Control de dispositivos con Android

Memoria del proyecto
de Ingeniería Técnica en
Informática de Gestión
realizado por
Albert Férriz Pérez
y dirigido por
Marc Tallo Sendra

Escola d'Enginyeria
Sabadell, *Septiembre* de 2013

FULL DE RESUM – PROJECTE FI DE CARRERA DE L'ESCOLA D'ENGINYERIA

Títol del projecte: Control de dispositivos con Android	
Autor[a]: Albert Férriz Pérz	**Data: Septembre, 2013**
Tutor[a]/s[es]: Marc Tallo Sendra	
Titulació: Enginyería Tècnica en Informàtica de Gestió	

Paraules clau (mínim 3)

1. Català: Android, Raspberry Pi, Control, Remot, Temperatura, Calefacció
2. Castellà:Android, Raspberry Pi, Control, Remoto, Temperatura, Calefacción
3. Anglès: Android, Raspberry Pi, Control, Remote, Temperature, Heating

Resum del projecte (extensió màxima 100 paraules)

4. Català:

   Aquest projecte de final de carrera de la titulació d'Enginyería Informàtica de Gestió consisteix en el disseny i desenvolupament d'un sistema de control de dispositius connectats a Raspberry Pi i gestionats des d'un dispositiu Android. Es podran portar a terme diferents accions com encendre, apagar, programar la calefacció i consultar la temperatura actual remotament des d'un dispositiu connectat a la red wifi domèstica o des de qüalsevol altre xarxa connectada a internet.

5. Castellà:

   Este proyecto de final de carrera de la titulación de Ingeniería en Informática de Gestión consiste en el diseño y desarrollo de un sistema de control de dispositivos conectados a Raspberry Pi y gestionados desde un dispositivo Android. Se podrán llevar a cabo diferentes acciones como encender, apagar, programar la calefacción y consultar la temperatura actual remotamente desde un dispostivo conectado a la red wifi doméstica o desde de cualquier red que esté conectada a internet.

6. Anglès:

   This is the final project of Engineering degree in Computer Science and it is about the design and development of a control system with Raspberry Pi devices connected to and managed from an Android device. It can perform various actions such as turn on and off, program the heating and check the current temperature remotely from one device connected to the home wireless network or from any network that is connected to the Internet

Tabla de contenidos

1 introducción

1.1 Presentación

El objetivo de este proyecto es crear un sistema de gestión de dispositivos que podemos tener en casa, y realizarlo todo ello remotamente desde dispositivos móviles.

La gestión incluye que el usuario pueda "programar" diferentes características de un dispositivo, como podría ser las horas de encendido de la calefacción y hacer una planificación diferente para cada día.

Además de la gestión también queremos permitir la recolección de datos (como por ejemplo la temperatura) añadiendo otros sensores a Raspberry y permitir al usuario el cómo y cuándo llevar a cabo estas tareas de monitorización.

Estos datos recogidos le proporcionarán al usuario una información valiosa para la toma de decisiones.

1.2 Objetivos

Cada vez más la sociedad está concienciada con el uso responsable de los recursos energéticos. Con este proyecto se quiere realizar un sistema que lleve a cabo una gestión y monitorización de diferentes componentes que podemos encontrar en una casa.

Queremos obtener la eficiencia energética de, por ejemplo, el sistema de calefacción basándonos en datos que hemos podido recoger.

Raspberry Pi funcionará como el sistema central, el corazón del sistema. Se encargará de monitorizar los datos de los diferentes sensores o datos que proporcionen de otros componentes conectados a Raspberry y la gestión de estos, como puede ser un simple encendido o apagado o algo más complejo como podría ser la automatización.

Volviendo al caso de la calefacción el usuario podrá programar cuando se pondrá en marcha dando, por ejemplo, horas de encendido para diferentes días, rangos de fechas, etc...

Y toda esta información, gestión, históricos y configuraciones el usuario las podrá realizar desde un dispositivo móvil Android.

1.3 Estado del arte

En los últimos años han aparecido diferentes dispositivos electrónicos que se pueden controlar mediante teléfonos móviles.

Estos dispositivos son independientes, no forman parte de un sistema: uno controla las luces de la casa, las activa o desactiva o cambiar el color de la luz u otros dispositivos como termostatos...

Como hemos comentado todos estos dispositivos al no formar parte de un sistema, cada uno tienen sus propias aplicaciones para los dispositivos móviles.

Esto puede resultar incómodo para el usuario tener que usar una aplicación para cada cosa, además de no poder obtener las ventajas que proporcionaría tener un sistema centralizado donde podemos obtener información de cada uno de los componentes.

1.4 Motivaciones

La aparición de estas micro computadoras ha facilitado la creación de proyectos que antes era imposible, debido a los altos costes que comporta el uso del hardware. La potencia y versatilidad de estos dispositivos junto con la potencia que ofrecen los dispositivos móviles hoy en día, abre un nuevo campo de desarrollo con muchísimas oportunidades de negocio.

La realización de este proyecto permite profundizar en los dos campos y relacionarlos para crear sistemas de control muy potentes.

Trabajar con estos dispositivos nos ofrece la oportunidad de crear sistemas realmente potentes y ofrecer soluciones a problemas cotidianos con unos costes considerablemente bajos.

1.5 Estructura de la memoria

En el **capítulo 1** haremos una introducción al proyecto desde el punto de vista de un cliente, donde describiremos el estado del arte, objetivos y motivaciones.

En el **capítulo 2** extenderemos la introducción hecha en el capítulo 1 y concretaremos las especificaciones del proyecto, las asignaciones de recursos y el material necesario.

En el **capítulo 3** expondremos las tecnologías que hemos usado para llevar a cabo el proyecto. En los primeros puntos introduciremos Raspberry y Android y también se detallarán sus características principales. Explicaremos también otras tecnologías utilizadas en el proyecto.

El capítulo 4 nos centraremos en detallar cuales son los requerimientos de la aplicación , la planificación, el uso de de los recursos y los costes del proyecto.

La implementación del proyecto la trataremos en el **capítulo 5**, donde mostraremos las clases creadas para llevar a cabo el proyecto y la interacción entre los diferentes componentes. También se mostraran cuales son los flujos de ejecución.

Las pruebas llevadas a cabo para comprobar que el proyecto funciona según las especificaciones y su correcto funcionamiento se describirán en el **capítulo 6**.

En el **capítulo 7** expondremos las conclusiones obtenidas con la realización del proyecto.

Por último añadiremos la bibliografía en el **capítulo 8,** donde listaremos las referencias a los documentos consultados.

Para completar la memoria, adjuntaremos en los **anexos** el **índice de tablas**, **índice de figuras** e **índice de diagramas**. También se adjuntará el **Glosario** donde se explicarán algunos de los términos usados y el diagrama de **Gantt** donde se mostrará la planificación del proyecto

2 estudio de viabilidad

Introducción

En los últimos años hemos experimentado un enorme crecimiento en el mundo de la telefonía móvil.
Los teléfonos móviles han pasado de ser utilizados sólo para llamar a convertirse en auténticos ordenadores. Con cada nuevo dispositivo disponemos de más sensores que nos permiten llevar aún más allá la experiencia del usuario.

Debido al rápido desarrollo de estos dispositivos, los sistemas operativos que los manejan (como lo es Android) han pasado a lanzar nuevas actualizaciones con muy poca diferencia de tiempo mejorando así las funcionalidades en versiones anteriores y además añadiendo nuevas. Además de mejorar la experiencia de usuario.

Esto ha abierto un infinito campo de posibilidades permitiéndonos crear aplicaciones que hagan uso de todas estas capacidades para crear software que mejore la vida de las personas.

Paralelamente a este boom de los teléfonos móviles, han aparecido otros dispositivos de hardware como son Arduino y Raspberry.

Arduino es una plataforma de hardware libre. Es un microcontrolador y un entorno de desarrollo que debido a su precio y la gran comunidad que la respalda se ha convertido en un de los proyectos más famosos dentro de la comunidad para llevar a cabo proyectos electrónicos.

Por otro lado, recientemente ha aparecido Raspberry Pi. Es una placa que podríamos definirla como una microcomputadora totalmente funcional que utiliza con un sistema operativo basado en Linux a un precio muy bajo. Esta además, tiene montadas en la misma placa unas pins de entrada y salida que permite la conexión de hardware u otros circuitos electrónicos, que podemos controlar desde la placa.

Como podemos ver todas estas tecnologías por si solas han abierto infinitas posibilidades para el mundo del software y de la electrónica.
Debido al bajo coste de estos ha permitido que muchísima gente haya podido acceder a estos dispositivos y crear proyectos que de otra manera jamás habríamos podido ver.

En este proyecto vamos a usar dos de estas tecnologías, Android y Raspberry Pi, para llevar a cabo un sistema que gestionará y monitorizará los diferentes dispositivos que podemos encontrar en casa.

2.1 Objetivos

El objetivo del proyecto es de poder crear un sistema compuesto por diferentes componentes.

Raspberry Pi será el corazón del sistema. Se encargara de responder a las peticiones que se envían desde los diferentes dispositivos móviles, como puede ser la consulta en tiempo real de la temperatura. También se encargará de la comunicación recolección de datos de los diferentes dispositivos y los realizará con la configuración que el usuario le proporcione: Monitorización de la temperatura cada X segundos, obtener estadísticas y realizar históricos.

La otra parte del proyecto será la creación del programa para Android, donde el usuario llevará a cabo todas las acciones, como la consulta de la temperatura actual, programación de las horas de encendido y apagado de la calefacción, estado de las luces de casa, etc...

2.1.1 Estado del arte

El poder gestionar diferentes componentes de casa, ha implicado grandes sistemas de gestión que se han ocupado completamente de la gestión de todos y cada uno de los componentes de una casa a precios desorbitados.

La aparición de componentes como Raspberry Pi o Arduino ha permitido el acceso a la creación de hardware y software para la creación de proyectos, los cuales antes sólo podían ser creados por compañías que disponían de un gran capital.

Con este proyecto queremos crear dispositivos a bajo coste para gestionar diferentes dispositivos que tenemos en casa

Nest **(**http://nest.com/**)**

Nest es un termostato inteligente. Aprende del uso que se le da favoreciendo así el consumo de energía y se puede controlar remotamente desde un teléfono inteligente, una tablet o desde un navegador web.

Wemo (http://www.belkin.com/us/wemo)

Wemo desarrollado por Belkin ha creado unos enchufes con wifi, que permite encender y apagar los dispositivos que estén conectados a este enchufe. Actúa como un interruptor wifi.

Philips Hue (https://www.meethue.com/es-US)

Philips hue son unas bombillas led conectadas a internet. Podremos controlar estas bombillas desde un teléfono inteligente permitiéndonos encenderlas, apagarlas o programar estas acciones, incluso cambiar el tono de luz de estas.

2.2 Especificaciones

2.2.1 Requisitos funcionales

Para facilitar la lectura y la comprensión, pasaremos a usar la palabra "**Servidor"** para referirnos a **Raspberry Pi** y "**Cliente**" para referirnos al **dispositivo Android**.

· Al iniciarse la aplicación se quedará a la escucha de peticiones de los diferentes clientes.

· Podrá recibir solicitudes de clientes que estén en la misma red local o desde internet.

· Cuando el cliente quiere hacer una petición u no conoce la dirección del Servidor, envía un paquete UDP a la dirección de broadcast con un código de operación.

· El servidor recibe la petición de descubrimiento y responde al cliente con la ip del servidor.

· Las siguientes comunicaciones se harán usando el protocolo TCP.

· El cliente cada vez que solicite una información al servidor, deberá enviar un código de operación que identificará cual es la acción que se debe llevar a cabo.

· El servidor creará un proceso aparte que llevará a cabo la petición requerida.

· El cliente podrá consultar la temperatura actual.
El servidor enviará esta información al cliente cuando:
- El cliente no envíe un código de fin de operación.
- Si la temperatura ha cambiado desde la última vez que se envío.

· El cliente podrá encender o apagar el sistema de calefacción:
- Enviará un código de activación o desactivación.

· El cliente podrá programar el encendido:
- Configurando un rango de días
- Configurando un rango de horas
- Cuando la temperatura esté por debajo de un mínimo.
- Combinaciones de las anteriores.

· El cliente podrá activar o desactivar la recolección de temperatura cada x tiempo.

· El servidor recolectará estos datos y los almacenará en la BBDD.

· El cliente podrá configurar cada cuanto tiempo se obtienen datos de la temperatura actual para guardarla en la base de datos.

· El cliente podrá consultar estos datos por:
- Rangos de fecha
- Rangos de hora

2.2.2 Requisitos no funcionales

· La comunicación entre el cliente y el servidor una vez se han descubierto se deberá cifrar.

· Los clientes sólo podrán hacer peticiones desde Internet si previamente se han registrado en el servidor desde.

· Los datos referentes a la monitorización de las temperaturas se podrán mostrar en diferentes formatos.

2.3 Planificación

El proyecto lo desarrollaremos usando el modelo en cascada (waterfall). Con este modelo dividiremos el proyecto en fases y planificaremos todas las actividades antes de empezar. Ninguna fase empezará antes de que se acabe la anterior.

1.- Documentación
2.- Análisis y definición de requerimientos-
3.- Diseño del sistema y el software.
4.- Programación.
5.- Pruebas.

2.3.1 Documentación

En esta fase investigaremos todo lo relacionado con los diferentes dispositivos con los que llevaremos a cabo el proyecto, documentarnos sobre como usar las diferentes tecnologías y usarlas a la vez.

2.3.2 Análisis y definición de requerimientos

Analizaremos que es lo que el sistema debe hacer y como interaccionarán los dispositivos. Una vez hecho este análisis podremos obtener los requerimientos funcionales y no funcionales para Raspberry y para el móvil

2.3.3 Diseño del sistema y el software

Con los requisitos y el análisis del paso anterior, pasaremos a diseñar la estructura de de la aplicación.

2.3.4 Programación

Esta es la parte más importante del proyecto y donde más tiempo vamos a invertir. En los dos pasos anteriores hemos definido qué ha de hacer y cómo ha de ser la aplicación. En esta etapa implementaremos el código de la aplicación.
Ya que ésta fase es muy compleja, la dividiremos en subtareas para facilitar la implementación:

· Programación de Raspberry Pi, que será el corazón central del sistema.
· Bases de datos, para guardar los datos que vamos recolectando.
· Programación de la aplicación para Android.
· Interfaz gráfica de la aplicación para Android.

2.3.5 Pruebas

Una vez acabada la programación de la aplicación, debemos asegurarnos que cumple con los requisitos que especificados. Para ello llevaremos a cabo diferentes pruebas para asegurar el correcto funcionamiento.

2.4 Valoración

En este apartado mostraremos todos los recursos necesarios para llevar a cabo el proyecto además de el coste de cada uno.

2.4.1 Materiales

Los componentes físicos que necesitaremos para realizar el proyecto son:

- Raspberry Pi (Modelo B)
- Dispositivo Android
- Sensor temperatura
- PC
  - Intel Core i7
  - 4GB de memoria RAM

Los costes de estos recursos son

Raspberry Pi	33 €
Dispositivo Android	324 €
Sensor temperatura	5,43€
PC	750€
Total	**1.112,43€**

Tabla 1: Costes de material

2.4.2 Software de desarrollo

Especificaremos todos los componentes de desarrollo necesarios para la programación de este.

Eclipse IDE 3.6.2

La programación tanto de la parte cliente como servidor se realizarán en Java. El desarrollo de la parte Android funciona con éste IDE.·

Java Development Kit (JDK) 6
Es el software que nos permitirá crear las aplicaciones en Java.

Android Developer Tools (ADT)
ADT es un paquete de recursos para la creación de aplicaciones para Android. Incluye el plugin para extender las funcionalidades de Eclipse para el desarrollo de Android y también contiene Android SDK Tools que son las herramientas de desarrollo de Android.

SQLite
SQLite es el gestor de Base de datos.

Ubuntu 12.04
Sistema operativo sobre el que usaremos las herramientas de desarrollo.

PHP 5.4
Es el lenguajede programación que usaremos para programar la consulta de información desde la web.

GIT
Gestor de control de versiones.

BitBucket
Repositorio online gratuito que usaremos para centralizar el control de versiones del proyecto.

Es importante remarcar que **el coste de todo este software** de desarrollo es **a coste 0**.

2.4.3 Personal

Para la realización del proyecto necesitaremos un analista, un programador y un equipo de test. El coste del personal es el siguiente:

Analista	50€/h
Ingeniero	35€/h
Tester	25€/h

Tabla 2: Coste/hora del personal

2.4.4 Desarrollo del proyecto

Tarea	Quién	Duración	Coste
Documentación	Analista	24h	1200€
Análisis y definición de requerimientos	Analista	16h	800€
	Ingeniero	8h	280€
Diseño del sistema y el software	Analista	16h	800€
	Ingeniero	8h	280€
Programación.	Ingeniero	175h	6125€
Pruebas.	Tester	40h	1000€
			10.485€

Tabla 3: Tareas y asignación

2.4.5 Coste total

Una vez desglosados todos los recursos del proyecto y sus costes relacionados, obtenemos el coste total:

Concepto	Coste total
Materiales	1.112,42
Software desarrollo.	0 €
Desarrollo	10.485€
Total	**11.597,42€**

Tabla 4: Coste del proyecto por categorías

2.5 Riesgos

Para garantizar el buen funcionamiento del sistema debemos prever posibles fallos o mal funcionamiento del sistema, de algunos de los componentes o del propio software.

Especificaremos unos requerimientos de de fiabilidad y seguridad donde tendremos los posibles eventos peligrosos que pueden surgir además de evitar otros daños derivados.

Se definirán los posibles riesgos con una descripción, la probabilidad de que este suceda (alto, medio o bajo) y la aceptabilidad de este fallo (aceptable, inaceptable).

Riesgo: Temperaturas extremas	Probabilidad: Baja	Impacto: Inaceptable

Descripción
Detección de temperaturas extremas para la zona en la que se encuentra.

Solución
Desactivar cualquier tipo de operación y avisar al usuario de unas temperaturas inusuales.

Riesgo: Acceso usuarios no permitidos	Probabilidad: Baja	Impacto: Inaceptable

Descripción
Registrar peticiones de dispositivos que no se han registrado previamente.

Solución
Bloquear todas las peticiones provenientes del dispositivo no autorizado y avisar al usuario principal de una posible intrusión.

Riesgo: Conversión errónea de unidades de temperatura	Probabilidad: Baja	Impacto: intolerable

Descripción
Se detectan solicitudes de cambio de temperatura con unos cambios muy grandes. Lo más probable es que el usuario haga cambios de pocas unidades.

Solución
Detectar solicitudes de cambio de temperatura con cambios muy bruscos. Comprobar si hay relación entre cambios de temperatura son unitarios en las diferentes unidades (celsius, fahrenheit).

Riesgo: Fallo inesperado del sistema	Probabilidad: Baja	Impacto: intolerable

Descripción
Se detectan fallos en el sistema de los que no es posible recuperarse ni obtener una solución.

Solución
Reportar del error en los logs del sistema y apagar el dispositivo para evitar males mayores.

Riesgo:Comunicación entre dispositivos	Probabilidad: Baja	Impacto: aceptable

Descripción
Se pueden producir errores a la hora de la comunicación entre o la pérdida de datos.

Solución
Usaremos tiempos de espera para la recepción de una petición.
Se reintentará el envío tres veces, cada uno con su tiempo de espera.
En caso de no poder conectar el dispositivo android deberá volver a hacer la petición de descubrimiento.
En caso de que el error persista el usuario deberá comproba la configuración de la aplicación para comprobar que los datos son los correctos.

Riesgo: Fallo en el sensor de temperatura	Probabilidad: Baja	Impacto: Inaceptable

Descripción
Durante la monitorización se detectan cambios bruscos y continuados en las mediciones de temperatura.

Solución
Guardamos el error encontrado en los logs del sistema, informamos al usuario de un posible mal funcionamiento del sensor y detenemos el sistema. El usuario deberá reemplazar el sensor.

Riesgo: Monitorización y guardado de datos	Probabilidad: Baja	Impacto: Inaceptable

Descripción
El sistema de comprobación de temperatura que se activa cada X segundos no está funcionando, o el guardado de en base de datos de la información recogida no se puede almacenar.

Solución

Comprobar si el error en la monitorización es un fallo del sensor. Si es un fallo del sensor atenderemos al "Riesgo: Fallo en el sensor de temperatura" especificado anteriormente.

Si el error es referente al guardado en base de datos, la aplicación comprobará si es un error que puede solucionar y lo reintentará. En caso contrario se detendrá la monitorización y avisará al usuario de que se ha producido un error en el guardado de datos.

2.6 Conclusiones

Hemos expuesto algunos de los productos existentes en el mercado que tienen una funcionalidad parecida a lo que pretendemos realizar en este proyecto, pero todas ellas están limitadas a una función específica y no es posible ampliarlos con otras funcionalidades.

Nuestro proyecto permitirá ampliar las funciones para gestionar otros dispositivos bien sea con nuevos dispositivos de control conectados a Raspberry Pi u ofreciendo nuevas funcionalidades a las aplicaciones de Android que controlan los diferentes dispositivos de control conectados a Raspberry.

Los costes que son uno de los factores claves para poder empezar a desarrollar el proyecto son considerablemente bajos.

Con todos estos puntos a favor **concluimos que el proyecto es viable.**

3 fundamentos teóricos

Introducción

En este capítulo presentaremos las diferentes tecnologías que se usan para desarrollar el proyecto y además de los diferentes conceptos relacionados.

3.1 Raspberry Pi

3.1.1 Introducción a Raspeberry

Raspberry Pi es un micro ordenador que que se creó con la idea de ofrecer dispositivos baratos y asequibles para todo el mundo y así poder facilitar el acceso al mundo del desarrollo de software a todas aquellas personas que estuviesen interesadas en aprender pero que no podían permitirse comprar un ordenador.

Este proyecto ha tenido una gran acogida en la comunidad educativa de todo el mundo debido a que uno de sus principales objetivos era ser un dispositivos para el aprendizaje.

Pero Raspberry Pi ha ido más allá del uso académico. Debido a las más que aceptables prestaciones que ofrece a un precio muy bajo, lo ha convertido en un dispositivo útil para muchos proyectos profesionales.

Miles de interesantes proyectos se están llevando a cabo gracias a este dispositivo, que además está fomentando la colaboración y el compartir información de la comunidad de desarrolladores.

3.1.2 ¿Qué es Raspberry Pi?

Como hemos comentado en el punto anterior, este dispositivo no es más que un placa electrónica, un micro ordenador que dispone una versión del sistema operativo Linux, llamado **Raspbian**.

Esta pequeña placa de unos 8 centímetros por 5 de ancho, dispone de componentes de entrada y salida similares a un ordenador convencional.

Pero lo que la hace muy interesante es que dispone de unas entradas llamadas **GPIO**.

GPIO es el acrónimo de General Purpose Input Output, y son unos pins que nos permiten conectar el dispositivo con el mundo real. Esto nos permite hacer cosas como activar o desactivar los diferentes dispositivos conectados a Raspberry.

Pero no sólo podemos enviar señales de encendido y apagado, parte de esos pins son diferentes tipos de conectores: existen pins para **I2C*** (Inter-Integrated Circuit), **UART*** (Universal Asynchronous Receiver-Transmitter) y **PWM*** (Pulse With Modulation).

Con todo este tipo de conexiones que nos ofrece Raspberry, podremos controlar su comportamiento con el software que nosotros escribamos.

3.1.2 Especificaciones

A continuación detallaremos las especificaciones componentes de que dispone la placa.

	Modelo A	Modelo B
Ethernet/Internet	Broadcom BCM2835 SoC full HD multimedia applications processor	Broadcom BCM2835 SoC full HD multimedia applications processor
Memoria	256MB SDRAM	512MB SDRAM
Chip	Broadcom BCM2835 SoC full HD multimedia applications processor	Broadcom BCM2835 SoC full HD multimedia applications processor
CPU	Dual Core VideoCore IV® Multimedia Co-Processor	Dual Core VideoCore IV® Multimedia Co-Processor
GPU		
USB 2.0	Un conector USB	Dos conectores USB
Salida de Vídeo	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
Salida de Audio	3.5mm jack, HDMI	3.5mm jack, HDMI
Almacenamiento	SD, MMC, SDIO card slot	SD, MMC, SDIO card slot
Sistema Operativo	Linux	Linux
Dimensiones	8.6cm x 5.4cm x 1.5cm	8.6cm x 5.4cm x 1.7cm
Ethernet	No	integrada 10/100 Ethernet RJ45 jack

Tabla 5: Especificaciones de Raspberry A/

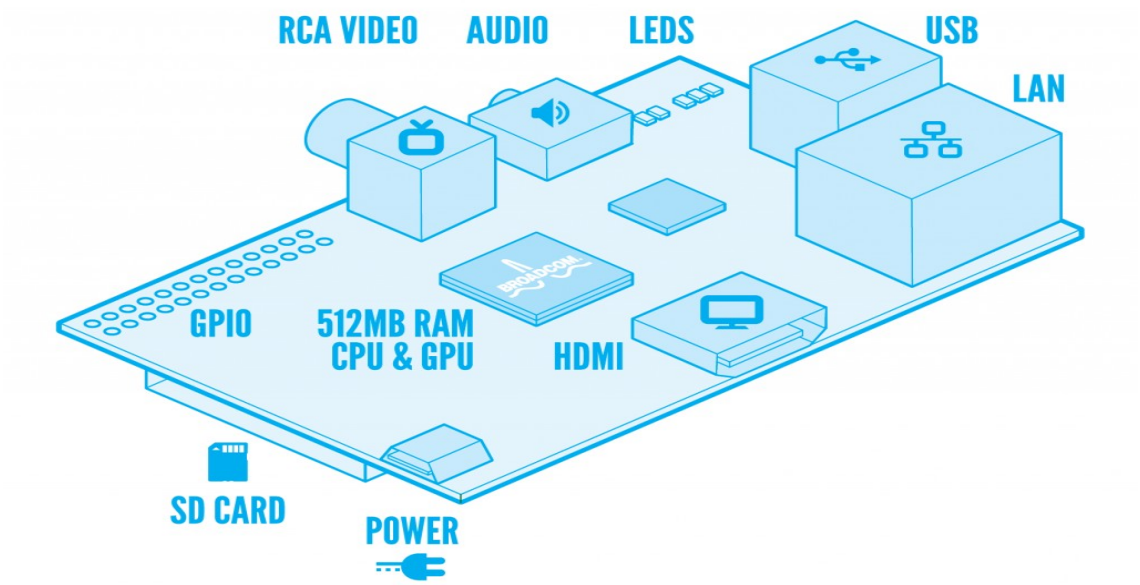


Figura 1: Raspberry modelo B

Aunque Raspberry funciona prácticamente como un ordenador, tiene algunas diferencias. Como podemos ver en la figura no disponemos de disco duro donde almacenar nuestro S.O y documentos. Para solucionar esto Raspberry dispone de una entrada para tarjetas de memoria SD, que hará las funciones de disco duro.

Esto nos ofrece la ventaja de poder disponer de diferentes Sistemas Operativos en diferentes tarjetas y utilizarlos según nos convenga.

El **SoC** (System on Chip) Broadcom funciona a una frecuencia de **700Mhz** y una **GPU** (Graphic Processing Unit) que es capaz de reproducir veo con calidad de BluRay. La **GPU** tiene un desempeño similar al de Xbox.

3.1.3 Sistemas Operativos

Disponemos de diferentes sistemas operativos que podemos instalar, todos ellos son sistemas operativos Linux para procesadores ARM.

Raspbian

Es una distribución basada en Debian Wheezy (**Debian** 7). Es un "port" no oficial para la arquitectura de procesadores ARM. Dispone de entorno gráfico **LXDE** (Lightweight X11 Desktop Environment).

LXDE es un entorno de escritorio ligero que tiene un ligero uso de los recursos para ofrecer estabilidad y usabilidad.

Pidora

Es una distribución basada en Fedora optimizada para Raspberry adaptada para funcionar en arquitectura ARM

Arch Linux ARM

Esta distribución está basada en Arch Linux adaptada a la arquitectura **ARM**. Esta distribución no está recomendada para los principiantes ya que para la instalación y configuración del sistema se requieren conocimientos altos. La filosofía de esta distribución es la simplicidad y el control absoluto para el usuario.

3.1.4 Otros

Arduino

Este es un proyecto de **hardware libre** con la finalidad de crear proyectos electrónicos. Es una placa electrónica compuesta por un microcontrolador. Los microcontroladores más usados son Atmega(168 y 328) y CortexM3.
Arduino proporciona un IDE con el que poder programar el microcontrolador.

Este dispositivo ha facilitado la creación de proyectos y ha contribuido a la construcción de un ecosistema donde cada vez más gente participa creando nuevos proyectos.

Una de las partes más interesantes de Arduino es que sea **hardware libre:** Tanto su diseño como su distribución es libre y se pueden crear proyectos sin disponer de una licencia.
Gracias a esta característica podemos encontrar diferentes tipos de placa Arduino que están basadas en la placa "base" pero con diferentes configuraciones, como procesadores, velocidades de reloj, número de pins o incluso tamaños muy reducidos.
Como ya hemos comentado Arduino proporciona su propio IDE para programar el microcontrolador, pero esta no es la única opción ya que podemos usar prácticamente cualquiera de los lenguajes más conocidos para programarlo (C, C++, C#, Python, Java …)

La combinación de Arduino y Raspberry nos ofrece una potente herramienta para la creación de proyectos muy interesantes.

Cubieboard

Este un proyecto muy parecido a Raspberry con una configuración diferente y más potente, pero a un coste más alto. Recientemente (Junio 2013) ha aparecido la 2ª versión de este proyecto el cual es capaz de correr un sistema como Ubuntu 12.04 o Android 4.2 JellyBean.

Sus especificaciones técnicas son las siguientes:

CPU	ARM® Cortex™-A7 Dual-Core
GPU	ARM® Mali400MP2, Complies with OpenGL ES 2.0/1.1
Memoria	1GB DDR3 @960M
Almacenamiento	4GB internos NAND flash, hasta 64GB en el slot de uSD, hasta 2T en disco 2.5 SATA
Alimentación	Entrada 5VDC a 2A o entrada USB
Conexiones de red	10/100 ethernet, wifi opcional
USB	2 USB 2.0 HOST, 1 USB 2.0 OTG
Otros	Una entrada para IR (infrarrojos)
Otras interfaces	96 pins, que incluyen **I2C**, **SPI**, **RGB**/**LVDS**, **VGA,** etc...

Tabla 6: Especificaciones Cubieboard

3.1.5 Elección de Raspberry frente a otros productos

La aparición de Raspberry ha dado pie a que se hayan creado otros dispositivos parecidos a este con diferentes especificaciones técnicas y más potentes. Para la realización de este proyecto las especificaciones que nos ofrece Raspberry cubren nuestras necesidades y con unos costes inferiores a los de por ejemplo Cubieboard.

Arduino no nos proporciona las capacidades que necesitamos, cosa que sí hace Raspberry por tanto más que un sustituto es un complemento para Raspberry y potenciar aún más los proyectos que se puedan crear.

3.2 Android

En nuestro proyecto Android toma una parte fundamental ya será el punto de acceso que el usuario tendrá a la aplicación que estará funcionando en la Raspberry Pi. Al usuario se le presentarán todas las posibles acciones que deberá llevar a cabo desde la pantalla de su dispositivo.

3.2.1 Introducción a Android

Android es plataforma abierta para el desarrollo de aplicaciones para dispositivos móviles. Quizás la mejor manera de explicar que es Android de forma concisa es la descripción que hizo Andy Rubin, ingeniero de Google :

> *"La primera plataforma comprensiva para dispositivos móviles. Incluye un sistema operativo, interfaz de usuario y aplicaciones – Todo el software necesario para hacer funcionar un dispositivo móvil pero sin los problemas de patentes que dificultan la innovación."*

Android forma parte de la **Open Handset Alliance** que es la unión de más de 80 empresas tecnológicas incluidas compañías de hardware, compañías de teléfono, desarrolladores de software como Samsung, Motorola, HTC, T-Mobile, Vodafone, ARM, y Qualcomm.

LA **OHA** pretende mejorar la experiencia del software para los usuarios proporcionando una plataforma para la innovación del desarrollo móvil a una mayor velocidad y calidad, sin problemas sobre licencias tanto por los desarrolladores como los desarrolladores de los dispositivos.

Cabe destacar que Android está instalado en miles de dispositivos en más de 190 países. Cada día se activan mas de 1 millón de dispositivos en todo el mundo. Esto nos ofrece un mercado potencial enorme.

3.2.2 ¿Qué es Android?

Como hemos comentado en el punto anterior, Android es una plataforma de software open source para diferentes dispositivos móviles. Pero para ser más específicos Android está compuesto de diferentes partes :

· Un kernel de Linux que proporciona una interfaz de bajo nivel para interactuar con el hardware, gestión de la memoria y control de procesos, todo optimizado para dispositivos móviles.

· Librerías open source para el desarrollo de aplicaciones, como puede ser SQLite, WebKit y OpenGL.

· **Dalvik Virtual Machine** (DVM) y el núcleo de librerías que proporcionan funcionalidad especifica para Android

· Un framework de aplicaciones que proporciona los servicios del dispositivo a la capa de aplicación incluyendo el gestor de ventanas, el gestor de geolocalización, bases de datos, telefonía y sensores.

· Software de desarrollo (SDK) que se usa para crear aplicaciones, incluyendo las herramientas necesarias, plug-ins y documentación.
En el siguiente punto desarrollaremos estos componentes.

3.2.3 Arquitectura de Android

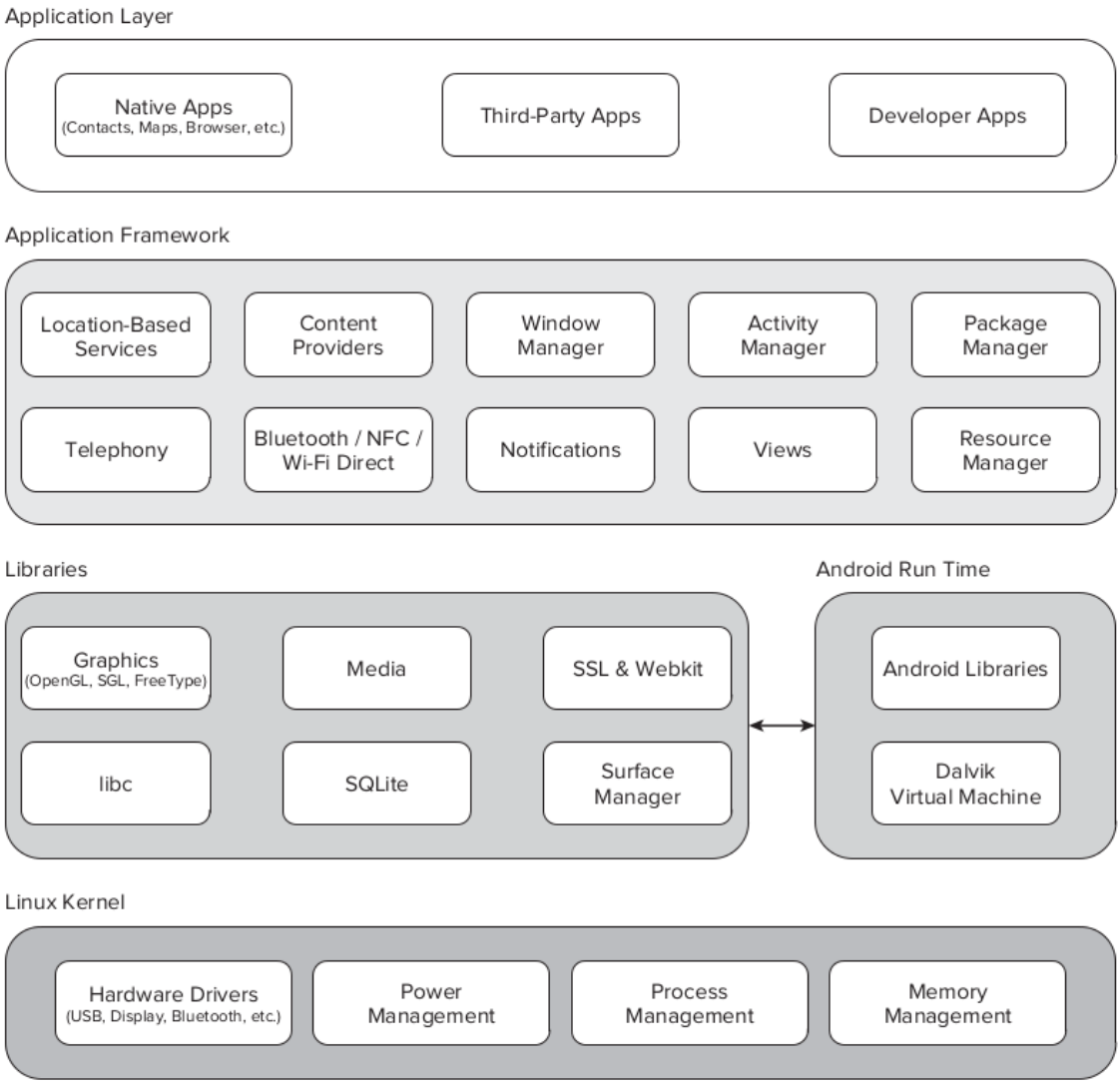


Figura 2: Arquitectura por capas de Android

Linux Kernel

Los servicios (incluyendo drivers de hardware, gestión de memoria y procesos, seguridad, redes u gestión de energía) están controlados por un kernel 3.0 de Linux (para versión JellyBean 4.3). El kernel también proporciona una capa de abstracción entre el hardware y el resto de la arquitectura.

Librerías

Están en la capa por encima del kernel incluye librearías como

· Librería para media para el audio y e vídeo
· Surface manager para proporcionar un gestor para la pantalla
· Librerías gráficas que incluyen SGL, OpenGL para gráficos en 2D y 3D
· SQLite para soporte nativo para base de datos.
· SSL y WebKit para seguridad integrada para navegadores web.

Android Run Time

Android Run Time es lo que marca la diferencia entre un teléfono Android y un teléfono con una implementación de Linux. Incluye las librerías principales y la máquina virtual Dalvik. Android Run Time es el motor que hace funcionar las aplicaciones y , aparte de las librerías, forma el núcleo básico del framework.

Android Run Time – Librerías

Aunque la mayoría de las aplicaciones desarrolladas para Android están escritas en Java, Dalvik no es una máquina virtual Java.
Las librerías del núcleo de Android proporcionan gran parte de la funcionalidad que está disponible en las librerías de Java, además de las específicas de Android.

Android Run Time – Dalvik

Dalvik es una máquina virtual que se ha optimizado para asegurar que un dispositivo puede hacer funcionar múltiples instancias eficientemente. Dalvik descansa sobre kernel de Linux para la gestión de procesos y de memoria de bajo nivel.

Application Framework

El framework de la aplicación proporciona las clases que se usan para crear aplicaciones Android. También nos proporciona una abstracción genérica para el acceso a hardware y gestiona la interfaz de usuario y los recursos de la aplicación.

Application Layer

Todas las aplicaciones, tanto las nativas como las de terceros, están construidas sobre esta capa a través de la API. La capa de aplicación funciona dentro del Android Run Time y usa las clases y los servicios disponibles del application framework.

3.2.4 Android frente a otros

Uno de los aspectos más importantes a la hora de elegir Android es la facilidad de acceso a los desarrolladores a diferencia de sus principales competidores:

Android tiene un potente SDK abierto y disponible para todos los sistemas operativos más importantes sin tener que desembolsar ni un céntimo. Dispone de una excelente documentación y una comunidad de desarrolladores enorme.

· No se necesitan certificaciones para ser un desarrollador de Android
· No existen procesos de aprobación por parte de Google para distribuir aplicaciones.
· Los desarrolladores tiene control total sobre sus marcas.

Otras compañías como Apple, limitan el acceso a desarrollar para sus dispositivos limitando los SDK's a sus máquinas y sistemas operativos, someten las aplicaciones subidas a su tienda a una aprobación totalmente subjetiva. Todo esto implica un gran desembolso, económicamente hablando, que puede ser difícil de asumir para algunos desarrolladores.

3.3 Java

Java es el lenguaje elegido para la programación de prácticamente el 99% de la aplicación.

Para Raspberry disponemos de la librería Pi4J que nos permite tener acceso total a Raspberry (configuración de pins, pulsos, lectura escritura, comunicación I2C...) creada por un equipo de desarrollo que no pertenece a Raspberry, pero muy activa.

El lenguaje de facto para el desarrollo de aplicaciones para Android es Java y dadas esta situaciones se ha elegido este lenguaje para llevar a cabo todo el proyecto.

3.3 SQLite

SQLite es una base de datos rápida, eficiente y compacta. Es autocontenida, transaccional y no necesita de servidor ni configuración.

Como en el caso de Java, SQLite es el motor de base de datos de facto de Android, y en nuestro proyecto lo usaremos en el lado del servidor (Raspberry) para guardar el histórico de temperaturas.

En Android el uso de esta base de datos esta justificado debido a las limitaciones de espacio que suelen tener los dispositivos que usan este sistema operativo.

4 análisis

Introducción

En este capítulo llevaremos a cabo un análisis de diferentes aspectos como requerimientos funcionales, materiales y el personal requerido para realizar el proyecto.

4.1 Requerimientos

Detallaremos cuales son los requerimientos funcionales y no funcionales del proyecto.

4.1.1 Funcionales

El servidor debe permanecer a la escucha

Una vez que el servidor se ha iniciado este debe permanecer a la escucha de los diferentes dispositivos que quieran interactuar con él.
Para esto el servidor creará dos **sockets** que permanecerán a la escucha.
Uno de estos **sockets** se encargará de responder al cliente para decirle que está activo.
El otro proceso será el que llevará a cabo la comunicación real entre servidor y cliente.

Disponible tanto en red local como en Internet

El tipo de red en la que está conectado el servidor ha de ser transparente al cliente. El cliente y el servidor se conectarán de manera diferente dependiendo de la red en la que este el cliente.

Detectaremos si el cliente está en una red local y en este caso enviaremos una solicitud a todos los dispositivos conectados a esa red y únicamente el servidor reconocerá esa solicitud con unos datos concretos y responderá al cliente con la ip y el puerto al que debe conectarse para hacer las peticiones que requiera.

En el caso de que el cliente no esté conectado a una red local, podrá conectarse remotamente accediendo a un IP concreta.

Notificar de cambio de IP

Cuando el cliente no está conectado a una red wifi necesita una IP a la que poder acceder. Las IP's que asignan los ISP's suelen se dinámicas. El servidor comprobará si se ha producido algún cambio de IP y en caso de que sea así notificará al cliente con la nueva IP a la que debe hacer las peticiones.

Creación de procesos en diferentes Threads

Para poder atender a las diferentes peticiones de uno o más clientes, todas las solicitudes que se lleven a cabo por un cliente deben realizarse en Threads diferentes del principal.

Las peticiones que requieran tares que se han de repetir durante un intervalo de tiempo también han de ejecutarse en procesos diferentes para no bloquear el proceso principal.

Acceso de múltiples clientes al servidor

Podrán conectarse varios clientes simultáneamente al servidore.
El acceso de clientes estará restringido a los dispositivo que estén identificados.
Se limitará el número de clientes que pueden estar conectados.

Identificación de dispositivos

Al establecer la conexión cada dispositivos deberá identificarse. De esta manera se evitarán posibles accesos no autorizados y se podrá monitorizar quién ha realizado un acción.

Consulta de temperatura actual

El cliente solicitará al servidor que le devuelva la temperatura que está obteniendo del sensor de temperatura. Estos datos se enviarán cada vez que se detecte un cambio en la temperatura.

El envío de los datos se llevará a cabo hasta que el usuario envíe la petición de detener el envío.

Encendido del sistema de calefacción

El cliente podrá encender o apagar la calefacción.

Programación de encendido

El cliente podrá programar el encendido de la calefacción con diferentes opciones.
Podrá configurar el encendido por horas, días y meses. También se podrá usar una combinación de estas como por ejemplo encender la calefacción de lunes a viernes a las 8h de la mañana hasta las 10h y de las 18h a las 23h.

Registrar temperatura

El cliente podrá programar el registro de la temperatura cada x tiempo. Estos datos se guardarán en Base de Datos con el fin de obtener estadísticas sobre las temperaturas.

Estadísticas de temperatura

Podrá solicitar al servidor datos estadísticos sobre las temperaturas registradas.

4.1.2 No funcionales

Comunicación cifrada
Al establecer la conexión entre servidor y cliente esta deberá estar cifrada, especialmente si se realiza desde fuera de una red local.

Restricción de acceso remoto
Antes de poder acceder remotamente al servidor, el dispositivo cliente se deberá registrar como cliente en servidor.

Formato de las estadísticas
Los datos obtenidos de las estadísticas se podrán mostrar en varios formatos. Ya sea simplemente mostrando los valores de resultados concretos o mostrando esos datos en forma de diferentes tipos de gráficas.

4.2 Planificación

El desarrollo del proyecto se llevará a cabo por fases. Se dividirá el proyecto en diferentes tareas y subtareas. El modelo de desarrollo que se usará será el modelo en cascada (Waterfall) donde no se puede iniciar una tarea hasta que la anterior no haya finalizado. Esto nos permite construir el proyecto de una manera más coherente con las tareas relacionadas.

Así construimos de forma ascendente las partes del proyecto, desde los elementos que serán comunes y que proporcionan funcionalidades a las “capas” superiores.

Se facilita el debug de la aplicación y se proporciona una construcción más sólida para las demás partes del proyecto.

4.2.1 Tareas

4.2.1.1 Servidor

Servicio de escucha de peticiones

El servidor ha estar a la escucha de las diferentes peticiones que lleguen del cliente.
Se crearán las clases que manejan las peticiones. Estas clases manejarán la creación de los sockets, puertos, etc.. que se necesitarán para conectar con el cliente.

· Se creará la parte que gestionará la recepción de paquetes de descubrimiento del servicio.

· Manejo de las conexiones con los diferentes clientes que se conecten al servidor.

· La comprobación de los cambios de IP del servidor para posteriores notificaciones a los clientes del cambio.

Gestión de las peticiones

Creación de las clase/s necesaria/s que gestionan la petición que hace el cliente.
El cliente enviará una petición con los datos que quiere obtener y esta petición estará compuesta por un estructura concreta, que identificará la clase y el método que se debe ejecutar.
Se ocupará de la creación de las clases correspondientes y la llamada al método requerido para completar la petición del usuario.

Controlador y Modelo

Se usará la implementación del patrón MVC de una forma particular ya que en este proyecto la parte de la vista la representa el socket.

El controlador realizará las funciones de gestión entre el Modelo y la Vista.
El modelo será el encargado de ejecutar toda la lógica de negocio.
Se creará el controlador y el modelo de Temperatura y todas la acciones que se podrán ejecutar.

Gestión del termostato y la calefacción

Crearemos los objetos que se encargarán de abstraer el funcionamiento de la calefacción y el termostato. Ejecutará las peticiones que reciba del modelo.
Se encargará de interactuar con los GPIO de Raspberry pi obtener los datos del sensor de temperatura y el encendido y apagado de la calefacción. Gestionará los estados de los diferentes componentes y almacenará la información de las diferentes interacciones.

4.2.1.2 Android

Descubrimiento del servidor

Se creará las objetos necesarios para tratar el descubrimiento del servidor y la conexión con este.

Conexión con el servidor

Se creará una clase que abstraiga la conexión con el servidor y se ocupará del envío y la recepción de los datos provenientes de este.
Otra clase se encargará de encapsular y gestionar las peticiones al servidor.

Estructura gráfica

Se crearán las pantallas necesarias para facilitar al usuario la interacción con el servidor y la obtención de estos datos y mostrarlos en pantalla en caso de que sea necesario.
Se añadirán los elementos gráficos necesarios para permitir la interacción con el usuario y el envío de datos al servidor.

Peticiones al servidor

Se implementará el código para que los componentes gráficos hagan las peticiones al servidor cuando el usuario interaccione con la aplicación. Se gestionará la recepción de datos devueltos y se implementará el código necesario para mostrar esta información en los componentes gráficos adecuados.

Notificaciones

Se creará la gestión de las notificaciones que enviará el servidor a través de Google para informarnos de cambios en la IP proporcionada por el ISP. LA gestión de la nueva IP a la que conectar será transparente al usuario que sólo será informado de que ha habido u cambio de IP.

Diseño de la aplicación

Partiendo de los componentes visuales básicos creados para el funcionamiento básico de la aplicación, se creará un diseño con la finalidad de que sea funcional para el usuario, fácil de entender y de usar.
Además del diseño se buscará usar la mejor forma para la navegación entre las diferentes pantallas que componen la aplicación.

4.3 Recursos

Tal y como describimos en el apartado 2.5.3,el personal que creemos sería el adecuado estaría compuesto por un Analista, un Ingeniero y un Tester.
Debido a la naturaleza del proyecto todos estos roles están gestionados por una sola persona.

Detallaremos cuales serían las funciones de los diferentes recursos y que aportarían.

Analista

El analista es la persona encargada de estudiar el dominio del software y prepara los requerimientos y la especificación. En este proyecto se encargará de diseñar la estructura de la aplicación que correrá en el servidor y de la aplicación de Android.

Deberá tener conocimientos sobre el funcionamiento de Raspberry Pi y sus características para adaptar el software a los recursos de que de dispone el dispositivo.

También tendrá conocimientos de como funciona Android y sus características como versiones disponibles, porcentaje de usos de cada versión para poder llegar a más dispositivos y los problemas inherentes a crear apliaciones que funcionen en la mayoría de dispositivos.

Las horas que se asignan a las tareas quedan repartidas de la siguiente manera

Tareas	Documentación	Análisis requerimientos	Diseño software	Total
Horas	24	16	16	**56**

Tabla 7: Asignación horas/tarea de Analista

Y el coste será

Coste hora (€)	Horas	Total
50	56	**2800 €**

Tabla 8: Coste de las horas asignadas al Analista

Ingeniero de software

El ingeniero de software se encarga de aplicar los principios de la ingeniería al diseño, desarrollo, mantenimiento, pruebas de funcionamiento y evaluación del software y los sistemas que funcionan en los ordenadores.

En este proyecto el Ingeniero de software trabajará junto al Analista en el diseño de la aplicación. También se encargará de la programación de la aplicación aplicando los conocimientos concretos del lenguaje que se usará y aplicando los principios de Ingeniería del Software. También especificará los tests que que deberá usar el Tester.

Las horas que se asignan a las tareas quedan repartidas de la siguiente manera

Tareas	Análisis requerimientos	Diseño software	Programación	Total
Horas	8	8	175	**191**

Tabla 9: Asignación horas/tarea del Ingeniero de Software

Y el coste será

Coste hora (€)	Horas	Total
35	191	**6685 €**

Tabla 10: Coste de las horas asignadas al Ingeniero de Software

Tester

Una vez se haya completado la aplicación llega el momento de comprobar que el producto cumple con todas las especificaciones y requisitos y que funciona correctamente. Para esta tarea asignaremos un Tester que comprobará el cumplimiento de lo especificado haciendo funcionar la batería de tests. Todos los posibles errores serán reportados al Ingeniero de software que los corregirá y volverán a ser sometidos a las pruebas.

Las horas que se asignan a las tareas quedan repartidas de la siguiente manera

Tareas	Pruebas	Total
Horas	40	**40**

Tabla 11: Asignación horas/tarea del Tester

Y el coste será

Coste hora (€)	Horas	Total
25	40	**1000 €**

Tabla 12: Coste de las horas asignadas al Tester

4.4 Material

Detallaremos todo el material (tanto físico, como de software) necesario para el desarrollo del proyecto.

Raspberry Pi, Modelo B	Coste: 33€

Sensor Temperatura	Coste: 5,43€

Teléfono móvill Android	Coste: 199€

PC	Unidades:2	Coste: 750€/u
Intel core i7 4GB Memoria Disco duro de 1 TB		

Monitor	Unidades: 4	Coste: 200€/u
Monitores de 23”		

Eclipse IDE 3.6.2	Coste: 0€
Eclipse es el IDE con el que programaremos todo el proyecto, aprovechando la integración que ofrece el SDK de Android.	

Java	Coste: 0€
Lenguaje con el que desarrollaremos el proyecto.	

SQLite	Coste: 0€
Base de Datos	

Android Developer Tools	Coste: 0€
Plugin para Eclipse para el desarrollo de aplicaciones Android	

Ubuntu 12.04	Coste: 0€
Sistema Operativo que instalaremos en los Pc's	

PHP 5.4	Coste: 0€
Lenguaje de programación	

GIT	Coste: 0€
Software de control de versiones	

BitBucket	Coste: 0€
Servicio de alojamiento web para sistemas de control de versión	

Coste total del material	**2537,43**

5 implementación

Introducción

En este capítulo describiremos de forma técnica como está diseñado el proyecto y la interacción entre los diferentes componentes del sistema.

Dividiremos el el capítulo en tres puntos principales: empezaremos explicando de forma muy general cómo es el proceso de descubrimiento del servidor por parte del cliente y seguidamente continuaremos con la descripción de la parte del servidor (Raspberry) y la parte cliente (Android).

5.1 Inicio

Cuando conectamos el servidor para que empiece a funcionar, este se queda a la escucha de las peticiones del cliente. Pero surge el problema de que el cliente desconoce que ip tiene el servidor y por tanto no puede comunicarse con él.

Por tanto necesitamos un paso previo, antes de empezar a enviar peticiones al servidor, y es el de saber quién es el servidor.

5.1.1 En una red local

Si nos encontramos en una red local el proceso de descubrimiento se desarrolla de la siguiente manera:

1.- El servidor está funcionando y está a la escucha de de solicitudes.

2.- El cliente inicia la aplicación y ve que no tiene ninguna conexión con el servidor, ni tiene un dirección de una conexión anterior. Por tanto envía un paquete UDP a la dirección broadcast de la red en la que está con un contenido especial. Se queda a la escucha del servidor.

3.- Todos los ordenadores de la red reciben este paquete, pero sólo el servidor reconoce ese contenido como un paquete para descubrir el servidor de control.

4.- El servidor envía un paquete al cliente, que contiene la ip y el puerto por el que está escuchando para peticiones.

5.- El cliente ahora dispone de la dirección del servidor y establece la comunicación.

Ahora el cliente puede hacer todas las solicitudes al servidor que considere.

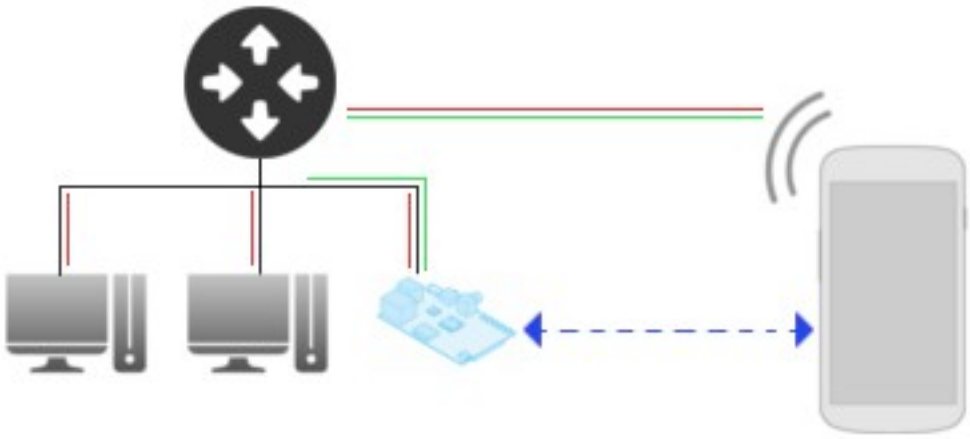


Figura 3: Proceso de conexión en una red local

5.1.2 En Internet

En el caso de que estemos fuera del alcance de una red local, la forma de conectar con el servidor es diferente y además nos encontramos de que la mayoría de las conexiones a Internet que nos proporcionan los ISP's son con IP's dinámicas y esto nos impide poder hacer una conexión directa con nuestro servidor.
Para solventar esto usaremos los servicios de notificación que nos proporciona Google, Google **Cloud Messaging** (GCM).

Gcm es un servicio que nos permite enviar información desde nuestro servidor a un dispositivo Android concreto.

El proceso es el siguiente:

1.- Al iniciar el servidor, se crea un proceso que se ocupa únicamente de comprobar cada X minutos si la IP que nos ha asignado el servidor ha cambiado.

2.- Si se detecta que la IP ha cambiado, se envía una notificación a los servidores de GCM con la nueva IP del servidor.

3.- El dispositivo recibe una notificación en su dispositivo informándole que la IP de su servidor ha cambiado. Automáticamente la aplicación Android se encargará de usar esa dirección para cualquier petición.

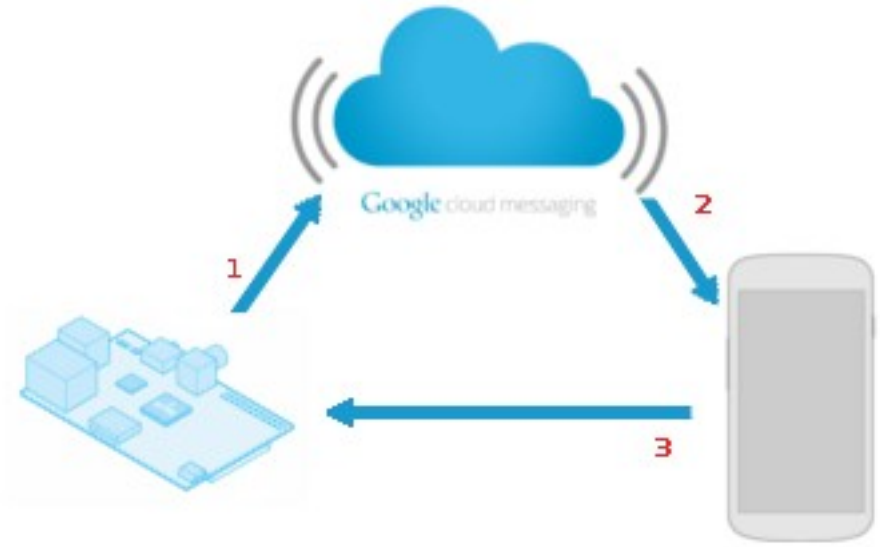


Figura 4: Proceso de comunicación por GCM

Una vez establecida el cliente ha descubierto la al servidor, ya se pueden realizar las peticiones que estén disponibles para el usuario.

5.2 Servidor

5.2.1 Diagrama de clases básico del servidor

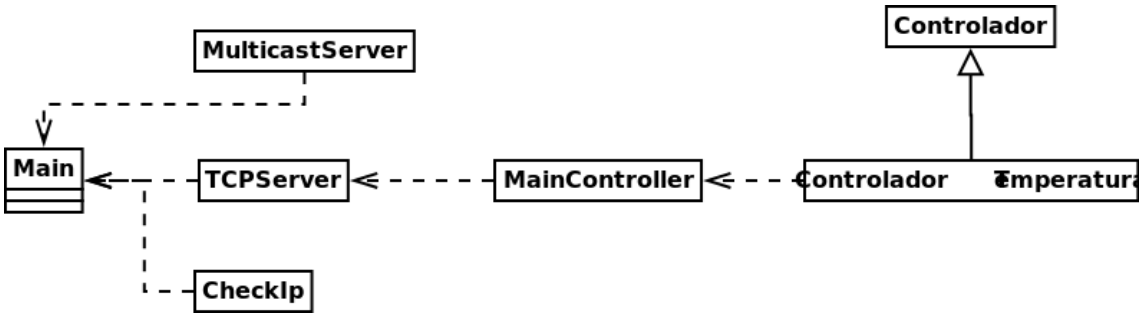


Diagrama 1: UML de clases básico del servidor

Este diagrama es una visión global de las principales clases que forman el proyecto. En los siguientes puntos entraremos en detalle y mostraremos las otras clases que forman parte del programa.

5.2.2 Funcionamiento del Servidor

El servidor corre la aplicación principal de todo el sistema. Es el encargado de el control de los diferentes dispositivos que se conecten a él, que en el caso de este proyecto será el sensor de temperatura y el encendido y apagado del sistema de calefacción.

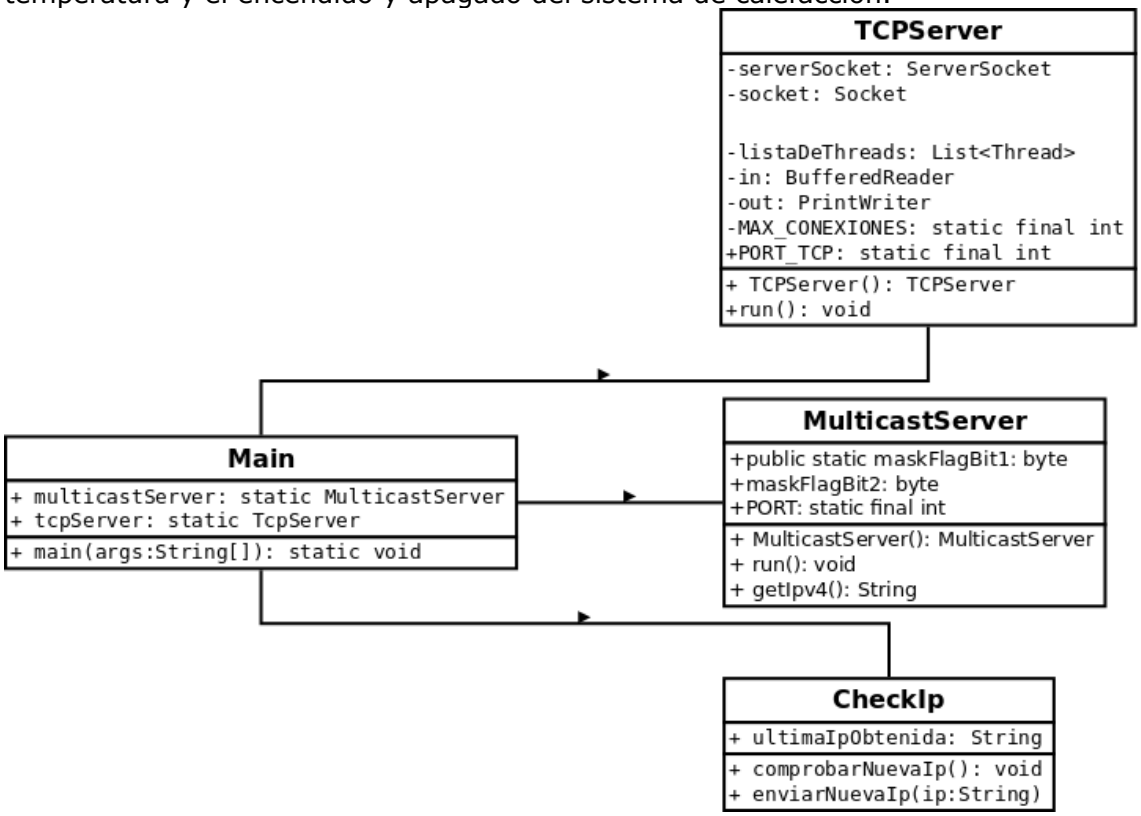


Diagrama 2: UML completo de clases del servidor

La clase **Main** es el punto de entrada a la aplicación. Al iniciarse el servidor, crea tres hilos: **MulticastServer** y **TCPServer** se crea los sockets que estarán a la espera de recibir solicitudes y **CheckIp** que comprobará los cambios de ip.

Cuando la aplicación se inicia esta desconoce que IP le ha asignado el router por eso uno de los los primeros hilos que crea al iniciar la aplicación es el de la clase **MulticastServer**.

MulticastServer

MutlicastServer se queda a la escucha de recibir un paquete enviado por broadcast, con unos datos concretos. Esta acción se llama "descubrimiento" y la envía el cliente desde el dispositivo Android para encontrar el servidor al que asociarse.

Entonces se le devuelve al cliente la dirección IP y el puerto del proceso que llevará a cabo todas las peticiones solicitadas por el cliente. Este proceso está implementado en la clase **TCPServer**.

TCPServer

TCPServer es el punto de entrada para la gestión de las peticiones del usuario y únicamente creará instancias de la clase **MainController** por cada dispositivo nuevo que se conecte.

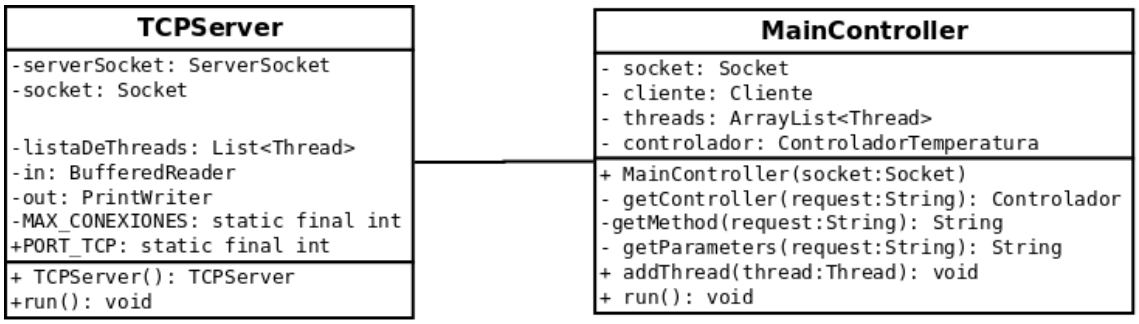


Diagrama 3: UML de clases TcpServer y MainCotroller

La lógica de la aplicación empieza en la clase **MainController**. Esta implementación sigue el patrón de diseño MVC (Modelo Vista Controlador) de una manera "sui generis" para el funcionamiento de esta proyecto.

MainController se encargará de interpretar la petición que llega desde el cliente.
Las peticiones están especificadas con el formato siguiente:

"**Nombre del Controlador:Nombre del método[:[parametros]]**".

Entonces **MainController** creará un objeto de la clase y llamará al método especificados y en caso de que sea necesario pasará los parámetros.

MainController quedará a la espera de que el cliente envíe una petición o directamente cierre la conexión.

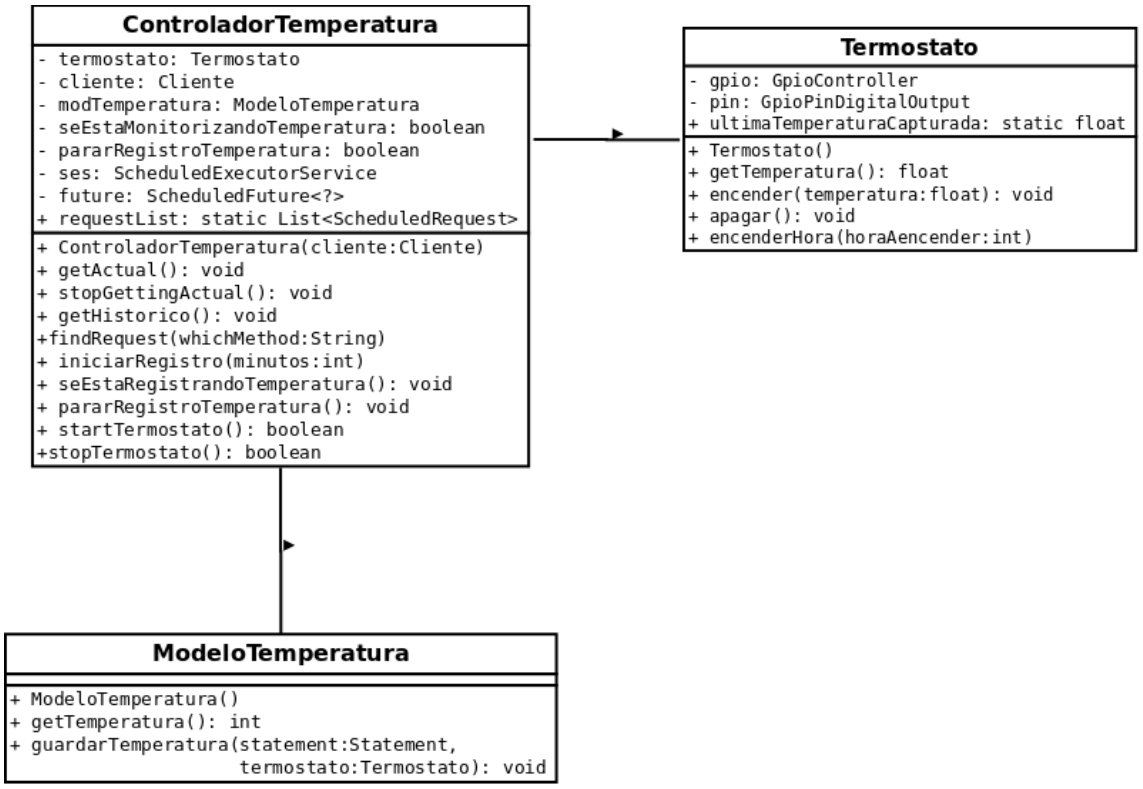


Diagrama 4: UML Controlador y Modelo y Termostato

ControladorTemperatura implementa el controlador en el patrón **MVC**. Se encarga de gestionar las peticiones del cliente haciendo las peticiones al Modelo que es el que hace la lógica de negocio del proyecto. Una vez el modelo le ha devuelto al controlador el resultado, lo pasa a la vista, que en este caso la vista lo implementa el **socket** que envía los datos al cliente. En este caso el **ControladorTermperatura** se encarga de la gestión de peticiones relacionadas con el sensor de temperatura y el estado de la calefacción.

La clase Termostato representa al estado del sensor de temperatura y el estado de la calefacción. Implemente el patrón **Singleton** ya que no debe existir ningún otro objeto que represente el estado actual de estos componentes. Se encarga de implementar las librerías Java que gestionan el control de los componentes de Raspberry Pi. Almacena la última temperatura registrada.

La clase **Cliente** representa al dispositivo móvil que se conecta con el servidor. Almacena el **socket** que conecta con el dispositivo. Se encarga de recibir y de enviar los datos y controla el estado de la conexión.

Diagrama 5: UML de la clase SQLiteConexion

SqliteConexion es la clase encargada de realizar la conexión con la base de datos sqlite. Busca el fichero que será la base de datos y la abre. En caso de que no exista la base de datos la creará.

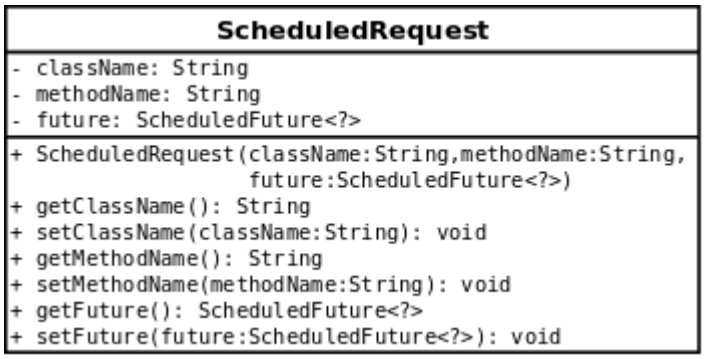


Diagrama 6: UML de la clase ScheduledRequest

ScheduledRequest es una de las clases más importantes de la implementación del servidor.

Algunas de las peticiones que se hacen al servidor, como puede ser el obtener la temperatura actual o el registrar la temperatura, necesitan que se ejecuten cada X tiempo. Estas tareas se llevan a cabo usando **ScheduledExecutorService** que donde se ejecutan Threads con el código necesario para llevar a cabo las peticiones.

Todos estos procesos los guardamos en un array de objetos **ScheduledRequest** donde se asocia el proceso que se ejecutará con el nombre del método y la clase que la ha creado. De esta manera cuando el usuario solicite la detención de estos servicios sólo deberemos buscar el objeto **ScheduledRequest** que haya realizado la llamada y cancelar la ejecución de ese proceso.

5.3 Android

En este apartado describiremos el diagrama de clases y componentes de la aplicación y además contaremos con el apartado gráfico que es uno de los aspectos fundamentales de la aplicación.

Todas las acciones que el usuario puede hacer conllevan la interacción con la interfaz gráfica del usuario y estas peticiones actualizan esa interfaz gráfica para mostrar los datos devueltos por el sevidor.

Empezaremos a describiendo el funcionamiento del framework que Android nos ofrece y la parte más directa con el usuario que son las **Activities**.

5.3.1 Activities

Las activities son la capa de presentación de las aplicaciones. Las interfaces gráficas de las aplicaciones se construyen extendiendo de la clase Activity. Las Activities usan Fragments y Vistas para mostrar la información y la disposición en que ésta se mostrará y para responder a las acciones del usuario.

Uno de los aspectos más importantes de las aplicaciones es el ciclo de vida de las Activities.

Las aplicaciones de Android no controlan sus proppios ciclos de vida sino que se ocupa de eso el RunTime, y lo hace por cada aplicaciones que está funcionando y por extensión de cada Activity dentro de cada proceso.
Aunque el Run Time se ocupa del manejo de las Activities y de cuando debe termina su proceso, el estado de una Activity ayuda a determinar cual es la prioridad de una Aplicación.
Las Activities se crean y se destruyen o son sacadas de la pila de Activities de la aplicaicón. Mientras se realizan esas transiciones, podemos encontrar las Activities en cuatro estados diferentes:

Activa

Cuando una activity esta arriba de la pila de activities significa que está en primer plano, que es visible y que el usuario está interaccionado con ella. Android intentará mantener funcionando la Activity a cualquier precio matando otras Activities que esten por debajo si es necesario para asegurarse que la Activity principal tiene todos los recursos que necesite. Esta activity pasará estar pausada cuando otra Activity sea la activa.

Pausada

En algunos casos la Activity podrá estar visible, pero no tendrá el foco. En esta situación la Activity estará pausada. Se llega a este estado si una Activity transparente o que no sea a pantalla completa pasa a estar activa delante de otra, como puede ser el caso cuando se muestra un Dialog con un mensaje informativo o esperando algúna confirmación por parte del usuario.

Cuando una Activity está pausada se le trata como si estuviese activa aunque no tenga interacción con el usuario.

En algún caso extremo se puede matar una Activity que está pausada en caso de que se necesiten recursos para la que sí está activa.

Cuando una Activity pasa a ser totatalmente invisble se para.

Parada

La activity permanece en memoria y guarda toda la información referente a los estados. En este caso, si el sistema necesita memoria esta activity es una candidadta a ser eliminada para obtener esos recursos.

Cuando una activity está en este estado es importante guardar todos los datos y el estado de la interfaz de usuario y detener cualquier operación que no sea crítica. Una vez que se ha salido de la activity o se ha cerrado, pasa a estar inactiva.

Inactiva

Antes de matar una activity y antes de que se muestre, su estado es inactivo.
Las activities que están en este estado, se han eliminado de la pila y se tienen que volver a iniciar antes de que se puedan mostrar al usuario.

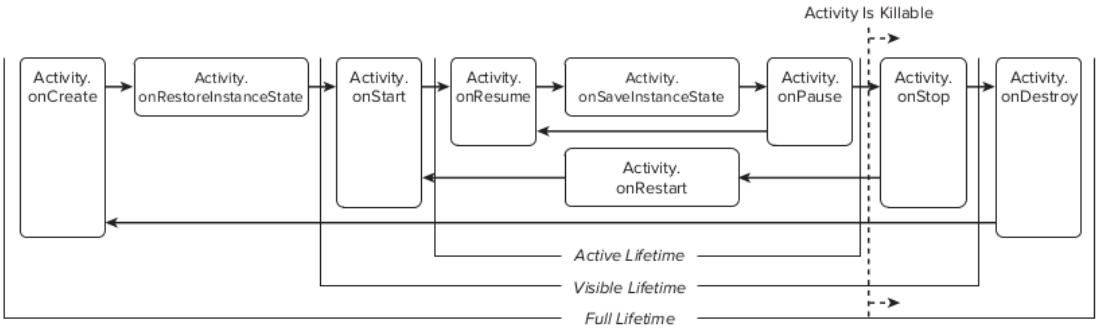


Figura 5: Ciclo de vida de las activities

Entre las transiciones de los estados comentados anteriormente, el sistema llama a un conjunto de métodos que se llaman para asegurar que la activity reacciona a los diferentes cambios de estado.

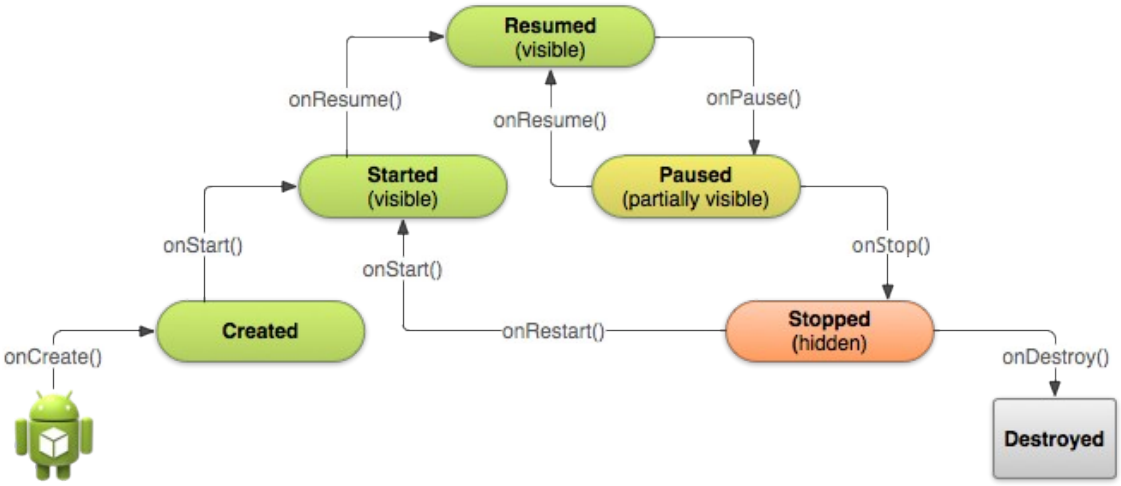


Figura 6: Ciclo de vida de las activities y callbacks

5.3.2 Diagramas de clase básico

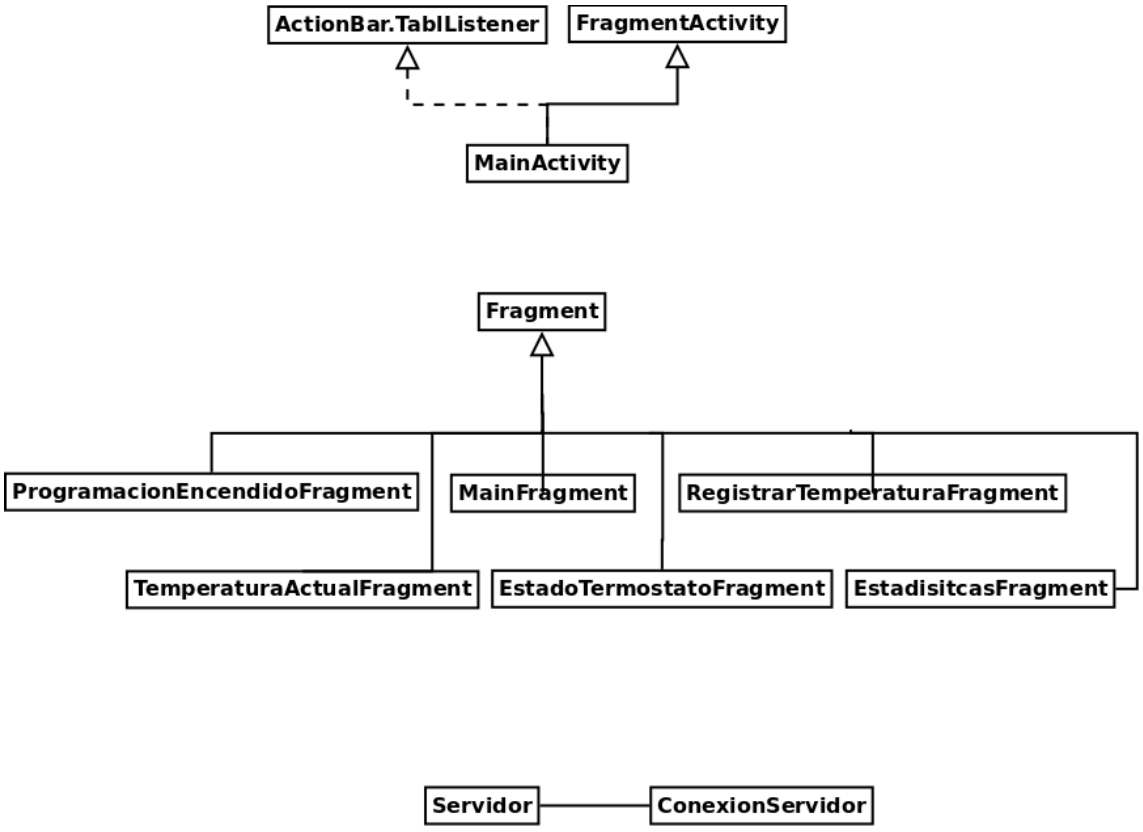


Diagrama 7: UML de clases básico Android

Como podemos ver en el diagrama, tenemos unas clases que extienden de la clase **Fragment**. Un **fragment** es forma parte del framework de Android y representa el comportamiento de una parte independiente de una **activity**. Los f**ragments** nos proporcionan flexibilidad a la hora de componer las interfaces gráficas y también aporta al poder reusarlo en otras partes de la aplicación. En una **activity** pueden convivir diferentes **fragments** y cada uno de ellos tiene su propia ciclo de vida aunque ésta depende principalmente del ciclo de vida de la **activity**. Cuando la la **activity** se pausa los **fragments** también, si se destruye los **fragments** también.
Un **fragment** siempre ha de estar dentro (pertenecer) a una **activity** y en nuestro proyecto cada **fragment** pasará a formar parte de **MainActivity** según el usuario lo requiera.

Cada fragment tendrá asociado un layout con los widgets necesarios para que el usuario puede interactuar con la aplicación y poder mostrar la información en pantalla.

Antes de empezar a describir que tarea lleva a cabo cada fragment, debemos detenerlos antes en explicar una de las clases que usaremos para proceder a hacer todas las peticiones al servidor, la clase **AsyncTask**.

Android se preocupa mucho por que la experiencia de usuario sea fluida. A veces las aplicaciones se pueden colgar, quedarse "congeladas" durante unos instantes o tarda mucho el llevar a cabo un proceso. Estas situaciones se conocen como **ANR** (Application Not Responding) y se lanzan después de que la IU este bloqueada durante 5 segundos.

Cuando Android detecta una de estas situaciones y la aplicaión deja de responder, se le muestra al usuario un Dialog preguntándole si quiere esperar a que la aplicación vuelva a funcionar o quiere cerrarla.

Las aplicaciones que hacen uso de peticiones de internet (como es nuestro caso) son susceptibles de encontrarse en la situación de **ANR**.

Para evitar esto Android nos propone sacar fuera del proceso principal de la interfaz de usuario todas estas operaciones que pueden bloquear la aplicación, y ejecutarlos en Threads diferentes.

El problema que aparece si usamos Threads aparte es que perdemos el contexto de la aplicación y en el caso de que tengamos que actualizar o mostrar información por pantalla al usuario, la aplicación se complicará.

Pero Android nos proporciona la clase AsyncTask que nos soluciona estos dos problemas. AsyncTask nos permite ejecutar estas aplicaciones que pueden ser bloqueantes en background, sin bloquear el hilo principal de la aplicación que es el de la UI. Además nos permite actualizar la UI en caso de que lo necesitemos.

En el método ***doInBackground*** es donde se lleva a cabo el código que puede bloquear la UI y llamando al método ***publishProgress*** desde dentro de ***doInbackground*** se ejecuta el método ***onProgressUpdate*** y desde ahí podemos actualizar la IU.

En nuestra aplicación cada **fragment** que hace peticiones al servidor, implementa estas acciones extendiendo de una clase **AsyncTask** creadas como clases privadas.

Diagrama 8: UML de EstadisticasFragment

Se encarga de hacer las peticiones sobre los datos de temperaturas recogidos por el servidor y los muestra en pantalla. El método ***onCreateView*** carga el layout asociado.

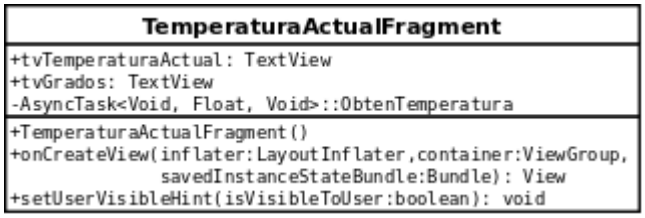


Diagrama 9: UML de TemperaturaActualFragment

Solicita al servidor cual es la temperatura actual que está ogbteniendo el sensor de temperatura. El socket permanece a la escucha para recibir todos los cambios que se detectan y lo muestra por pantalla. El método ***onCreateView*** carga el layout asociado.

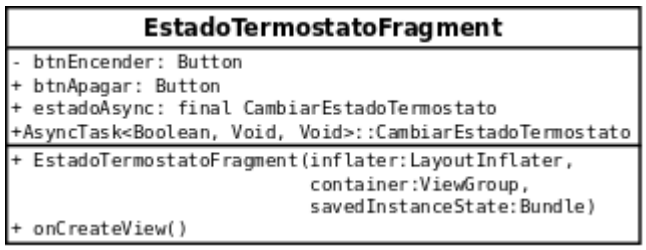


Diagrama 10: UML de EstadoTermostatoFragment

Hace las peticiones al servidor para encender o apagar la calefacción. Comprueba antes cual es el estado en que se encuentra. El método ***onCreateView*** carga el layout asocidado.

5.3.3 Interfaz de usuario

En Android las interfaces gráficas se crean un ficheros independientes del código. Estos ficheros son los Layouts y definen la estructura visual de la interfaz de usuario. Los layouts y los componentes que forman parte de la UI están escritos en XML.

En cada **activity** o **fragment** se cargan estos ficheros que definen la UI y de esta manera queda asociado la activity o fragment a el layout concreto. Una vez están asociados podremos acceder a los componentes del layout desde el código y podremos realizar las modificaciones que consideremos.

Disponemos de diferentes maneras de acceder a las actividades o fragments, ya sea de en forma de Tabs o de la forma más general que es accediendo desde los botones que nos lleven a las diferentes pantallas.

En nuestro proyecto hemos implementado una de las últimas formas de navegación y más eficientes entre pantallas llamada **Swipe Views** y lo combinaremos con **Tabs**.

Las vistas Swipe proporcionan una navegación lateral entre pantallas, así el usuario sólo ha de desplazar la pantalla hacia uno de los laterales para acceder a otra pantalla.

Figura 7: Navegación entre las diferentes activities

Pantalla Principal

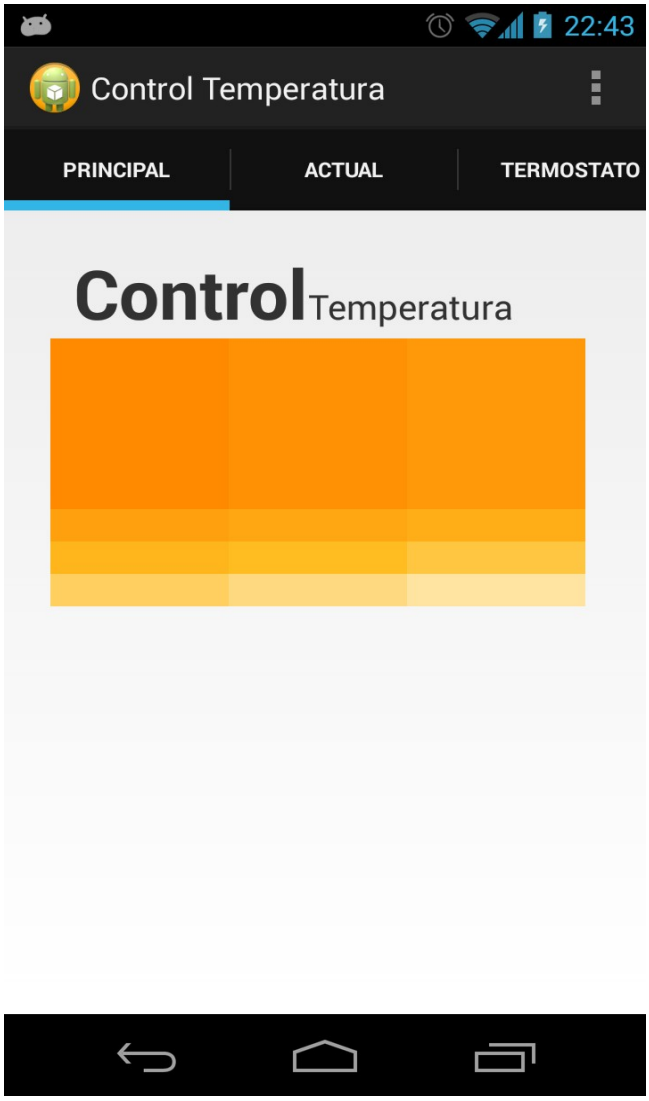


Figura 8: Activity principal

Esta es la primera pantalla que aparecerá al abrir la aplicación de Android.

La primera activity que se carga es el MainActivity que será el contenedor de todos los fragments que forman parte de la aplicación.

El primer fragment que se carga es MainFragment.

Este únicamente se encargará de mostrar el layout. No habrá ningún tipo de interacción con el los componentes.

Temperatura actual

Figura 9: Activity Actual

En esta pantalla se le mostrará al usuario la temperatura que esta registrando el sensor de temperatura.

El fragment que carga esta vista es **TemperaturaActualFragment.**

Cuando navegamos entre pantallas y llegamos a esta, cuando la vista pasa a ser visible el fragment se encarga de llamar al AsyncTask que hará la petición al servidor.

El socket se quedará leyendo lo que recibe del servidor hasta que el usuario navega hacia otra pantalla. Entonce el fragment volverá a llamar a otro AsyncTask para decirle al servidor que cancele el envío de los cambios de temperatura.

Este proceso se repetirá cada vez que esta pantalla esté activa.

Se activa el Tab Actual para mostrar en la pantalla que estamos.

Termostato

Figura 10: Activity Termostato

La pantalla de Termostato nos da las opciones de enceder o apagar la calefacción y de registrar la temperatura cada x tiempo.

Para el encendido de la calefacción deberemos proporcionar los grados a los que queremos que se matenga encendida.

Cuando navega hasta esta pantalla se pregunta al servidor cual es el estado del termostato.

El encendido y apagado tiene su propio asynctask que hará las peticiones al servidor cuando el **switch** de encender y apagar cambie de estado.

En el registro de temperatura el usuario podrá especificar cada cuando quiere registrar la temperatura actual. Introducirá el número y seleccionará en el spinner si son minutos o segundos.

Si ya hemos iniciado un proceso de registro en servidor, el botón de "Iniciar monitorización" pasará a mostrar el texto "Detener monitorización".

El registro de temperatura también tiene su propio asynctask para hacer las peticiones al servidor, que se ejecutará cada vez que el usuario presione el botón.

El fragment que se carga es **EstadoTermostatoFragment**.

Estadísticas

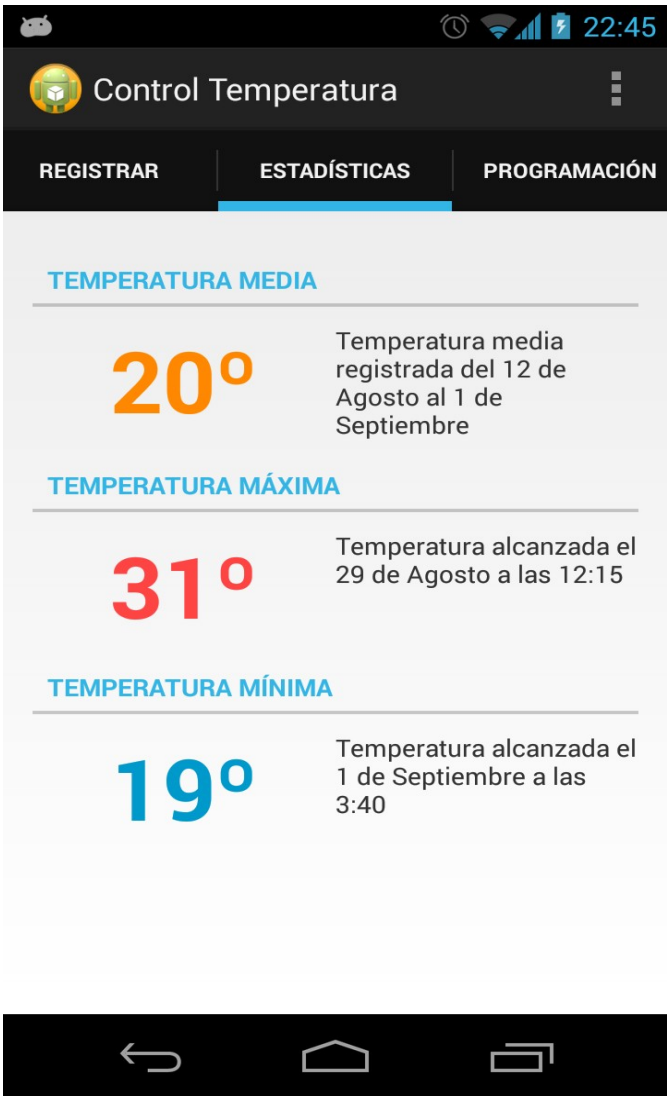


Figura 11: Activity Estadísticas

Como su nombre bien indica, en esta pantalla se muestran los datos que se han recogido al activar el registro de temperaturas de la pantalla anterior. Nos devuelve 3 resultados con las temperaturas media, máxima y mínima de todos los datos de que se dispone en el servidor, además del momento en que se registro esa temperatura.

El asyncTask hace la petición al servidor y se queda esperando hasta la recepción de los datos.

La actualizaciones son constantes si hemos activado el registro y por tanto cada vez que se entre en esta pantalla se solicitarán los datos nuevamente.

El fragment que se carga es EstadisticasFragment.

Programación

Figura 12: Activity Programación

En la pantalla de programación es donde el usuario puede programar el encendido de la calefacción.

La elección es múltiple ya que puede seleccionar una combinación de días y meses, sólo días o sólo meses.

También especificará de que hora a que hora estará encendidas proporcionando un rango de horas.

El asyncTask enviará la selección que haya hecho el usuario y la guardará y ejecutará el encendido según lo especificado.

Para desactivar cualquier programación previa, se desactivarán todas las opciones y el botón de "Programar" pasará a ser "Desactivar".

Se puede sobrescribir una programación anterior simplemente creando una nueva.

5.3.4 Notificaciones

Como comentamos en el punto 5.1, cuando estamos fuera del alcance de una red local necesitamos saber cual es la Ip que nos ha asignado el ISP para poder conectar con nuestro servidor, pero esta IP cambia dinámicamente y necesitamos saber cuando nuestra IP ha cambiado y cual es la nueva dirección.

Para informarnos de este cambio el servidor nos envía notificaciones al al teléfono móvil con la nueva ip.

Figura 13: Notificación cambio de IP

Para gestionar la recepción de las notificaciones y el manejo de estas y del contenido que nos proporcionan usaremos las clases **GCMBroadcastReceiver**, **GCMIntentService** y **GCMRegister**.

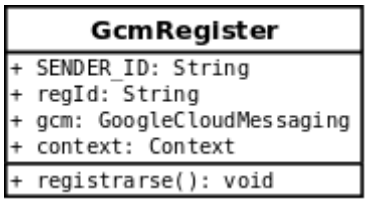


Diagrama 11: UML de GcmRegister

Antes de poder recibir notificaciones, necesitamos que nuestro dispositivo se registre en los servidores de GCM. Cuando un dispositivo se registra obtiene un ID que identificará al dispositivo al que enviar las notificaciones. De estos pasos se encarga la clase **GCMRegister**.

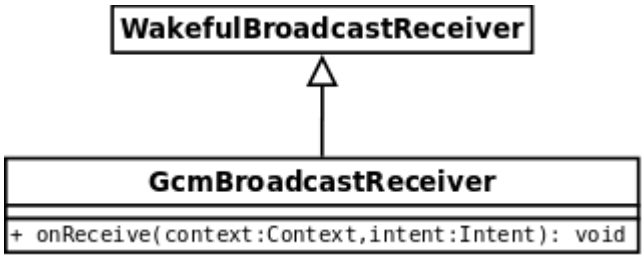


Diagrama 12: UML de GcmBroadcastReceiver

GCMBroadcastReceiver lleva a cabo la recepción de los mensajes de GCM. Como podemos ver hereda de la clase **WakefulBroadcastReceiver** que es un tipo especial de receptores de broadcast que se asegura de la CPU está activa aunque la pantalla no esté activa para que el proceso de pasar los mensajes a la clase que se encargará de tratar la información contenida en la notificación no se queda sin concluir.

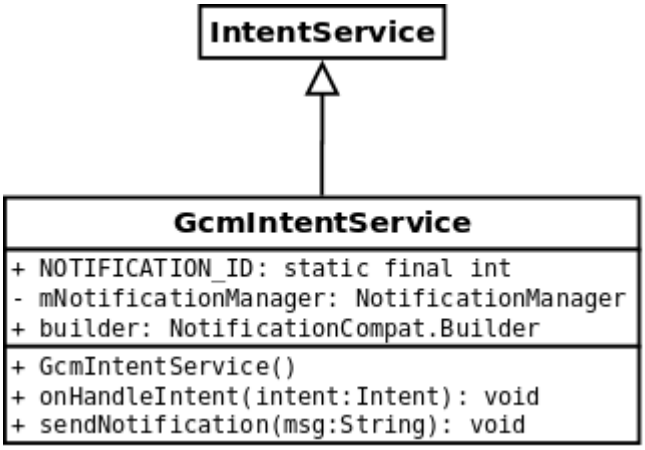


Diagrama 13: UML de GcmIntentService

GCMIntentService es la clase que se encarga de gestionar el mensaje recibido. Obtiene los datos de la nueva IP para usarlos par la aplicación y muestra la notificación que verá el usuario en la prte superior de su teléfono donde podrá ver el icono de la aplicación y un pequeño texto informándole que se ha obtenido la nueva ip del servidor.

5.4 SQLite

Cuando el usuario decide registrar las temperaturas durante unos intervalos de tiempo, esta información se guarda en una base de datos llamada SQLite. Esta base de datos nos ofrece toas lo que necesitamos para almacenar la información relacionada con la temperatura y las fechas.

Esta base de datos no precisa de configuración, tiene un tamaño realmente pequeño y es igual de eficaz y confiable que otras bases de datos de más renombre.

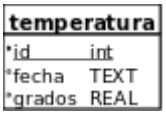


Diagrama 14: UML de Base de Datos

Únicamente usaremos una tabla para almacenar estos datos. Podremos realizar desde código

varios tipos de de estadísticas y filtrar por rangos de fecha y hora.

5.5 Pi4J

Esta es la librería para Java que usará el servidor para realizar todas las comunicaciones con los pins GPIO de Raspberry.

Pi4J es un proyecto creado por una comunidad de usuarios que proporcionan un puente entre las librerías nativas y el lenguaje Java para proporcionan un acceso completo a las pins de raspberry.

Aunque es proyecto aún se encuentra en fase de desarrollo es muy estable.

Algunas de las funciones disponibles para la versión 0.0.5 son:

· Export de pins GPIO
· Configurar de los pins GPIO para entrada/salida
· Configurar el limite de detección de cambio para los pins GPIO
· Enviar pulsos
· Leer los estados de los pins GPIO
· Permanece a la escucha de cambios de estado (basado en interrupciones)

6 pruebas

Introducción

Para asegurarnos de que el software que hemos programado cumple con los requisitos especificados y que funciona correctamente, someteremos el software a las pruebas requeridas para hacer estas comprobaciones.

Pruebas realizadas

Recepción/Envío descubrimiento

Creación del Thread

En caso de que en Main de la aplicación del servidor encuentre un error al crear la clase TCPServer que extiende de Thread. Este debe lanzar una expcepción NullPointerException.

Iniciaremos la aplicación varias veces para comprobar que la creación se crea correctamente.

Forzaremos que falle la creación de la clase para comprobar que capturamos la excepción.

Tipo de prueba	Resultado
Creación del Thread	**CORRECTO**
Forzamos el error y comprobamos excepción	**CORRECTO**

Creación del socket

Comprobamos que se crea el socket correctamente y permite el envío y recepción de datos. En caso de que surja algún problema al abrir el socket lanzará una expcepción IOException. Comprobaremos si controla la excepción lanzada, informa al usuario mostrándole el mensaje por consola y para la ejecución del programa.

Si el número de puerto al que se pone a escuchar el socket está fuera del rango de puertos permitidos (0 a 65535) se lanzará una excepción del tipo IllegalArgumentException. Comprobaremos que captura la excepción, informa al usuario mostrándole la información por consola y paramos la ejecución del programa.

Tipo de prueba	Resultado
Creación del socket	**CORRECTO**
Captura error número puerto fuera del rango	**CORRECTO**

Máximo de conexiones del servidor

Existe un máximo de clientes permitidos haciendo peticiones al servidor.
Cuando se supera el límite de clientes conectados el servidor debe rechazar la conexión. Comprobaremos que el comportamiento al alcanzar el máximo + 1 por parte del servidor es el de cerrar la conexión con el cliente.

Tipo de prueba	Resultado
Cerrar socket al sobrepasar el límite de conexiones	**CORRECTO**

Comprobación de cambio de IP

Se crea un Thread que ejecutará una comprobación de la IP cada x tiempo. Comprobaremos que se crea correctamente y forzaremos que falle para comprobar que lanza la excepción NullPointerException. En caso de que falle la creación se notificará por consola al usuario con un mensaje y reintentará la creación del socket.

Comprobaremos que la consulta de cambio de IP se realiza en el tiempo especificado.

Forzaremos el cambio de IP para comprobar si envía la notificación y el dispositivo lo recibe con la nueva IP.

Tipo de prueba	Resultado
Creación del Thread	**CORRECTO**
Consulta de IP en el tiempo especificado	**CORRECTO**
Forzamos el error y comprobamos excepción	**CORRECTO**

Bucle de recepción de peticiones

Cuando el cliente ha establecido la conexión con el servidor, el servidor se queda en un bucle esperando a la recepción de peticiones. Se detendrá el bucle en el caso de que el cliente cierre la conexión o envíe la orden explícita de cerrar la conexión.

Comprobaremos que el bucle se detiene cuando recibe la petición de parar y cierra la conexión con el cliente correctamente y acaba el Thread correspondiente con esta conexión.

Comprobaremos que el servidor cierra el socket si el cliente cierra la conexión repentinamente. Se lanzará una excepción que deberemos capturar. Saldremos del bucle y acabará el Thread.

Tipo de prueba	Resultado
Recibimos la orden de detener la ejecución	**CORRECTO**
Sale del bucle y cierra la conexión con el cliente	**CORRECTO**
Se cierra el Thread	**CORRECTO**

Tipo de prueba	Resultado
Cierra el socket si el cliente cierra la conexión	**CORRECTO**
Tratamos la excepción y salimos del bucle	**CORRECTO**
Se cierra el Thread	**CORRECTO**

Peticiones de clases o métodos no existentes

El cliente envía las peticiones especificando la clase y el método a ejecutar y los parámetros si se requiere. En el caso de que no exista la clase o el método se lanzarán excepciones.

Cuando se solicita un método o una clase que no existe, el tipo de los parámetros no es correcto u otras posibles situaciones en las que no se pueda llevar a cabo la llamada al objeto y método requerido, se lanzarán las siguiente excepciones: **NoSuchMethodException, SecurityException, IllegalAccessException, IllegalArgumentException, InvocationTargetException**

Tipo de prueba	Resultado
Capturamos la excepción que se lanza al invocar una clase inexistente	**CORRECTO**
Capturamos la excepción que se lanza al invocar un método inexistente	**CORRECTO**
Capturamos la excepción que se lanza al invocar un método con tipos erróneos, o número de parámetros incorrectos.	**CORRECTO**

Obtener temperatura

El envío de la temperatura se envía cada vez que esta cambia y mientras el usuario no cancele el envío.

Comprobaremos que envía la temperatura cada vez que cambia la temperatura aplicando calor al sensor de temperatura y dejando que vuelva a la temperatura ambiente donde el cambio de temperatura es mínimo y el envío del cambio se realizará con menos frecuencia.

Tipo de prueba	Resultado
Envío de la temperatura con cada cambio de temperatura siempre que el usuario no cancele la monitorización.	**CORRECTO**

Detener procesos que se ejecutan cada x tiempo

Los procesos que se han de ejecutar cada x tiempo se ejecutan en Threads aparte.
Se creará un objeto que guardará el proceso y la clase y el método que lo ha ejecutado. Cada vez que se cree uno de estos objetos se almacenará en una lista.

El usuario enviara la petición de detención de ese proceso, especificando la clase y el método. Se buscará ese proceso en la lista y se detendrá. Comprobaremos que encuentra el proceso, y que lo detiene. Si no existe el proceso mostrará por consola que no existe ningún proceso con esas características y continuará el programa normalmente.

Tipo de prueba	Resultado
Creación del objeto ScheduledRequests que identifica los procesos que se ejecutan cada x tiempo	**CORRECTO**
Detiene el proceso cuando el usuario solicita la detención	**CORRECTO**
Si no se encuentra ningún método con los parámetros especificados, muestra un mensaje por consola y continua la ejecución.	**CORRECTO**

Gestión sensor y pins Raspberry

Comprobaremos que podemos encender y apagar la calefacción envíando las peticiones a los pins seleccionados.

Detección y lectura de la temperatura del sensor, en caso de que no pueda leer la temperatura el valor será 0.

El acceso a los recursos de Raspberry se han de hacer como superusuario, en caso contrario la aplicación se detiene.

Tipo de prueba	Resultado
Encendido y apagado	**CORRECTO**
Sensor de temperatura	**CORRECTO**
Acceso como superusuario y como usuario sin privilegios	**CORRECTO**

7 conclusiones

Introducción

Dedicaremos el último apartado de la memoria para aportar las conclusiones finales del proyecto que se ha realizado.

7.1 Valoración

Para llevar a cabo el proyecto se han utilizado muchas tecnologías diferentes y de diferentes complejidades.

Podríamos diferenciar 3 grandes bloques, que han supuesto grandes retos para solucionar.

Por un lado el uso de Raspberry Pi ha supuesto que tuviese que documentarme sobre el aparatdo electrónico del proyecto com el uso de sensores, relés, resistencias y conexiones. La investigación sobre como funcionan este tipo de dispositivos me permitido tener una visión global de los proyectos que se pueden llegar a llevar a cabo con este tipo de dispositivos.
Además esta investigación me he parmitido conocer los otros dispositivos que complementan a Raspberry, como pude ser Arduino, y el gran potencial que esta combiandión dispone.

En lo referente a la comunicación entre diferentes dispositivos mediante sockets y el uso de Threads ha sido de las partes más complejas de resolver. Esto me ha supuesto la investigación y la lectura de muchos recursos y por supuesto muchas hora de programación.

Y por último destacaría la parte de Android que ha sido la otra parte que ha requerido muchas horas de investigación y búsqueda en los difertentes tutoriales que Google nos ofrece.

Lo más importante a destacar es el conocimiento que he obtenido al tener que investigar en los tres bloques descritos anteriormente. Ha supuesto todo un reto personal y una gran satisfacción poder cumplirlo con éxito.

Además me ha hecho interesarme mucho por los proyectos que se desarrollan con Raspberry y Android. Sin duda continuaré investigando con todo lo relaciondo con estos campos ya que, personalmente, el campo de innovación es realmente grande.

7.2 Objetivos marcados

Por la naturaleza del proyecto y de que trabajamos con tecnologías de las que hemos ido aprendiendo durante la realización de este, durante el desarrollo del proyecto hemos encontrado problemas inesperados que han ralentizado el progreso.

Debido a esto no hemos podido cumplir con algunas de las funcionalidades que se habían propuesto llevar a cabo, debido a la falta de tiempo.

A pesar de esto se han cumplido con todas las funcionalidades necesarias para hacer las dos aplicaciones, tanto Raspberry como Android, funcionen satisfactoriamente y ofrecer un producto acabado al usuario.

7.3 Lineas futuras

Habiendo cumplido con prácticamente todos los requisitos propuestos y basándonos en el objetivo global del que se propuso este proyecto, se propondrán ampliaciones al proyecto.

Registros

Uno de los objetivos del proyecto era el del control de diferentes dispositivos y el de la recolección de los datos que estos dispositivos u otros sensores nos puedan proporcionar.

La recolección de estos datos es loq ue le aportará la información al usuario sobre sus consumos.

Por tanto una de las ampliaciones futuras será la ampliación de la base de datos del servidor con diferentes tablas para capturar información de otros dispositivos o sensores.

Estadísticas

Los datos recogidos y almacenados deberían poder mostrarse al usuario de diferentes maneras para que le sean los más útiles posibles.

La ampliación de las formas de visualización de estos datos tanto en dispositivos móviles como en una aplicación web, o el uso de filtros para seleccionar los datos precisos, sería otra de las ampliaciones a llevar a cabo.

Aplicación WEB

Aunque la consulta de los datos el teléfono móvil es una forma fácil y rápida de consulta de datos, esta es bastante limitada.

La flexibilidad que nos ofrecen las aplicaciones web es un punto a considrear para posibles ampliaciones, y ofrecer al usuario otras formas de consulta de sus datos con formas más potentes y visuales.

Control otros dispositivos

El control de otros dispositivos o componentes de una casa, como podrían ser las luces, es uno de los campos naturales de ampliación de este proyecto. Además la integración del control de diferentes dispositivos en una misma aplicación.

8 bibliografía

[1] Android Developers
http://developer.android.com/index.html

[2] Raspberry Pi
http://www.raspberrypi.org/

[3] The Pi4J Project
http://pi4j.com/

[4] RPi Low-level peripherals
http://elinux.org/RPi_Low-level_peripherals

[5] Professional Android 4 Application Development
Reto Meier, Wrox

[6] Bruce Eckel, Piensa en Java 2ª Edición
Pearson, Prentice Hall

[7] Java 7 API Specification
http://docs.oracle.com/javase/7/docs/api/

[8] Java Tutorials: Concurrency
http://docs.oracle.com/javase/tutorial/essential/concurrency/

[9] Java Tutorials: Java Remote Method Invocation (RMI)
http://docs.oracle.com/javase/tutorial/rmi/index.html

[10] Java Tutorials: Sockets
http://docs.oracle.com/javase/tutorial/sdp/index.html

[11] SQLite Documentation
http://www.sqlite.org/docs.html

[12] Sommerville, Ingeniería del software 9ª Edición
Addison-Wesley, Pearson

[13] Raspberry Pi Temperature Sensor, University of Cambridge
http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/temperature/

[14]Google Cloud Messaging for Android
http://developer.android.com/google/gcm/index.html

Anexos

Índice de tablas

Índice de figuras

Índice de diagramas

Glosario

Hardware
El término hardware hace referencia a los diferentes componente físicos que forman parte de un sistema informático.

Microcontrolador
Un microcontrolador es un circuito integrado (también conocido como chip) que se encarga de ejecutar las instrucciones almacenadas en la memoria. Un microcontrolador está compuesto por una unidad central de procesamiento, memoria y periféricos de entrada y salida.

Red local
Interconexión de diferentes dispositivos en una misma red.

UDP
UDP es el acrónimo de **User Datagram Procol**. Es un protocolo perteneciente a la capa de transporte y permite el envío de datagramas sin que exista un establecimiento previo de la conexión.

TCP
TCP es el acrónimo de **Transmision Control Protocol**. Al igual queUDP perenece a la capa de transporte. TCP es un protocolo orientado a la conexión, lo que significa que prèviamente establece una conexión entre los dos dispositivos. Además permite el control de errores de envío/recepción.

BBDD
Es la base de datos donde se almacenan los datos, donde esta información es indexada y estructurada de forma que se pueda obtener esta información posteriormente de forma rápida.

Memoria RAM
Es la memoria de acceso rápido de la que disponen todos los ordenadores. Es donde se almacenan los datos de los programas en ejecución en un ordenador y las instrucciones que estas deben hacer.

Plugin
Los plugins son pequeñas funcionalidades que se instalan en un programa ya existente, para añadirle nuevas características y funcionalidades.

Gestor de control de versiones
Los gestores de versiones facilitan el desarrollo del software, almacenando los cambios que se van produciendo a lo largo del desarrollo. Nos permite hacer modificaciones en el código de un programa sin afectar a la versión principal y posteriormente unir los dos códigos. También facilita el desarrollo en equipo ya que evita que un desarrollador pise los cambios realizados por otro.

Repositorio
El repositorio es donde se almacena el código que gestiona el Gestor de control de versiones. Todos los desarrolladores obtendrán el código de este repositorio central y lo actualizarán con los nuevos cambios que haya realizado.

GPIO
GPIO es el acrónimo de **General Purpose Input Output** son los pins de Raspberry Pi donde podremos conectar dispositivos externos, como pueden ser leds, motores de pulsos, circuitos externos.

I2C (Inter-Integrated Circuit)
Algunos de los pins de Raspberry Pi disponen de I2C, que es un bus de comunicaciones en serie. Se usa para comunicar microcontroladores y sus periféricos.

PWM (Pulse With Modulation)
Algunos de los pins de Raspberry Pi disponen de PWM, que es una señal que se envía periódicamente para transmitir información para, por ejemplo, el control de motores.

Ethernet
Ethernet es una tecnología que se usa en las redes locales y que especifica las características de cableado y señalización del nivel físico y los formatos e las tramas de datos de la capa de enlace de datos.

CPU
Es la parte principal de un ordenador, es donde se llevan a cabo todas las instrucciones especificadas por los programas, las interpreta y las ejecuta.

GPU
Es un procesador similiar a la CPU pero que en este caso se encarga únicamente del procesamiento gráfico.

SoC (System on Chip)
Son las tecnologías que integran algunos componentes que forman parte en un ordenador, en un único circuito integrado.

ARM
Es un arquitectura de procesadores que está compuesta por 32 instrucciones. Son procesadores simples y ideales para dispositivos de baja potencia, por este motivo son los que más se usan en dispositivos móviles.

Hardware libre
Son los dispositivos de hardware de los cuales las especificaciones y los diagramas son públicos y que pueden ser recreados por cualquiera.

Dalvik Virtual Machine
Es la máquina virtual que utiliza Android para ejecutar los programas de Android.

Framework
Es un conjunto de herramientas y patrones que usaremos para la implementación de una aplicación, donde nos proporcionará las librerías y clases necesarias.

SDK
Es el kit de desarrollo necesario para poder crear programas para un sistema concreto.

Kernel
El kernel es la parte principal de un Sistema Operativo. Facilia el acceso seguro al hardware, gestiona la memoria y gestiona los procesos.

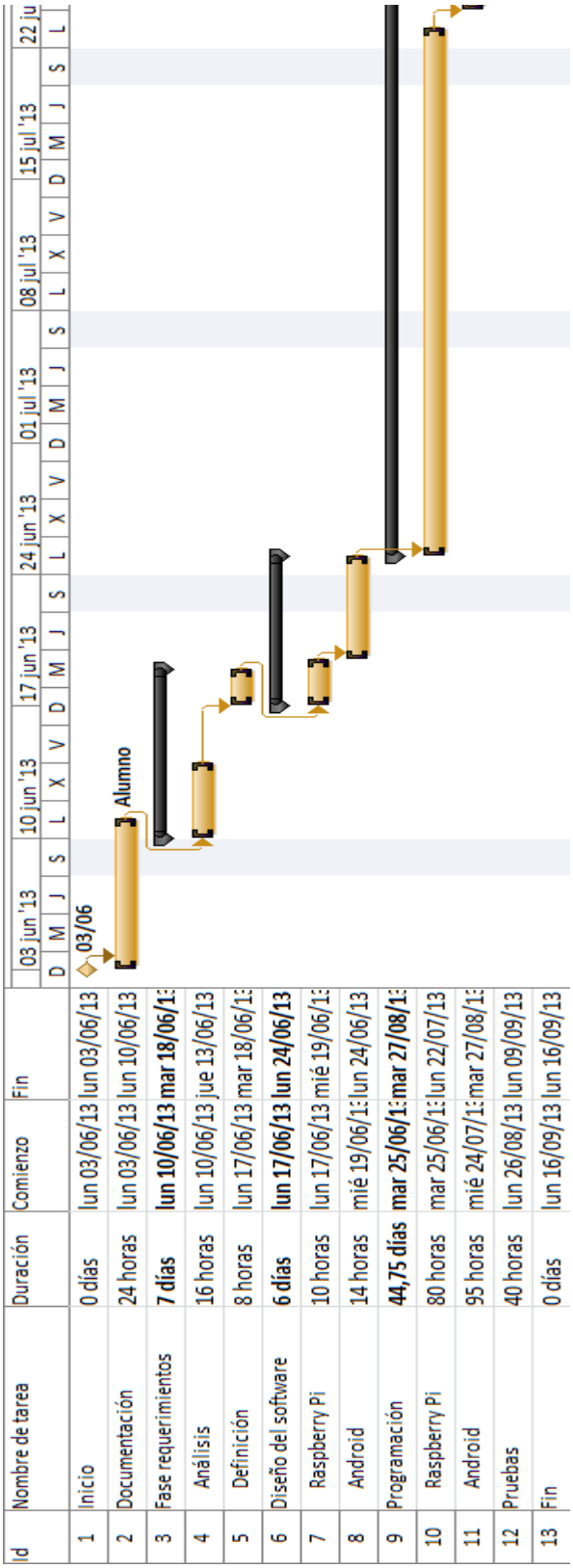
Sockets
Un socket es una abstracción que representa los puntos de conexión de un ordenador a otro a través de una red.

Puertos
Un puerto está estrechamente relacionado con el socket, ya que representa dónde nos hemos de conectar del socket y que parte de nuestro socket dejamos para que otros se conecten a nosotros.

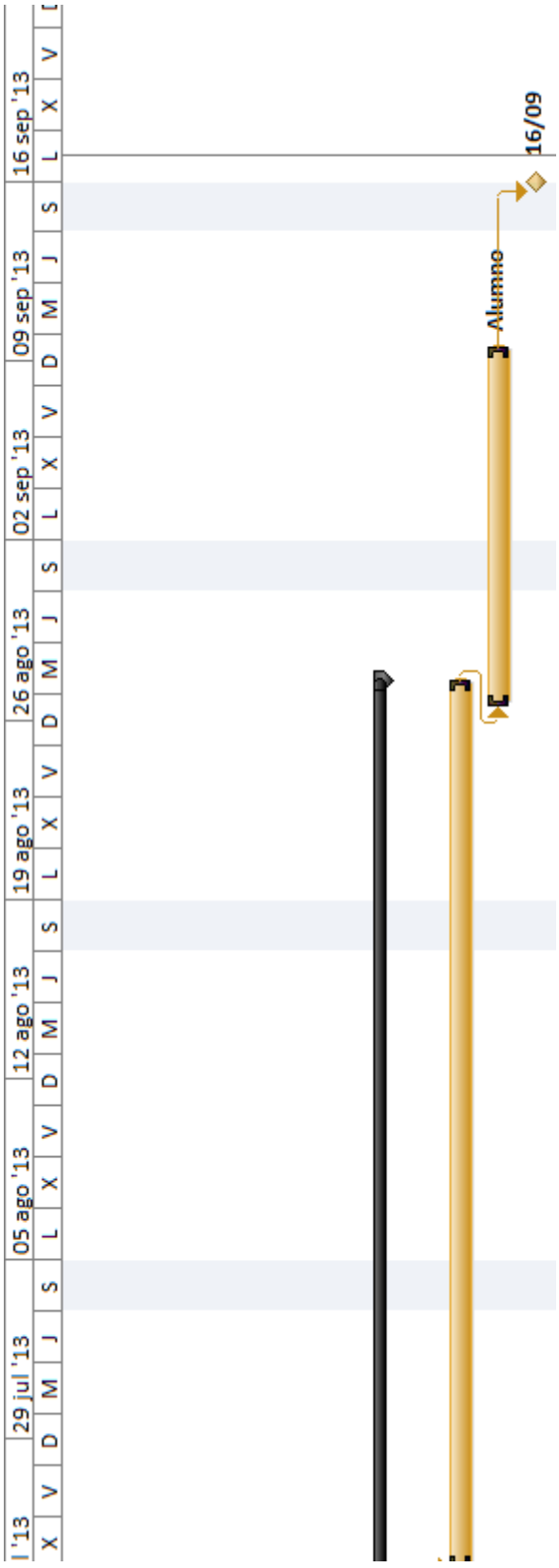
IP
Es un protocolo de internet que identifica los dispositivos dentro de esta.

ISP
Son las empresas que contratamos para que nos provean acceso a internet.

Diagrama de Gantt



Id	Nombre de tarea	Duración	Comienzo	Fin
1	Inicio	0 días	lun 03/06/13	lun 03/06/13
2	Documentación	24 horas	lun 03/06/13	lun 10/06/13
3	Fase requerimientos	**7 días**	**lun 10/06/13**	**mar 18/06/1**
4	Análisis	16 horas	lun 10/06/13	jue 13/06/13
5	Definición	8 horas	lun 17/06/13	mar 18/06/1
6	Diseño del software	**6 días**	**lun 17/06/13**	**lun 24/06/13**
7	Raspberry Pi	10 horas	lun 17/06/13	mié 19/06/1
8	Android	14 horas	mié 19/06/1	lun 24/06/13
9	Programación	**44,75 días**	**mar 25/06/1**	**mar 27/08/1**
10	Raspberry Pi	80 horas	mar 25/06/1	lun 22/07/13
11	Android	95 horas	mié 24/07/1	mar 27/08/1
12	Pruebas	40 horas	lun 26/08/13	lun 09/09/13
13	Fin	0 días	lun 16/09/13	lun 16/09/13



Sabadell, Septiembre de 2013

Firmado: Albert Férriz Pérez