ilustración 48. Gráfica de temperatura de las últimas 24 horas

En la ilustración 49, se puede ver el comportamiento de la temperatura a lo largo de un día. Gracias a que este sensor tiene una precisión de 0.01 Celsius, se pueden apreciar los minúsculos cambios de temperatura durante el transcurso de la noche. Durante el día hay variaciones más significativas debido a la apertura de puertas y ventanas.

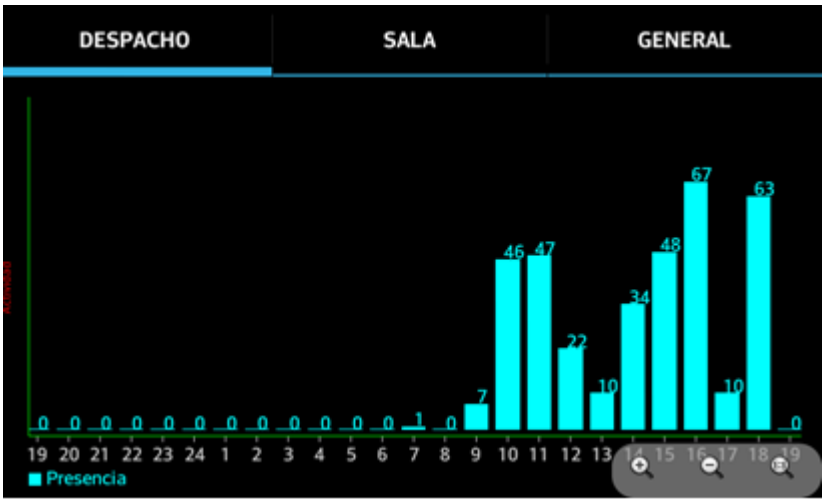

Ilustración 49. Gráfica del número de alarmas de presencia por hora.

En la ilustración 50, podemos ver el número de activaciones del sensor de presencia por cada hora. Con esta información podemos ver la cantidad de actividad en la estancia. Además no es necesario que la alarma esté activada para recoger estos datos.

9.3 Cámara

Una vez pulsado el botón de cámara de una estancia, automáticamente aparece a pantalla completa la imagen de la cámara en tiempo real. Si se pulsa el botón del menú del dispositivo, emerge un dialogo para configurar la dirección de la cámara. Para mover la cámara, se puede hacer mediante la pantalla táctil del dispositivo.

9.4 Comunicación de alarmas

Cuando se produce una alarma, se activa la aplicación, en el caso de qué no esté, y se muestra en una ventana emergente, el tipo de alarma, en que estancia se ha producido y la probabilidad.

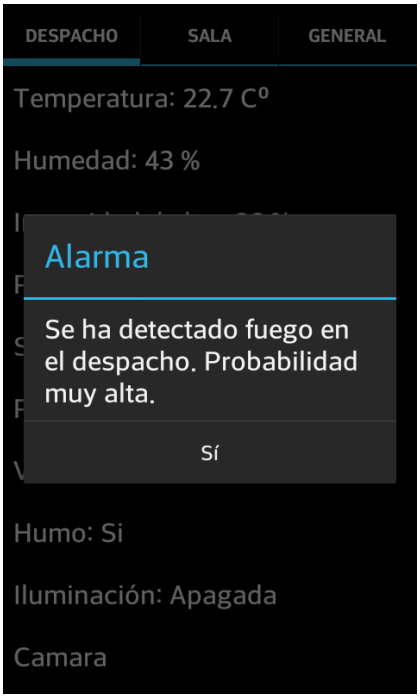

Ilustración 50. Ejemplo de visualización de una alarma.

9.5 Opciones generales

Esta pantalla es única independientemente del número de estancias del sistema, en ella se puede ver y modificar parámetros que afectan a todo el sistema en general. Todas las opciones de esta pantalla establecen una comunicación con el servidor.

Entre todas las opciones, cabe destacar el botón de pánico, ya que además de generar una alarma de pánico, una vez que es pulsado, el terminal guarda silencio

total hasta la siguiente vez que se ejecute la aplicación. Esto es debido a que si el teléfono sigue emitiendo alertas después de pulsar el botón, podría crear una situación indeseable para el usuario.

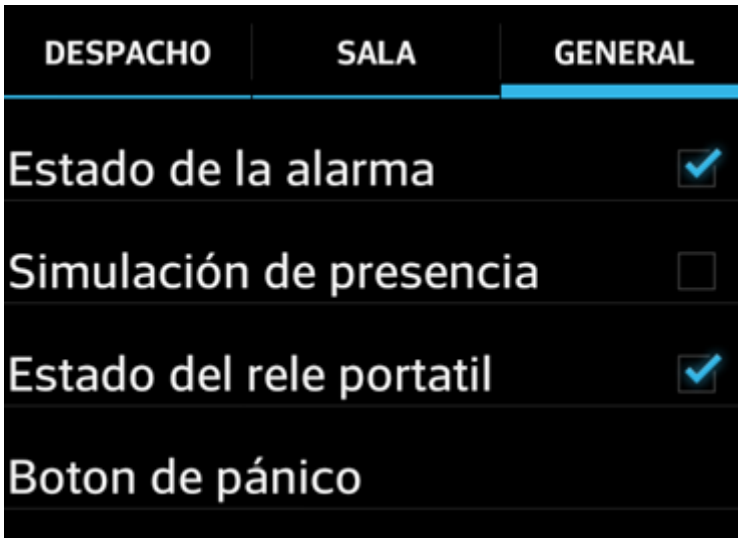

Ilustración 51. Pestaña general de la aplicación Android.

9.6 Opciones de la iluminación y relés

Si pulsamos en el botón de la iluminación, obtendremos un menú despegable indicando si queremos encender o apagar, establecer un evento diario, o un evento puntual. Para diseñar las ventanas para seleccionar las fechas y horas, se ha utilizado el elemento ListView, y los elementos de selección de fechas sobre ventanas emergentes. Así no tendremos que preocuparnos por el tamaño de la pantalla del dispositivo, ya que las listas y ventanas emergentes son fácilmente adaptables. También se han añadido las comprobaciones necesarias para evitar

errores del usuario, como añadir una fecha de inicio anterior a la actual o una fecha de fin anterior a la de inicio.

Ilustración 52. Ventana de configuración del evento puntual de un relé

9.7 Menú

El menú se despliega después de pulsar el botón del menú del dispositivo, en él se pueden realizar las acciones detalladas a continuación.

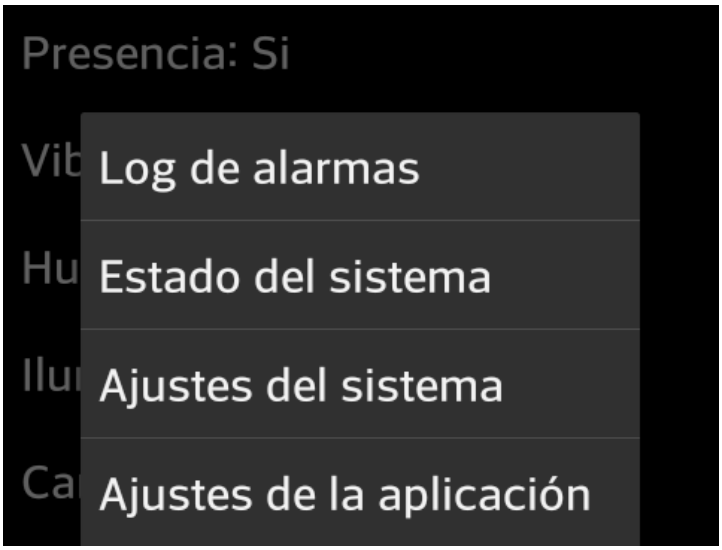

Ilustración 53. Menú de la aplicación.

9.7.1 Log de alarmas

El log de alarmas muestra las 120 últimas alarmas de cada sensor, parámetro modificable en el servidor. Tiene la finalidad de que el usuario pueda analizar detalladamente en qué momento se han producido las alarmas. También se guardan las alarmas generadas, aunque la alarma general del sistema no esté activada.

Ilustración 54. Log de alarmas del sistema.

9.7.2 Estado del sistema

La pantalla de estado del sistema muestra si todas las secciones del sistema están funcionando correctamente. La adquisición de datos tiene un especial interés ya que indica en tiempo real la velocidad con la que se reciben los datos de los sensores. Esta característica es muy útil ya que indica la calidad de la conexión entre los nodos y el ordenador central y puede utilizarse para ajustar las posiciones de las antenas para maximizar la velocidad.

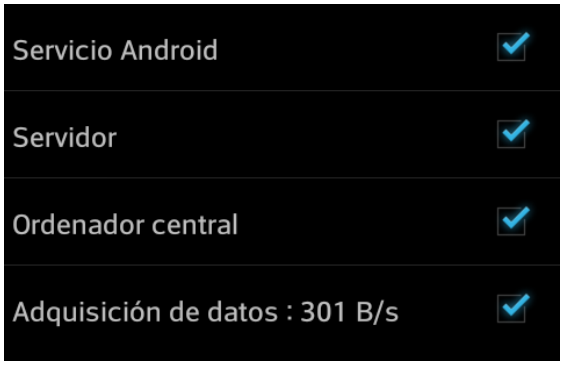

Ilustración 55. Ventana del estado del sistema.

9.7.3 Ajustes de la aplicación

En los ajustes de la aplicación se pueden configurar solo ajustes relacionados con la aplicación, cada dispositivo diferente tiene sus propios ajustes únicos.

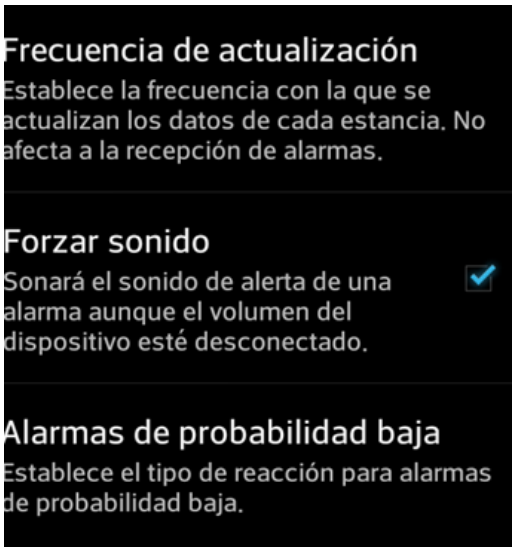

Ilustración 56. Ventana de configuración de parámetros de la aplicación.

9.7.4 Ajustes del sistema

Los ajustes del sistema contienen parámetros que afectan a todo el sistema, si desde un dispositivo se modifica un parámetro, todos los demás también lo pueden ver, ya que se queda guardado en el servidor.

Ilustración 57. Ventana de ajustes del sistema.

En la elaboración de la interfaz de *teléfonos para la recepción de alarmas*, se cometió un error debido a que no se tenía experiencia previa en la elaboración de estas interfaces para Android. La primera interfaz diseñada se muestra a continuación en la ilustración 59.

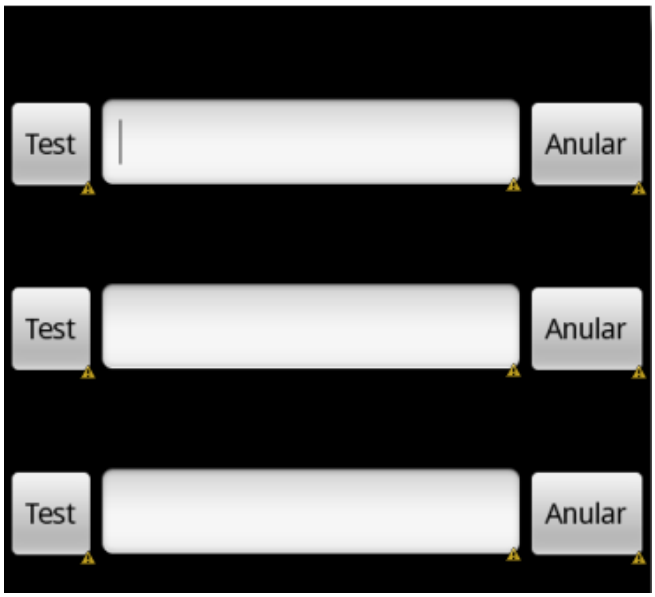

Ilustración 58. Primer diseño de la interfaz de teléfonos.

Esta interfaz fue probada con éxito en el simulador y en un dispositivo Android de 4.3". Pero al probarlo en un dispositivo real de 3,3", los botones quedaron deformados o escondidos. Aplicando un "parche" a este interfaz, podría ser posible que funcionara bien en ambos dispositivos, pero este no es el camino a seguir en un mundo en el que pueden existir dispositivos totalmente diferentes ya que es imposible probarlo en todos.

Es entonces cuando se comprendió que el desarrollo de este tipo de pantallas no se tiene que elaborar en una pantalla estática, e insertando elementos en ella si no utilizando elementos dinámicos como listas y diálogos emergentes. A partir de estas reflexiones se diseñó la interfaz definitiva para la gestión de los teléfonos que se puede ver en la ilustración 59 y 60.

Ilustración 59. Gestión de teléfonos.

Ilustración 60. Opciones de teléfono en una lista emergente de cinco elementos, donde el último es una entrada de texto.

9.8 Tiempos de espera

La aplicación realiza peticiones constantes al servidor con determinadas acciones por parte del usuario, estas peticiones requieren un tiempo de espera hasta que se completen. Dependiendo del tipo de petición el tiempo de espera variará, ya que no es lo mismo pedir el dato de un sensor que el log de alarmas de todos los sensores.

Para que el usuario pueda ver claramente que se está realizando una operación, se ha utilizado el elemento de Android ProgressDialog, que se visualizará durante toda la duración de la operación.

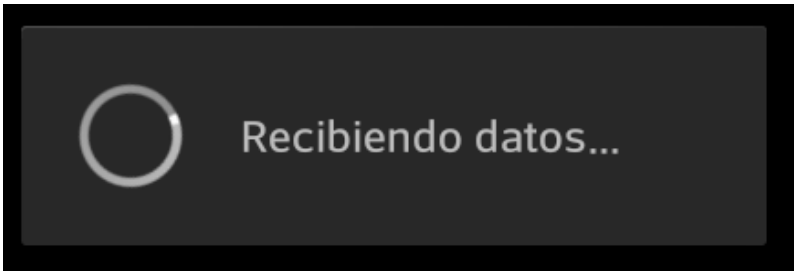

Ilustración 59. Ejemplo de elemento ProgresDialog de Android.

9.9 Pruebas

Durante el transcurso de la implementación del proyecto se han realizado pruebas unitarias simples, como las mencionadas a los sensores. O pruebas de comunicaciones con todos los diferentes pares implicados. Pero las pruebas de mayor importancia, han sido las realizadas desde la interfaz Android, con todo el producto físico terminado.

Se ha tomado esta decisión, debido a que la inclusión de un elemento hardware en un nodo puede afectar el comportamiento de los demás elementos, por lo tanto, si se hubiesen realizado pruebas estrictamente rigurosas en cada ampliación del sistema, hubiese requerido mucho conste temporal, ya que en una ampliación siguiente se pueden estar modificando los comportamientos verificados en una prueba anterior.

Por esa razón todas las pruebas finales se han realizado desde la interfaz Android, de esta manera se han realizado pruebas simultaneas de todo el sistema. Para que esto sea posible, se ha utilizado el sistema de log en todas las partes del sistema, *interfaz – servidor – ordenador central.* De esta manera una vez que se produce un error, es fácilmente identificable haciendo posible su reparación.

Para realizar esta prueba, el sistema ha estado en funcionamiento durante 720 horas continuadas o un mes. Durante este tiempo, se le ha sometido a un uso intensivo por parte de la aplicación Android, con tres dispositivos conectados de forma simultánea periódicamente y al menos siempre estando uno de ellos conectado.

Como resultados de estas pruebas, se ha corregido pequeños errores en todas las partes del sistema y se han añadido mecanismos de robustez como los comentados en los capítulos 5.2.3 y 6.2.4.

También se ha observado puntualmente una pérdida parcial de cobertura con los dispositivos Xbee. Se ha comprobado utilizando la pantalla de estado del sistema, donde se ha observado una reducción de velocidad de hasta 75% en algunas ocasiones, esta reducción no hace que el sistema deje de funcionar siempre y cuando no se reduzca totalmente la adquisición de datos. Este comportamiento anómalo desaparece cuando se acercan el emisor y receptor y surge muy puntualmente cuando el emisor y receptor están a una distancia superior a 16 metros, como el rango de estos dispositivos es de 100 metros, se puede considerar que este comportamiento se podría producir por una fuente de ruido externa. Ya que en el mismo manual del dispositivo indica que el rango es más reducido para las ciudades por el ruido. Como el sistema está implantado en una zona de alta densidad de puntos WiFi, esta podría ser una de las posibles causas. Para solucionar este problema, se podría intentar buscar la fuente de ruido o utilizar antenas más potentes en los dispositivos Xbee que las de fábrica. Pero no se han tomado estas medidas ya que cambiando la localización de los dispositivos a una más elevada, se ha reducido la perdida de cobertura, pero en el caso de que el producto fuera a ser comercializado sería un aspecto a tener en cuenta.

10. Gestión del proyecto

En este capítulo, se describirán todos los aspectos relacionados con la gestión del proyecto. En primer lugar analizaremos la evolución del alcance e hitos del proyecto, después describiremos los costes del proyecto y para terminar se presenta la gestión de los riesgos.

10.1 Evolución del alcance e hitos del proyecto

El desarrollo de este proyecto se ha dividido en diferentes fases. En primer lugar describiremos los contenidos de cada fase, para más tarde mostrar el diagrama de hitos. Las fases del proyecto han sido, *prototipo funcional con un sensor*, *prototipo funcional con varios sensores*, *ampliación del sistema* y *mejora del sistema y pruebas*

10.1.1 Prototipo funcional con un sensor

En esta primera parte del proyecto se ha diseñado e implementado la funcionalidad completa de un sensor, desde la recogida de datos en el sensor, hasta un desarrollo de una interfaz trivial en Android, incluyendo toda la arquitectura necesaria para un único sensor. Se ha elegido un sensor de temperatura debido a su sencillez y a la fácil verificación de los datos obtenidos.

La finalidad de esta fase, ha sido tener información práctica de los costes que requiere diseñar e implantar un sistema de este tipo, para poder concretar los siguientes hitos del proyecto, así como investigar las tecnologías útiles para el proyecto y poder asegurar con certeza lo que se puede y no se puede hacer.

10.1.1 Prototipo funcional con varios sensores

En esta segunda fase del proyecto, se ha diseñado e implementado una arquitectura, que soporte el manejo de N sensores, poniendo especial interés en la escalabilidad del sistema así como la sustitución de elementos. Como interfaz, se ha reutilizado el de la fase anterior, sin pensar en la interacción persona-computador.

Para probar la nueva arquitectura, se han utilizado cuatro sensores, con la particularidad de que dos de los cuatro sensores, miden la temperatura ambiente, aunque por diferentes métodos, de manera analógica y digital, estos dos sensores se han utilizado para comprobar qué dificultad supone hacer un cambio de sensor si son de diferente tipo.

10.1.2 Ampliación del sistema

En esta fase, se producen cambios significativos en todas las partes del sistema.

En cuanto a las comunicaciones en general, se implementan todas las clases necesarias para que la comunicación se produzca en los dos sentidos, hasta ahora solo podíamos leer datos de sensores, pero ahora también podemos enviar órdenes a los actuadores.

En la adquisición de datos, surge el concepto de *nodo*, para agrupar un mismo conjunto N sensores conectados al ordenador central. Se diseña e implementa esta arquitectura.

En la aplicación Android, se desarrolla completamente su diseño incluyendo las interfaces de usuario.

10.1.3 Mejora del sistema y pruebas

En esta fase se añaden elementos que no afectan a la arquitectura global del sistema, a diferencia de la fase anterior

Se cumplen objetivos adicionales como añadir el visionado de una cámara IP integrada en la aplicación Android o la integración en el ordenador central de un sistema para generar llamadas telefónicas, incluyendo un sintetizador de voz.

Se arreglan fallos menores que afectan a la Interfaz de usuario.

Se mejora la robustez de todas las partes del sistema para que genere menos fallos y sea más tolerante ante ellos y se prueba el sistema completo.

10.1.4 Diagramas de Gantt y de hitos

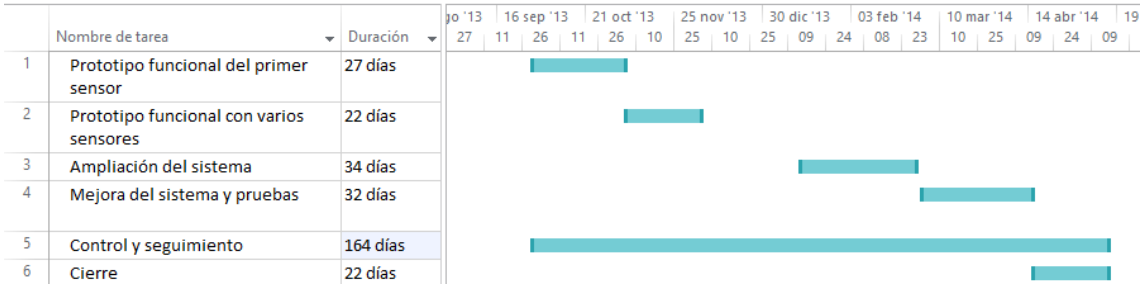

Ilustración 60. Diagrama Gantt.

En el diagrama de Gantt se muestra la duración de las diferentes tareas del proyecto. El contenido de las fases de desarrollo ha quedado detallado en el comienzo de este capítulo y la fase de cierre engloba todas las tareas relacionadas con el cierre del proyecto, como la entrega de la memoria y la defensa.

Ilustración 61. Diagrama de hito.

En el diagrama de hitos, se pueden ver los hitos más significativos del proyecto, corresponden a la fecha de fin de cada fase y otros eventos significativos como el fin del producto físico, la finalización de la documentación y la entrega.

10.2 Gestión de los costes

En este apartado se describen los costes del proyecto, en primer lugar el tiempo utilizado y en segundo lugar los costes económicos.

10.2.1 Tiempo

Tarea	Estimación	Desviación
Prototipo funcional del primer sensor	135	+8
Prototipo funcional con varios sensores	105	+24
Ampliación del sistema	220	+8
Mejora del sistema y pruebas	150	+16
Total	610	+55

Lo que supone un total de 665 horas, si tenemos en cuenta las 55 horas no previstas inicialmente.

10.2.2 Costes económicos

En este proyecto, los únicos costes apreciables son costes generados por todos los elementos hardware, a continuación quedan detallados en dólares, ya que es la moneda que se ha utilizado al ser compras internacionales.

Elemento	Unidades	Coste
Sensor de movimiento HC-SR501	2	2.35 $
Sensor de sonido para Arduino	2	5.55 $
Sensor de presión BMP085	2	4.84 $
Sensor de humo MQ-2	2	5 $
Acelerómetro ADXL345	2	4.10 $
Fotorresistencia 10k	5	0.43 $
Sensor de humedad dht 22	2	6 $
Módulo relé	4	7 $
Xbee pro series 1	4	148 $
Fuente de alimentación 220VAC – 5VDC – 1A	3	7.56
Arduino pro mini	6	16.08 $
Arduino Mega	1	20 $
Arduino uno	1	15 $
Raspberri Pi modelo B	2	70 $
Modem GSM sim 900	1	24.75 $
Buzzer	1	0.75 $
Cables, estaño y otros elementos	1	10 $
Cámara Wanscam JW0008	1	42 $
Total		389.41 $

Si hacemos la conversión a Euros, la suma asciende en total a 283 Euros.

10.3 Gestión de riesgos.

En este capítulo, analizaremos cada riesgo, cuantificándolo por su importancia, para ello analizaremos la probabilidad de que ocurriera y las consecuencias en el caso de que ocurra.

- **Perdida de información de algún documento o archivo del proyecto**. La probabilidad es alta, ya que la máquina con la que se ha trabajado en el proyecto es portátil. Existe la posibilidad de perder la información del disco duro por un error, o que la máquina física sea destruida, ya sea por un accidente o por un robo. Las consecuencias de producirse este riesgo supondrían un impacto muy alto en el proyecto. Para reducir el impacto, todos los días se ha hecho una copia de seguridad de todo el material modificado. Ahora el impacto se considera bajo, ya que solo supondría una pérdida de tiempo por la restauración.
- **Problemas en la estimación del tiempo**. Debido a que este proyecto contiene tareas que pertenecen a distintos campos, y no se tiene experiencia anterior en muchas de las tareas concretas a realizar en el proyecto, la probabilidad de que las estimaciones de tiempo sean incorrectas es alta. Para reducir el impacto, se ha planificado el proyecto por fases, definiendo al inicio de cada fase los objetivos a cumplir, además de presentar periodos de contingencia, en caso de desviarse. Ahora la probabilidad es media y el impacto bajo.
- **Fallo en un elemento hardware**. La probabilidad es media y el impacto es medio o alto dependiendo el tipo de componente. En este caso, tendríamos que pedir el nuevo componente con un periodo de espera que puede variar entre dos y seis semanas. Para reducir el impacto cuando ocurra este riesgo, se han adquirido una unidad más de lo necesario, para todos los elementos de bajo coste y también para los elementos de importancia alta aunque el coste sea alto. Un componente es de importancia alta, si cuando falla, el sistema no puede seguir funcionando de ninguna manera. Después de las medidas tomadas, la probabilidad no varía, pero el impacto es bajo, ya que solo supondría un coste de tiempo por cambiar el elemento.

A continuación se resumen para cada riesgo su valoración, en los resultados de la tabla siguiente ya se han tenido en cuenta las medidas de contingencia tomadas para cada riesgo.

Riesgo	Probabilidad	Impacto	Valoración
Perdida de información de algún documento o archivo del proyecto.	Alta	Bajo	Media
Problemas en la estimación del tiempo.	Media	Bajo	Media
Fallo en un elemento hardware.	Media	Bajo	Media

De los riesgos gestionados al comienzo del proyecto, el único que se ha producido es, *problemas en la estimación del tiempo,* pero el impacto de este riesgo no ha sido para nada significante debido a los amplios márgenes en la planificación de las fechas.

Por otro lado, sí que se ha producido un riesgo que no se gestionó al inicio. Ya que en la adquisición de determinados componentes electrónicos no se previó su funcionamiento exacto antes de adquirirlos y al adquirirlos se comprobó que la calidad real del dispositivo era menor de la esperada, el caso más significativo se ha producido con el sensor de sonido, al que se le han tenido que ir añadiendo "parches" hasta mejorar el funcionamiento del dispositivo. Podrían haber sido totalmente evitables si desde un principio se hubiera adquirido un dispositivo de un coste económico algo mayor y calidad también superior.

11. Conclusiones y líneas de trabajo futuras

Los objetivos de este proyecto, se han cumplido satisfactoriamente, ya que se ha adquirido una experiencia muy valiosa en la creación de aplicaciones Android, debido a los variados elementos utilizados en la creación de la misma, como puede ser el uso de actividades, servicios, o BroadcastReceiver.

También se ha adquirido experiencia en el diseño de sistemas que requieran una comunicación múltiple entre diferentes dispositivos, además de adquirir una experiencia práctica del manejo de datos en sensores en tiempo real, ya que no es lo mismo utilizar una simulación, que encontrarte con problemas reales de ruido y otros efectos que se producen continuamente en el mundo real.

En cuanto a los objetivos técnicos descritos en el alcance, también se han podido cumplir todos ellos, aunque con una desviación del tiempo, debido a que uno de los principales riesgos, es la adquisición de materiales necesarios para el proyecto y ha ocurrido una pequeña incidencia en este aspecto.

Para abordar la estandarización en la adquisición de datos, se han catalogado los diferentes tipos de sensores en tres categorías, en función de la salida de sus datos, de esta forma, podemos cambiar un sensor por otro, o ampliar el número de sensores indicando únicamente a que clase pertenece. Tiene que quedar claro, que esta categorización solo se puede producir utilizando sensores del bajo nivel como los utilizados en el proyecto, ya uno de los objetivos mencionados es la construcción y el entendimiento del funcionamiento de un sistema de sensores desde el más bajo nivel, y no partiendo de protocolos como “UPnP”.

11.1 Líneas de trabajo futuras

Durante el transcurso del proyecto, han surgido nuevas ideas, que podrían ser implementadas en un desarrollo futuro, a continuación se describen las más significativas.

- **Asistente de configuración dinámico.** Actualmente los datos de la estructura del sistema son los siguientes. Cuantos nodos tiene el sistema, cuantos sensores en cada nodo, de que tipo es cada sensor, si el sensor puede generar alarmas, a partir de que umbral se tiene que considerar que un sensor genera una alarma, etc. Todos estos datos, actualmente están definidos en el propio código. Pero sería muy interesante desarrollar un asistente de configuración, en el que se puedan definir todas estas constantes por parte del propio usuario o técnico, mientras se analizan los datos de los sensores.

- **Estudiar una ampliación con el protocolo UPnP.** Se podría estudiar la interacción con dispositivos UPnP encontrándolos adentro de la red Wi-Fi del hogar y posibilitando la interacción con electrodomésticos de este protocolo. Como punto de partida, se podría añadir una antena Wi-Fi al ordenador central para detectarlos desde ahí.
- **Mejora de los almacenamiento y visualización de datos.** En la solución actual, se utiliza la clase *sensores* para almacenar los datos del servidor. Se podría sustituir esta clase, por un gestor de base de datos, posibilitando un almacenamiento más continuo y a largo plazo de los datos. Por ejemplo para el caso de los sensores, se guardan los datos de cada hora durante las últimas 24 horas, se podría ampliar obteniendo datos cada 15 minutos y guardándolos durante incluso meses.
- **Geolocalización con el botón de pánico.** Es verdad que este sistema está diseñado para establecimientos, que son lugares estáticos, y cuando alguien pulsa el botón de pánico se presupone que está en el establecimiento. Pero sería muy interesante añadir posibilidades de "tracking", para seguir todas las posiciones de la persona después de haber pulsado el botón.
- **Botón de pánico oculto.** Se puede dar el caso de que en determinadas situaciones, no se pueda pulsar el botón de pánico sin que un posible atacante se percate de ello. Para accionar este botón sin llamar la atención se podrían añadir determinadas acciones, como un botón de apagado falso, o realizar un determinado movimiento con el dispositivito y detectarlo con el acelerómetro.
- **Múltiples ampliaciones en el área de la domótica.** A partir del trabajo realizado, se pueden incluir múltiples sensores y actuadores utilizados en el ámbito de la domótica. Como sensores de inundación para baños y cocinas, o controles para motores de persianas. También se podría añadir a todos los relés, sensores de potencia eléctrica para medir el consumo, y en función del precio eléctrico actual, elaborar múltiples estadísticas de consumo eléctrico.
- **Funcionalidades añadidas en las cámaras.** Se podrían integrar más funcionalidades de las cámaras de vigilancia en el proyecto, utilizándolas como otro sensor para detectar movimiento mediante el procesamiento de imagen. Además de diseñar un sistema con el fin de respaldar las grabaciones de las últimas horas en el servidor.
- **Conectividad a internet por GSM en el ordenador central.** Para que el ordenador central se conecte a internet, se utiliza el router que dispone el establecimiento, y en caso de que este falle, por fallo del mismo o por un problema del ISP, el ordenador central se quedaría sin conexión. Por eso puede ser interesante estudiar la posibilidad de utilizar la red de datos GSM

con el modem GSM ya presente en el proyecto, para tener un medio complementario para conectarse a Internet en caso de fallar el principal.

- **Actualización del software de los nodos.** Actualmente se puede actualizar tanto el software de la interfaz Android, como el servidor y el ordenador central de manera remota. Pero para actualizar el software de las tarjetas Arduino hay que hacerlo manualmente accediendo a ellas físicamente. Pero los dispositivos Xbee, tienen una funcionalidad para insertar código en las tarjetas Arduino de manera remota, con esto se podría implementar en el ordenador central los mecanismos necesarios para actualizar el código de las tarjetas de esta manera por medio inalámbrico.
- **Mejora del hardware de los nodos**. Arduino es una plataforma que permite crear prototipos electrónicos de manera muy rápida, por tanto el hardware creado en este proyecto es un prototipo, es decir, que las tarjetas, conectores y numerosos elementos electrónicos no están adaptados estrictamente a nuestro proyecto, por poner un ejemplo la tarjeta Arduino Mega del ordenador central tiene más de cincuenta entradas digitales, de las cuales no usamos más de dos, además como cada sensor está construido en su propio módulo hardware, además de estar separados físicamente se utilizan elementos electrónicos que posiblemente se podrían simplificar después de un análisis detallado. Gracias a que Arduino es hardware libre, se pueden disponer de todos sus esquemas detallados gratuitamente. Con esta información y con un experto en la construcción de circuitos, partiendo del prototipo funcional de este proyecto, se puede diseñar una placa, que incluya de forma integrada todos los elementos necesarios para cada nodo de tal forma que se podría reducir el tamaño con respecto al prototipo significativamente, y se podría reproducir en masa fácilmente.

11.2 Lecciones aprendidas

- Para probar aplicaciones Android, no es recomendable tener confianza ciega en el emulador ni en un solo dispositivo concreto. Lo ideal es utilizar dos dispositivos físicos totalmente diferentes en cuanto a resoluciones de pantalla y versión del sistema, combinándolo con pruebas eventuales en diferentes dispositivos emulados.
- Para desarrollar interfaces en Android no hay que utilizar pantallas estáticas que puedan verse afectadas ante dispositivos diferentes, sino que hay que utilizar elementos como tablas, listas o ventanas emergentes.
- Para utilizar los diferentes elementos gráficos de Android, como listas, menús o ventanas emergentes. Hay que utilizar una estructura de código concreta, donde se inicializan los elementos y se definen las acciones y eventos de los

elementos. La primera vez que se utiliza un elemento nuevo, el tiempo necesario para implementarlo es alto debido al desconocimiento de su funcionamiento. Por esa razón, es muy conveniente crear una librería propia con estas estructuras para agilizar su reutilización. Y no solo con el desarrollo para Android, sino también para cualquier otro lenguaje en el que se recurra al reusó de estructuras.

- En un sistema como el implementado en el que para que un dato llegue desde el origen hasta el destino tiene que pasar por diferentes subsistemas. Muchas veces al implementar una nueva característica no es fácil saber que parte del sistema está fallando, y comprobarlo requiere un gasto de tiempo alto. Por esa razón es recomendable en primer lugar desarrollar un sistema que monitorice la información cuando pasa por cada subsistema, es decir un log, y después implantar el sistema completo con todas sus características.
- Partes del código implementado solo se ejecuta en determinadas situaciones muy puntuales que se producen muy difícilmente. Aunque el código sea claro y preciso, no hay que asumir que funcionará cuando esta situación se produzca. Si no que hay que probar este código de forma unitaria forzando estas situaciones.
- Es de mucha utilidad, dividir las diferentes partes del código en funciones lo más reducidas posibles, ya que al añadir nuevas características sobre un código ya implementado facilitan en gran medida la reusabilidad.
- Antes de pedir un determinado componente electrónico, es conveniente estudiar detalladamente la documentación disponible, para entender cómo se comportará el componente una vez adquirido y si es realmente lo que necesitamos, incluso buscar opiniones de personas que hayan comprado el mismo producto para prevenir los posibles problemas.
- En la adquisición de datos de sensores, cualquier fenómeno puede alterar su funcionamiento. Por ejemplo si los datos que adquirimos los enviamos por un puerto para poder visualizarlos, ese simple hecho ya está modificando la velocidad de muestreo, y por lo tanto alterando los datos. O si utilizamos una fuente de alimentación concreta, el “rizado” de esta fuente puede estar alterando los datos, también si añadimos más sensores alimentados por la misma fuente puede cambiar el comportamiento de estos. Por lo tanto, hay que tener en cuenta que cuando hacemos una modificación en un sistema electrónico, por pequeña que sea puede estar alterando otro funcionamiento del mismo. Por esa razón, después de cada actualización hardware o cambio mínimo conviene hacer pruebas rigorosas de todo lo implementado con anterioridad.

- En un proyecto como este en el que ofrece muchas posibilidades de desarrollo, es fácil dispersarse en otras ideas que surgen mientras se está desarrollando. Pero es conveniente centrarse primero en cumplir los objetivos actuales y después estudiar las posibles ampliaciones.
- Sin una experiencia previa, no es posible desde un principio hacer una planificación de un proyecto de una duración de casi un año y pretender que se cumpla la planificación. Por eso es recomendable hacer una planificación más a corto plazo dividiendo el proyecto por fases y planificar cada una.

Referencias

- Sistema de sintetizador de voz, https://wiki.archlinux.org/index.php/Festival
- Guía de programación de Arduino, http://www.arduino.cc/en/Reference/HomePage
- Guía de Android, https://developer.android.com/guide/index.html
- Comunidad profesional para desarrolladores, http://stackoverflow.com/about
- Api de Python, https://docs.python.org/2/library/
- Documentación de Amazon EC2, http://aws.amazon.com/es/documentation/ec2/
- Guía de SIM900, ftp://imall.iteadstudio.com/IM120417009_IComSat/DOC_SIM900_AT%20Command%20Manual_V1.03.pdf
- Librería de Android achartEngine , http://www.achartengine.org/
- Librería de Python streaming pickle, https://code.google.com/p/streaming-pickle/
- Wikipedia, http://es.wikipedia.org

Agradecimientos

En primer lugar quiero agradecer a mis padres el apoyo que me han dado a lo largo de toda la carrera.

También quiero agradecer la ayuda que me ha prestado mi director de proyecto José Miguel Blanco, al cual agradezco todos sus consejos.

Finalmente, no me quiero olvidar de aquellos familiares, compañeros y amigos que también me han apoyado con sus mensajes de ánimo.