UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE SISTEMAS AUDIOVISUALES

TRABAJO FIN DE GRADO

DESARROLLO DE UNA APLICACIÓN MÓVIL ANDROID PARA CONTROL REMOTO DE UN SERVICIO WEB

Autor: María Lozano Pérez

Tutor: Mario Muñoz Organero

A todos los que me han ayudado, de una manera o de otra, a llegar hasta aquí. Especialmente a mi familia y amigos.

Resumen

El objetivo de este trabajo es el desarrollo de una aplicación para el sistema operativo Android, que permita obtener una interfaz de usuario conectada a un servicio web.

Para ello, se han utilizado una gran variedad de tecnologías diferentes que, al final, componen un sistema completo que facilita la tarea de manejar aparatos electrónicos de una manera remota, desde un dispositivo con sistema operativo Android.

El desarrollo propuesto no sería válido en el escenario actual de telecomunicaciones, pero se espera que en el futuro las tecnologías sigan este camino.

Índice general

Índice de figuras

Índice de códigos

1. INTRODUCCIÓN

1.1. Introducción

En la sociedad actual, los dispositivos móviles son el elemento del que más dependemos. Desde que salieron al mercado han avanzado de una manera sorprendente. Han evolucionado hasta su posible utilización en casi cualquier ámbito. Este cambio se ha dado en un corto espacio de tiempo, en el que hemos pasado de utilizar estos dispositivos solamente como teléfonos portátiles a usarlos para cualquier tarea imaginable. En los próximos años, este crecimiento va a seguir siendo previsiblemente tan frenético como lo es ahora.

Las tecnologías que, en este momento, se están desarrollando y empezando a aplicar y que tienen relevancia en este proyecto, son las que están empezando a dotar, a todos los aparatos electrónicos y otros objetos, de acceso a la red Internet. Esto, a lo que se ha llamado "The Internet of Things" o, en español, "El Internet de las Cosas", implicaría un enorme abanico de posibilidades en el mundo de la computación.

Una de estas posibilidades de esta nueva tecnología es la que se va a tratar en este proyecto, el manejo de dispositivos electrónicos por medio de un dispositivo móvil, ya sea un teléfono o una tableta.

En este proyecto realizaremos una aplicación para el sistema operativo portátil Android, que junto a otras tecnologías, permitirá centralizar el uso a distancia de los aparatos electrónicos dentro del entorno del hogar.

1.2. Motivación y objetivos

Actualmente, en cada hogar existen multitud de electrodomésticos, que van acompañados cada uno por sus respectivos mandos a control remoto, para poder manejarlos de una manera más cómoda.

Esto que, en un principio, debería implicar una mayor comodidad acaba siendo una molestia por varias razones, que se enumeran a continuación.

- Si no lleva el suficiente etiquetado o no se es una persona que entienda de este tipo de aparatos, se pueden confundir unos mandos con otros o podemos no saber cuál es el que buscamos.

- Para algunas opciones es necesario utilizar al menos dos mandos a la vez para hacer una única función, por ejemplo, ver un DVD, para lo que necesitamos el mando de la televisión y el del reproductor de DVDs.

- Cada mando tiene que llevar una alimentación individual para que pueda funcionar. Si los eliminásramos conllevaría un ahorro tanto energético como ecológico.

- Es fácil, si no se tiene un lugar fijo para dejarlos, que estos dispositivos se pierdan.

Esta situación podría ser mucho más fácil si pudiéramos concentrar todos los mandos en un sólo dispositivo. Y para hacerlo todavía más manejable si cabe, podemos hacer que este único dispositivo sea nuestro teléfono móvil, que siempre llevamos encima y que nunca (o casi nunca) se pierde.

1.4. Contenido de la memoria

En esta sección se va a detallar el contenido de los diferentes apartados de los que se compone esta memoria para hacer más fácil su lectura.

En el segundo capítulo se explica el estado actual de las tecnologías utilizadas en el desarrollo de este proyecto.

En el tercer capítulo, se describe la arquitectura del sistema a desarrollar indicando los requisitos que debe cumplir y la explicación del esquema general de la aplicación.

En el cuarto capítulo se presentan las soluciones que se han desarrollado además del entorno de aplicaciones usado.

En el quinto capítulo se explican las pruebas realizadas a partir de las soluciones a las que se han llegado.

En el sexto capítulo se recogen las conclusiones sacadas del trabajo realizado, además de describir posibles líneas futuras en las que podría avanzar el proyecto.

En el séptimo capítulo se agruparán todos los recursos utilizados.

2. ESTADO DEL ARTE

2.1. "El Internet de las Cosas"

La comunicación entre ordenadores [1] empezó por dos maquinas que intercambiaban información al estar conectadas, de ahí evolucionó a la red Internet propiamente dicha, mediante la interconexión de todos los ordenadores de alrededor del mundo. El siguiente paso ha sido incluir en esta conexión otros elementos que no son ordenadores propiamente dichos como smartphones y tablets. Ahora Internet está siendo transformado en la herramienta de comunicación que permitirá a los objetos interactuar entre ellos.

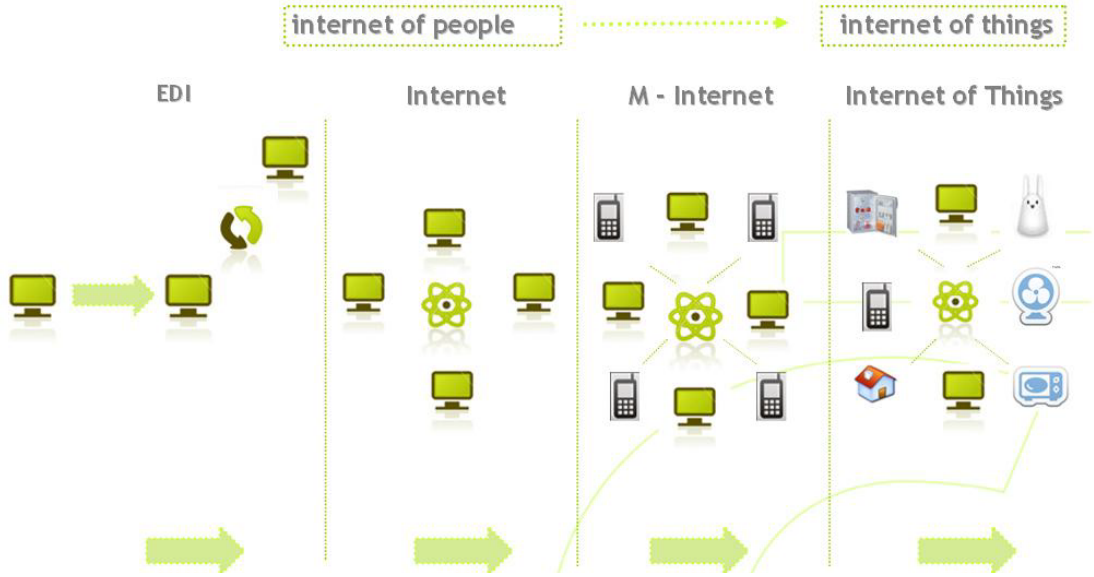

Figura 1. "El Internet de las Cosas"

Para llegar a esta interacción, los elementos de una red tienen que estar identificados por una dirección dentro de ella. Aquí es donde entra el protocolo IP.

IP es un protocolo de comunicación usado en Internet y desarrollado en la década de 1970. Generalmente se usa conjuntamente con el protocolo de transporte TCP. La información en este protocolo se transmite en paquetes de datos. Cada paquete IP incluye una cabecera que cuenta con información sobre el origen, el contenido y el destino del paquete.

Este protocolo soporta direcciones únicas para ordenadores dentro de una red. Hasta hace muy poco la versión IPv4 era la que se utilizaba para la mayoría de tráfico de Internet, pero con el creciente aumento de usuarios en todo el mundo y la utilización de las direcciones IP

también para objetos, se ha tenido que crear una nueva, IPv6, que poco a poco va a ir implantándose alrededor del mundo.

La diferencia principal entre estas dos versiones del protocolo IP es la cantidad de bits que usan para las direcciones. En la versión 4 se usan 32 bits, lo que permite tener aproximadamente 4,3 billones de direcciones, mientras que en la nueva versión 6, las direcciones son de 128 bits, es decir, posibilita tener 340 sextillones (1 sextillón = 10^{36})de direcciones. Además, esta versión también permite un tamaño de paquetes más grande.

El formato también ha cambiado. Una dirección IPv4 se escribe como cuatro números del 1 al 255 separados por puntos (192.168.1.32), mientras que una IPv6 se escribe como ocho grupos de cuatro dígitos hexadecimales (2001:0db7:64a3:08b3:1d19:4a2f:65a0:2364).

Podemos decir, entonces, que "El Internet de las Cosas" será un escenario en el cual habrá más objetos conectados a Internet que personas, y estos objetos estarán constantemente obteniendo información de manera real procedente del medio en vez de estar supeditados a que la información que contiene la red sea totalmente introducida por humanos.

2.2. M2M (Machine to Machine)

El término M2M [2] se refiere a la tecnología que permite a todos los ordenadores, objetos, sensores, dispositivos móviles, etc., interactuar entre ellos, hacer medidas y tomar decisiones, a menudo sin intervención humana. En esta arquitectura podemos diferenciar dos integrantes: Los usuarios M2M, que son las personas que se conectan a la red por medio de ordenadores o equipos similares, y los terminales M2M, dispositivos de tercera generación especialmente adaptados para la comunicación M2M.

La tecnología M2M [3] se refiere a sistemas que permiten a las máquinas comunicarse con sistemas de información de compañías y otras máquinas, o con los dispositivos móviles de las personas, y proporcionar datos en tiempo real. Se utiliza un enlace sin cables para monitorización y control y la transferencia de datos ocurre mediante peticiones o intervalos de tiempo predeterminados.

Las soluciones M2M son creadas para aumentar los beneficios y competitividad de una compañía a través de procesos más eficientes, mejor servicio a clientes o maneras nuevas de hacer las cosas. M2M trata de conectar a gente, dispositivos y máquinas. Trata de permitir a las máquinas hablar.

La gran variedad de soluciones de comunicación M2M son desarrolladas para enviar indicaciones de situaciones inusuales, recoger información o cambiar parámetros de acuerdo a

las necesidades del negocio. Las nuevas aplicaciones M2M están emergiendo continuamente para servir a todas las áreas de negocio: monitorización de ascensores en centros comerciales, descarga de nuevos juegos en consolas, comprobar la temperatura del agua en piscinas, por nombrar algunas.

Las mejores soluciones M2M son creadas combinando las competencias de los expertos en el mercado y después ofreciendo a los clientes soluciones completas, sin separación de elementos. Las soluciones se desarrollan en diferentes campos como las tecnologías de la información, desarrollo de software, integración de hardware, atención al cliente y facturación. Como las soluciones son diseñadas para cumplir las especificaciones de cada situación y cliente, se necesita un conocimiento especializado del entorno empresarial.

Las soluciones M2M son normalmente inversiones a largo plazo, que hace hincapié en la importancia de las soluciones que se adapten a lo que pueda pasar en el futuro.

2.2.1. GSM. LA TECNOLOGÍA DE M2M

A continuación, se explica porqué la tecnología GSM sería una buena opción para el despliegue de los sistemas M2M.

GSM (Global System for Mobile Communications) es un sistema global y comprobado con beneficios considerables comparado con otras tecnologías. La historia de la estandarización de GSM permite la creación de soluciones GSM internacionales. Con el despliegue de nuevas bandas de frecuencias las solucions M2M son una realidad. Por lo tanto no es una exageración decir que la tecnología GSM permite la creación de soluciones M2M realmente internacionales.

Hay muchos proveedores de esta tecnología, lo que permite el cumplimiento de una gran variedad de necesidades del cliente. Además, al haber muchos proveedores se produce una gran estabilidad y compatibilidad en estos sistemas.

Un gran beneficio de las tecnologías GSM son los servicios avanzados de datos, incluyendo las componentes de seguridad que ya están disponibles con las redes y productos GSM. Los servicios de datos están siendo mejorados continuamente, ayudando a la continuidad de las soluciones M2M construidas sobre una tecnología GSM.

Las soluciones GSM son baratas de instalar, de hecho la conectividad GSM puede introducirse en las máquinas por el fabricante en vez de ser instalado después a un coste mayor. No se necesitan cables, el equipamiento con conectividad GSM puede moverse de una localización a otra.

La itinerancia entre las redes de diferentes operadores es más fácil cada día, esto solo hace mejorar la movilidad de las soluciones M2M construidas en la tecnología GSM.

Un aspecto importante de la tecnología GSM es que permite introducir el negocio M2M en estos momentos. Con los sistemas GSM existentes en más de 200 países, la tecnología se está convirtiendo en la más famosa del mundo para la evolución de la tercera generación.

2.3. Dispositivos Móviles

Los dispositivos móviles son aparatos pequeños, con capacidades de procesamiento, con conexión permanente o no a alguna red, que tienen una memoria limitada, diseñados para una función, pero que pueden realizar otras más generales. Un ejemplo de dispositivo móvil clásico serían los teléfonos móviles, para los que su función básica sería la posibilidad de hacer llamadas, pero que también puede llevar a cabo otras como enviar mensajes, utilizar una agenda...

Se denomina *smartphone* o *teléfono inteligente* [4] a un tipo de dispositivo móvil que ya no tiene como objetivo principal ser emisor y receptor de llamadas telefónicas. Estos móviles suelen estar permanentemente conectados a Internet y, además, tienen una capacidad de procesamiento mucho más cercana a la de un ordenador que a la capacidad con la que contaban sus predecesores.

Las características principales que diferencian a un *smartphone* de un teléfono móvil tradicional son:

- **Sistema operativo.** Estos dispositivos están basados en un sistema operativo sobre el que ejecutan sus aplicaciones.

- **Aplicaciones.** Mientras que los móviles convencionales tenían una serie de funciones muy limitadas, éstos tienen la habilidad de hacer un mayor número de tareas, desde ver documentos de texto a obtener direcciones mediante GPS. Además, son extremadamente personalizables, ya que cada persona puede descargarse las aplicaciones que más le convengan.

- **Acceso a la web.** Cada vez más *smartphones* pueden acceder a la web a velocidades más altas, gracias a las tecnologías 3G y 4G así como al soporte de conexión Wi-Fi de estos terminales.

- **Teclado QWERTY.** Los teléfonos móviles siempre han tenido un teclado alfabético, pero en este momento en el que tenemos funciones para las que es necesario escribir textos más extensos, es mucho más cómodo contar con un teclado QWERTY.

2.4. Sistemas operativos de dispositivos móviles

Un sistema operativo [5] puede definirse como un conjunto de software y hardware que hacen funcionar a una máquina y proporcionan un entorno para la ejecución de programas. Además, los programas utilizan al sistema operativo como intermediario para obtener acceso a recursos del sistema como el procesador, archivos y dispositivos de entrada y salida. Por lo tanto, podemos decir que el sistema operativo constituye la base sobre la cual se van a escribir los programas de la aplicación. Para los usuarios es una interfaz que les ayuda a comunicarse con el hardware.

Un sistema operativo actúa gestionando los recursos del sistema como la memoria, responde a las peticiones del usuario e intenta optimizar el rendimiento general de la máquina en el que se encuentra instalado.

En el siguiente gráfico se puede comparar el uso de los principales sistemas operativos móviles que existen actualmente.

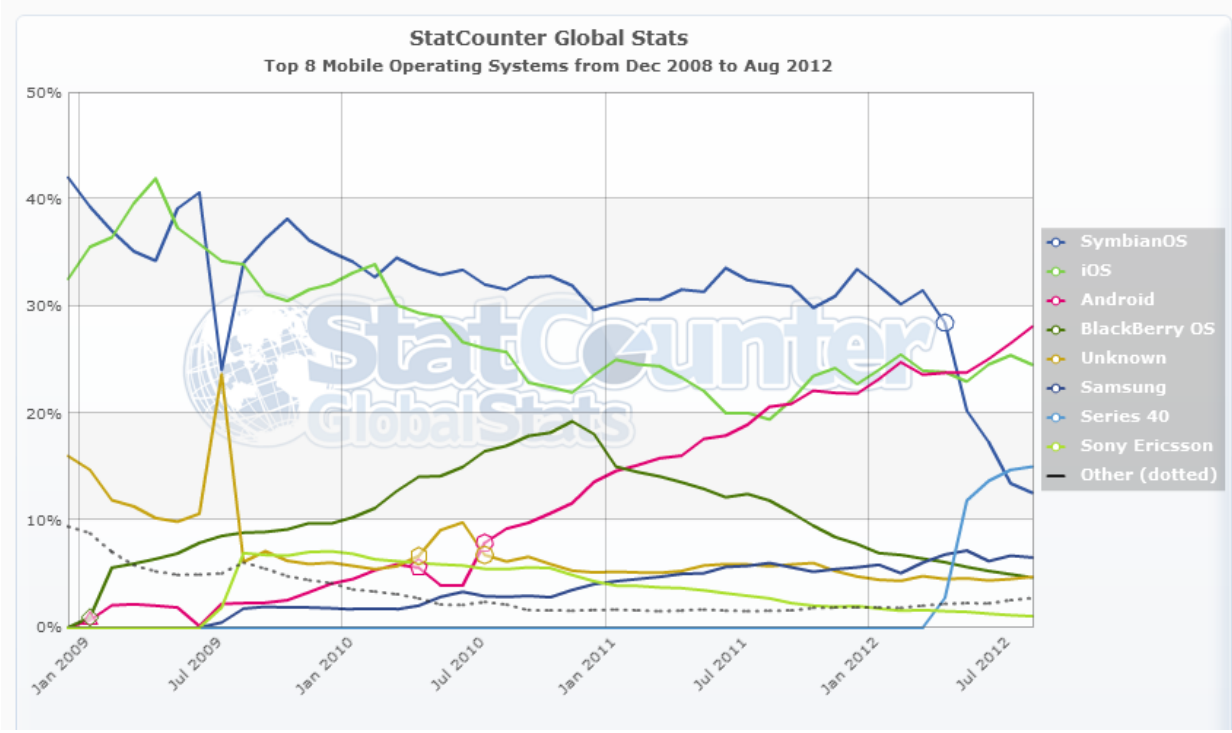

Figura 2. Sistemas operativos más utilizados en el mundo

Como podemos ver en la imagen [6], el sistema más utilizado en todo el mundo hasta hace muy poco era Symbian, que ahora ha sido superado por iOs y Android, que se encuentra

en la primera posición en estos momentos. Hay que tener en cuenta que este gráfico se refiere a todo el mundo y que depende de en qué continentes, según el poder adquisitivo y otros factores, van a utilizarse unos sistemas más que otros.

Por ejemplo, como se puede ver en el siguiente gráfico, que representa el mismo periodo de tiempo, pero tan solo se refiere al continente europeo, iOs es el sistema más utilizado, seguido muy de cerca por Android, que es el que ha sufrido un mayor crecimiento.

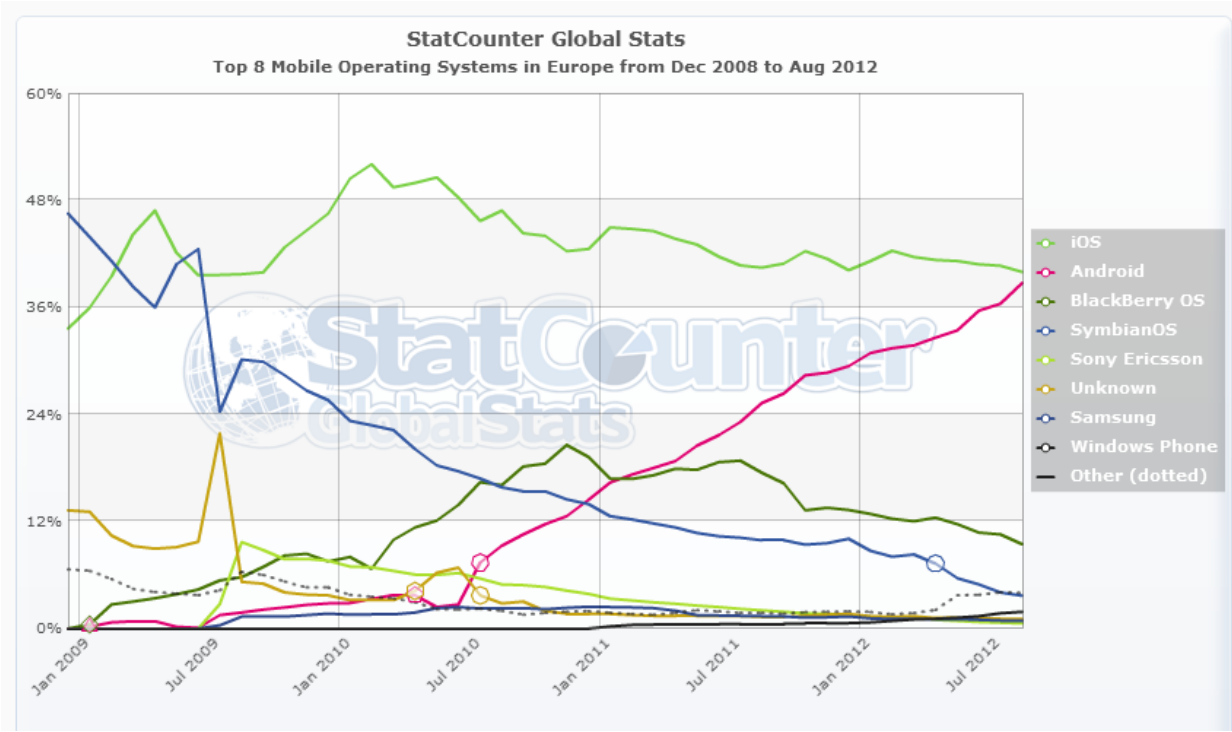

Figura 3. Sistemas operativos más utilizados en Europa

A continuación se van a explicar algunos de estos sistemas operativos móviles más populares.

2.4.5. ANDROID

Está basado en una versión modificada del kernel de Linux [7]. Su propietario es la empresa Google. Es de código abierto, lo que quiere decir que cualquier desarrollador puede crear y desarrollar aplicaciones para este sistema. Su entorno de ejecución está basado en Java.

Está instalado en dispositivos de una gran variedad de marcas. Esto hace que según las características del terminal, podremos instalar una versión u otra del sistema y, por lo tanto, no se podrán ejecutar ni las mismas aplicaciones ni con las mismas prestaciones en todos ellos.

2.4.1. SYMBIAN

Symbian OS [8] es un sistema operativo que fue creado por una alianza de varias empresas de telefonía móvil entre las que están Nokia, Sony Ericsson, PSION Samsung, Siemens, Benq, Fujitsu, LG, Motorola, etc. Sus orígenes provienen del sistema EPOC32 que era utilizado en PDA's de PSION. El objetivo de su creación fue competir contra otros sistemas como el de Palm o Windows Mobile.

Algunas de sus características son:

- Rendimiento: Está diseñado para optimizar el tiempo de vida de la batería.

- Multitarea: Sus componentes fundamentales son la telefonía y los mensajes. Todas las aplicaciones están diseñadas para funcionar en paralelo.

- Usa tecnologías basadas en estándares, asegurando que las aplicaciones son robustas, y portables.

- Utiliza una arquitectura de software orientada a objetos.

- Tiene una gestión de memoria optimizada al utilizar ejecutables de pequeño tamaño y códigos basados en ROM.

- Tiene reducidos requerimientos de memoria en tiempo de ejecución.

- Sus aplicaciones están orientadas al manejo de eventos en vez de a los hilos de ejecución. Se evita el uso de múltiples hilos en las aplicaciones porque produce sobrecarga.

2.4.2. BLACKBERRY OS

Desarrollado por la empresa Research in Motion (RIM) [9]. Se encuentra solo en terminales de su misma marca. El sistema permite multitarea y tiene soporte para diferentes métodos de entrada adoptados por RIM para su uso en computadoras de mano. Nacieron para ir dirigidos al mundo empresarial por su gestión del correo electrónico y la agenda.

Este sistema se está quedando atrás con respecto a otros por no ofrecer una tienda de aplicaciones tan grande como alguno de sus competidores. Es un dispositivo que utiliza Java e incluye su propia máquina virtual.

2.4.3. WINDOWS PHONE

Anteriormente llamado Windows Mobile [9], es un sistema operativo desarrollado por Microsoft. Se basa en el núcleo del sistema operativo Windows CE y cuenta con un conjunto de aplicaciones básicas utilizando las API de Microsoft Windows. Es un sistema multitarea y con capacidades multimedia. Se encuentra en dispositivos de varias marcas.

2.4.4. iOS

Es el sistema operativo que se incluye en los dispositivos Apple (iPhone, iPod y iPad) [10]. Está basado en la arquitectura de MAC OS X. En el nivel más alto, iOs actúa como un intermediario entre el hardware y las aplicaciones que aparecen en pantalla.

Su arquitectura está basada en capas, las capas más bajas contienen los servicios fundamentales y las tecnologías en las que se apoyan todas las aplicaciones y las capas más altas contienen servicios y tecnologías más sofisticados.

Es muy cerrado comparado con otros sistemas, porque es más restrictivo en la publicación de aplicaciones y mantiene un control mayor sobre ellas.

2.5. Android

Como ya se ha dicho antes, el sistema operativo Android está basado en el núcleo de Linux y es una plataforma de código abierto. En principio, se diseñó para dispositivos móviles y utiliza una adaptación del lenguaje de programación Java.

El primer teléfono con Android apareció en el año 2007, en el teléfono G1 de Google desarrollado en colaboración con T-Mobile.

2.5.1. CARACTERÍSTICAS

Algunas características [11] de este sistema operativo son:

- Tiene un entorno de aplicaciones que permite el reemplazo y la reutilización de los componentes.

- Consta de un navegador integrado basado en el motor open source Webkit.

- Sus aplicaciones pueden integrar directamente SQlite, que implementa bases de datos para un almacenamiento estructurado.

- Tiene soporte para infinidad de formatos de audio, vídeo e imágenes.

- Utiliza la máquina virtual Dalvik, optimizada para dispositivos móviles.

- Tiene gráficos optimizados alimentados por una biblioteca de 2D y 3D basado en la especificación OpenGL.

- Telefonía GSM.

- Incluye cámara, GPS, brújula y acelerómetro (depende del hardware).

- Suministra Bluetooth, EDGE, 3G y Wi-Fi.

- Tiene un entorno de desarrollo muy rico que incluye un emulador, herramientas de depuración y una extensión para el programa Eclipse.

2.5.2. DESARROLLO DE APLICACIONES

Los desarrolladores que quieran programar aplicaciones Android tienen que descargar su SDK desde la página de Google. Todas las aplicaciones se ejecutan sobre la máquina virtual Dalvik que está optimizada para ejecutarse en paralelo con otras máquinas en sistemas con baja memoria.

Preparar el entorno de desarrollo es fácil, hay que descargar el SDK y la plataforma eclipse e instalar la extensión de Android para ese programa. La versatilidad del desarrollo de aplicaciones Android puede verse porque se pueden crear programas de alta calidad usando las mismas herramientas que para desarrollar en el lenguaje de programación Java.

2.5.3. ARQUITECTURA

La arquitectura de la plataforma Android está compuesta por cinco capas: el núcleo de linux y herramientas de bajo nivel, las bibliotecas nativas, el entorno de ejecución de Android, el marco de aplicaciones y encima de todas, las aplicaciones.

- Núcleo linux: Proporciona servicios básicos como drivers de hardware, gestión de energía y de memoria y una capa de abstracción entre el hardware y el resto de las partes.

- Bibliotecas nativas: Android incluye un conjunto de bibliotecas C y C++, a las que se puede acceder mediante el marco de aplicaciones.

- Entorno de ejecución de Android: En esta capa se encuentran las bibliotecas de funciones básicas de Java y las específicas de Android. Además, la máquina virtual Dalvik, que esta basada en registros y ejecuta las clases compiladas por el compilador de Java que se transforman al formato *Dalvik Executable* (.dex) ejecutables por la máquina virtual.

- **Marco de aplicaciones:** Los desarrolladores tienen acceso total al código fuente usado en las aplicaciones básicas. De esta forma no se generan duplicados de componentes básicos que realizan las mismas acciones si no que se utilizan los que ya existen para evitar empezar a programar desde cero.

- **Aplicaciones:** En esta parte podemos ver las aplicaciones creadas con Android, tanto las nativas del sistema operativo como las que están creadas por terceros.

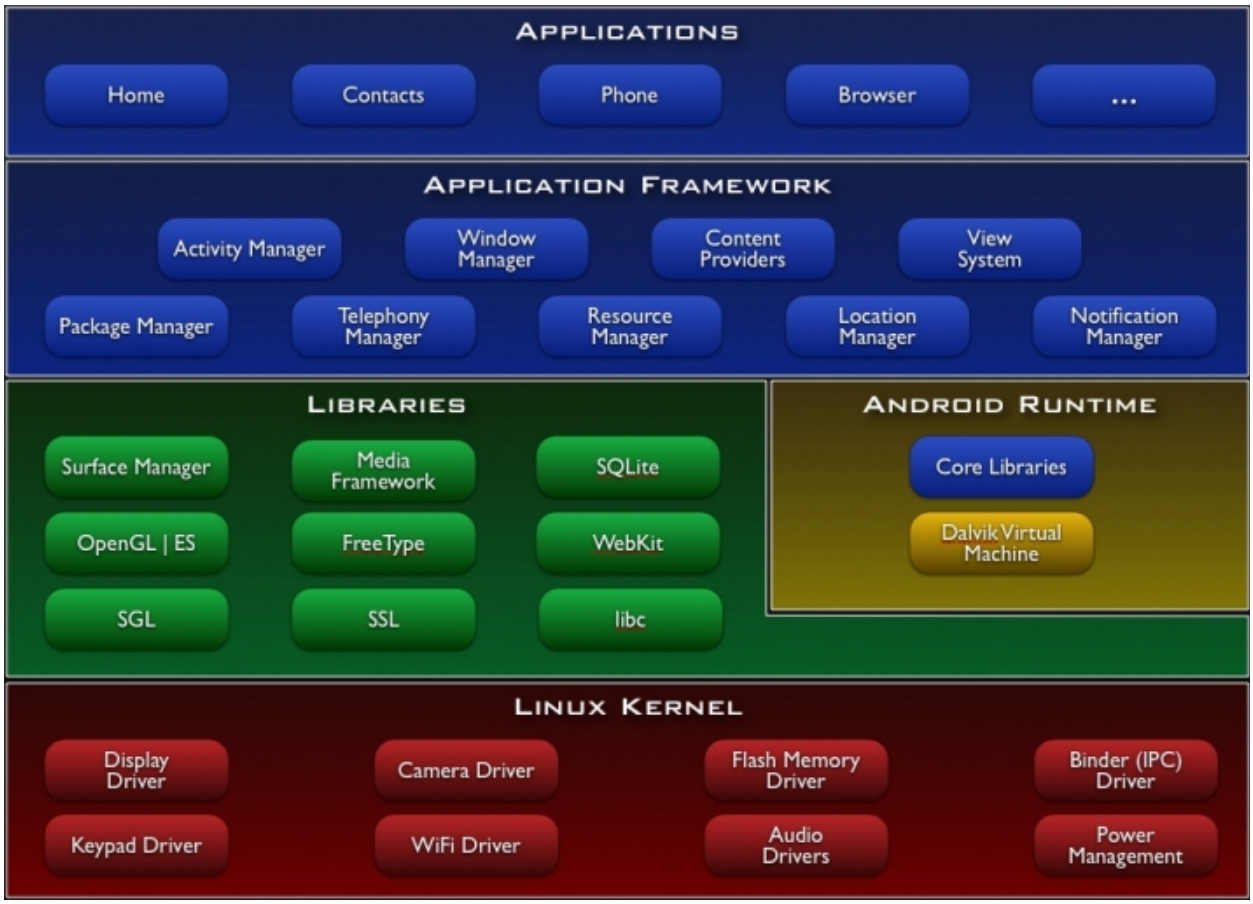

Figura 4. Arquitectura del sistema Android

2.6. Códigos QR

Un código QR [12] (Quick Response - de respuesta rápida - en inglés) es un código de barras de dos dimensiones que puede escanearse con la cámara de un teléfono móvil y que puede, según del tipo que sea, redirigir a una página web, mostrar un texto, enlazar a un número de teléfono...

Fue inventado en el año 1994 por Denso, una de las compañías más grandes del grupo Toyota, y aprobado como un estándar ISO internacional en junio del año 2000.

Comparados con los código lineales, los códigos QR pueden contener aproximadamente 100 veces más información. La matriz es leída usando un sensor CCD, los datos que se leen se guardan en la memoria y, usando el software, se analiza la imagen se buscan los patrones necesarios y la posición en la que está la imagen y a partir de ahí se decodifica la información.

Por lo tanto, podemos decir que es resistente a símbolos distorsionados, por ejemplo, que no se encuentren en una superficie lisa o que se capte con una cierta inclinación. Además, implementa un error de corrección, por lo que sería posible leer un código que hubiera perdido como máximo un 30% de su información, como por ejemplo, códigos medio tapados o recortados.

Como se ha dicho antes, cada símbolo puede contener tipos de información distinta, como texto, datos numéricos, direcciones de correo electrónico, imágenes... por lo tanto, es necesario que el tipo de servicio se incluya en los datos, para así hacerle saber al teléfono lo que tiene que hacer.

Estos códigos pueden ser creados por cualquier persona ya que hay muchas páginas web con generadores a los que solamente hay que introducirles los datos que quieres en el código y obtienes el símbolo que enlaza a ellos. Para leerlos, se necesita alguna aplicación de las muchas disponibles que son capaces de leerlos.

2.6.1. ESTRUCTURA DE UN CÓDIGO QR

Estos códigos son símbolos matriciales con una estructura en rejilla en forma de cuadrado. Tiene varios patrones funcionales para que la lectura de los datos sea fácil. A continuación se van a explicar cada uno de los patrones de los que se compone un código.

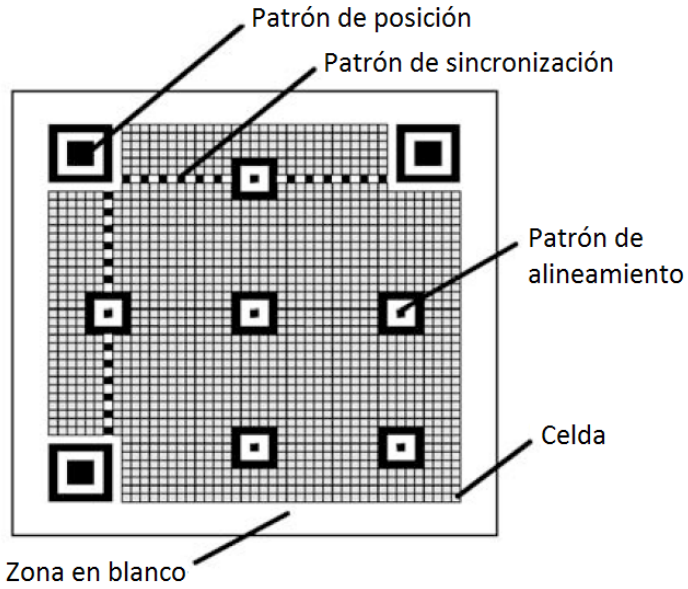

Figura 5. Estructura de un código QR

- **Patrón de posición.** Este patrón sirve para detectar la posición del código. Como este patrón está colocado en tres esquinas del símbolo, la posición, el tamaño y el ángulo del mismo pueden ser detectados. Este patrón consiste en una estructura que puede ser detectada en todas las direcciones.

- **Patrón de alineamiento.** Patrón para corregir la distorsión del código. Es muy efectivo para corregir las distorsiones no lineales. Para corregir la distorsión, miraremos la coordenada central de este patrón. Para este propósito, se coloca una celda negra en el centro para que sea más fácil detectar la coordenada central.

- **Patrón de sincronización.** Patrón para identificar la coordenada central de cada celda en el código con cuadros negros y blancos colocados alternativamente. Se usa para corregir la coordenada central de las celdas de datos cuando el símbolo está distorsionado o cuando hay un error en el tono de la celda. Se coloca tanto vertical como horizontalmente.

- **Zona en blanco.** Se necesita un margen para leer el código. Esta zona facilita la detección del símbolo dentro de la imagen leída. Debe tener un espacio mínimo de cuatro celdas.

- **Área de datos.** Los datos serán codificados en el área de datos, que es lo que representa la zona gris de la imagen. La información será codificada en símbolos binarios 1 y 0 que se convertirán en celdas negras y blancas y serán colocadas en el cuadrado. Este área también incorporará códigos de corrección de errores.

2.7. Java

La tecnología Java [13] es tanto un lenguaje de programación como una plataforma.

El lenguaje de programación es un lenguaje de alto nivel que puede caracterizarse por este conjunto de palabras: simple, orientado a objetos, conocedor de las redes, interpretado, robusto seguro, neutro en la arquitectura, portable, de alto rendimiento, multihilo y dinámico.

Una plataforma es el entorno en el que un programa se ejecuta. Algunas plataformas pueden ser Windows, Linux o Mac. La mayoría de las plataformas pueden describirse como la combinación del sistema operativo y el hardware interno. La plataforma Java difiere del resto en que sólo tiene un software que se ejecuta sobre otras plataformas hardware.

2.7.1. CARACTERÍSTICAS

Veamos qué significa cada uno de los términos que definen a esta tecnología.

- **Simple.** Puede ser programado fácilmente sin mucha formación. Se diseñó lo más parecido a C++ que fuera posible para hacerlo más comprensible.

Java omite muchos rasgos de C++ que no se usan apenas y que daban más problemas que beneficios. Se añadió el recolector automático de basura, para simplificar la tarea de programar, pero a la vez haciendo el sistema más complicado. Una fuente de problemas común en C y C++ es el manejo del almacenamiento: la asignación y liberación de memoria. Al hacerlo automático con Java la tarea de programar se hace más fácil.

Otro aspecto de simpleza es que es pequeño. Una de las metas de Java es la construcción de software que pueda ejecutarse en ordenadores pequeños.

- **Orientado a objetos.** Esta técnica consiste en centrar el diseño en los datos. Un objeto es un conjunto software de estado y comportamiento. Los objetos software son usados a veces para modelar objetos del mundo real que se encuentran todos los días. Una clase es un

prototipo a partir del cual son creados los objetos. Modelan el estado y comportamiento de un objeto.

- **Conocedor de las redes.** Tiene una biblioteca extensa de procedimientos para enfrentarse fácilmente a protocolos TCP/IP como HTTP y FTP. Esto hace que crear conexiones de red sea mucho más fácil que en C o C++. Las aplicaciones escritas en Java pueden abrir y acceder objetos de un extremo a otro de la red a través de una URL con la misma facilidad con la que se accede a un archivo local.

- **Robusto.** Java pretende ser utilizado para escribir programas que sean fiables por diversas razones. Pone mucho énfasis en anticiparse en la búsqueda de posibles problemas, para después hacer una segunda comprobación dinámica en ejecución que elimina situaciones que son propensas a errores.

El modelo de manejo de memoria es extremadamente simple: los objetos se crean con el operador `new`. No hay tipos de datos que representen punteros, ni aritmética de punteros, pero sí tiene un recolector automático de basura. Este modelo tan simple elimina clases enteras de errores de programación que ocurren en C y C++. Se puede programar con la confianza de que el sistema encontrará los errores rápidamente y que los grandes problemas no se mantendrán inactivos hasta alguna de las diversas ejecuciones.

- **Seguro.** Java está destinado para uso en entornos de red por lo que se ha puesto mucho empeño en su seguridad. Java posibilita la construcción de sistemas que no se pueden acceder desde fuera. En un entorno de red las aplicaciones escritas en Java están seguras de la intrusión por código no autorizado que intente crear virus o invadir archivos del sistema.

- **Neutro en la arquitectura.** Java fue diseñado para sostener aplicaciones de red. En general, las redes están formadas por una gran variedad de sistemas con distintas especificaciones. Para permitir a una aplicación Java que se ejecute en cualquier parte de la red, el compilador genera un formato de archivo neutro independiente de las arquitecturas, lo que significa que el código compilado es ejecutable en muchos procesadores, siempre que en el sistema se encuentre el entorno de ejecución de Java.

Esto se hace generando instrucciones en código byte que no tienen nada que ver con una arquitectura concreta, si no que son diseñados para ser interpretados fácilmente en cualquier máquina y fácilmente traducidos en código nativo en el momento.

- **Portable.** La definición de neutro en la arquitectura ya representa una parte de portabilidad pero hay más razones. Al contrario que en C y C++ no hay aspectos dependientes

de la implementación, por ejemplo, un *int* siempre será lo mismo en cualquier sistema en el que ejecutemos la aplicación.

Las librerías que son parte del sistema definen interfaces portables, hay clases abstractas generales e implementaciones de ellas para cada sistema.

- **Interpretado.** El intérprete de Java puede ejecutar código byte directamente en cualquier ordenador al que se haya llevado el entorno de ejecución. Esto es beneficioso para obtener ciclos de desarrollo mucho más rápidos.

- **De alto rendimiento.** La plataforma Java alcanza un rendimiento superior adoptando un esquema en el que el intérprete puede ejecutarse a máxima velocidad sin comprobar el entorno de ejecución. El recolector automático de basura se ejecuta como un hilo en segundo plano de poca prioridad, asegurando una alta probabilidad de que la memoria está libre cuando es requerida, llevando a un mejor rendimiento.

- **Multihilo.** Es una manera de construir aplicaciones con muchos hilos de ejecución. Desafortunadamente, escribir programas que tratan con muchas cosas ocurriendo al mismo tiempo puede ser más difícil que escribir un programa con un único hilo.

Algunos beneficios de esta característica son: mejores respuestas interactivas y comportamiento en tiempo real. Esto está limitado por la plataforma inferior, porque el entorno de ejecución de Java tiene un buen comportamiento en tiempo real, pero como se ejecutan dentro de otros sistemas como Unix, Windows o Mac, la respuesta puede ser más limitada que la original. Los programas multihilo tienen como resultado una alta interactividad para el usuario final.

- **Dinámico.** Java es un lenguaje más dinámico que C o C++. Fue diseñado para adaptarse e un entorno en desarrollo. Por ejemplo, si una librería añade un método nuevo, a los programas que usaban esa librería que ya estaban implementados no les afecta en nada.

2.7.2. ¿POR QUÉ UTILIZAMOS JAVA?

Para resumir todo lo explicado anteriormente, podemos decir que se ha utilizado el lenguaje de programación Java en este proyecto y se sigue usando en todo el mundo porque es más sencillo que C y C++, es orientado a objetos, podemos ejecutarlo en cualquier máquina y tiene infinidad de posibilidades de desarrollo.

3. ARQUITECTURA DEL SISTEMA

3.1. Descripción del problema propuesto

Tenemos un escenario en el que queremos centralizar todos los dispositivos de manejo de electrodomésticos en un único mando. Para mayor comodidad, en vez de escoger un mando neutro que realice esta función, vamos a utilizar un dispositivo móvil ya que de esta manera no se añade ningún otro elemento que podamos no tener en casa ya.

Como hemos visto anteriormente, el sistema operativo móvil Android es el que más ha crecido en los últimos años, el que está disponible en mayor número de terminales y, por lo tanto el sistema que tiene una mayor cantidad de personas. Además, ofrece mayores facilidades que el resto en cuanto a la creación de aplicaciones por parte de terceros. Estas son las razones básicas por las que se ha elegido este sistema como base para este proyecto, además de que la programación en este lenguaje es bastante sencilla si se tiene una base de conocimiento en el lenguaje Java.

Para comenzar, necesitamos encontrar una manera de identificación de los diferentes aparatos electrónicos por parte del dispositivo Android. Una posible manera de hacer esto sería asignar un código a cada uno de los aparatos que vayamos a querer manejar remotamente. El código se introduciría en la aplicación donde tendríamos una tabla relacional de códigos.

Una vez identificado el aparato electrónico por el dispositivo Android, se hará una llamada para la obtención de la interfaz gráfica correspondiente a ese aparato, que se visualizará en el propio terminal. A partir de ahí existirá una comunicación entre nuestro dispositivo Android y ese electrodoméstico que estemos manejando, que se intercambiarán mensajes según el comando que pulsemos en la interfaz obtenida anteriormente.

Se han decidido tomar como código de identificación los códigos QR, ya que se pueden crear códigos distintos para cada aparato y quedarían completamente diferenciados. Como se ha visto antes, podemos ligar cada código a un tipo de información, en este caso podríamos querer enlazar el código a un texto descriptivo o a una URL.

Como estamos en un escenario M2M en el que los elementos están conectados a Internet, va a ser posible enviar los comandos de funcionamiento mediante algún protocolo que se utilice en la web, por lo que lo más simple sería enlazar el código QR a una URL que identifique a algún servidor web en el que esté alojada la réplica del mando a utilizar con el electrodoméstico seleccionado. Una vez obtengamos este elemento en el dispositivo móvil, haremos que nuestro servidor y el aparato se comuniquen mediante envío de mensajes.

3.2. Requisitos del sistema

En este apartado de la memoria se van a enumerar los requisitos que se deben cumplir para poder hacer un buen uso de esta aplicación.

- Se deberá contar con un dispositivo móvil con sistema operativo Android.

- Este dispositivo tendrá que tener una cámara para posibilitar la interacción con los códigos QR utilizados. Por lo tanto, no es imprescindible que sea un teléfono móvil, si no que también se pueden emplear tabletas.

- Se necesitará una conexión a Internet en el dispositivo móvil.

- Al estar en un escenario M2M, los aparatos electrónicos con los que vayamos a usar el sistema también deberán tener una conexión a Internet.

- Una alternativa a esto sería utilizar la aplicación con una aplicación de ordenador que, por supuesto, deberá estar conectado a la red.

- El sistema deberá ser fácil de utilizar para cualquier tipo de usuario, sin que requiera ningún tipo de entrenamiento.

- La aplicación tendrá que tener una tiempo de espera adaptado a lo que se podría esperar de un sistema parecido, como un mando de televisión común.

- Se minimizarán los errores que se pudieran producir a lo largo de la comunicación

3.3. Partes del sistema

En este apartado se procederá a explicar de qué partes se tendrá que componer el sistema según el problema propuesto y los requisitos.

3.3.1. APLICACIÓN ANDROID

El punto de entrada al programa es una aplicación Android. Esta aplicación tendrá que consistir en un lector de códigos QR que pueda leer el símbolo y además actuar en consecuencia con los datos que contiene.

Como los códigos van a llevar una URL, lo que se tendrá que hacer con ella será abrir un navegador y obtener los controles que lleva asociada la dirección para mostrarlos en el dispositivo móvil.

3.3.2. SERVIDOR WEB

El servidor web es al que apuntará la URL del código QR, deberá devolver los controles hacia el móvil y además llevar a cabo el intercambio de mensajes hacia el aparato electrónico según lo que haya pulsado el usuario.

Deberemos implementar esta comunicación según algún protocolo de comunicación como TCP/IP, UDP o incluso HTTP. Los dos primeros se encuentran en el nivel de transporte y el último corresponde al nivel de aplicación. Más adelante veremos cual podría ser el más apropiado para alcanzar el objetivo propuesto.

3.3.3. APLICACIÓN RECEPTORA

Se puede llamar aplicación receptora al dispositivo electrónico que vamos a querer manejar con la aplicación móvil. En este caso, se ha decidido implementar una aplicación java de escritorio para simular el comportamiento de un aparato electrónico real.

En este caso, vamos a tener dos simuladores: uno que representa el funcionamiento de una bombilla y otro que hace la función de un reproductor de vídeo.

4. DESARROLLO DE LA APLICACIÓN

4.1. Entorno utilizado

En este apartado se van a explicar las herramientas utilizadas para el desarrollo de la parte práctica de este proyecto.

4.1.1. ECLIPSE IDE

Eclipse [14],[15] es el entorno de desarrollo más utilizado para programar en Java. Fue creado por una comunidad de código abierto y se usa en muchas áreas diferentes. La comunidad de eclipse tiene más de 200 proyectos que cubren distintos aspectos del desarrollo de software.

Los proyectos eclipse están dirigidos por la Fundación Eclipse, que es una corporación sin ánimo de lucro soportada por los miembros que aloja los proyectos y ayuda a su desarrollo.

Los programas de eclipse tienen una licencia pública que permite que puedan ser usados, modificados, copiados y distribuidos de forma gratuita.

Eclipse requiere que se tenga instalada alguna versión del entorno de ejecución de Java. Además, el programa contiene su propio compilador Java.

La versión que se ha utilizado en este proyecto es Eclipse EE Indigo.

4.1.2. TOMCAT

Tomcat [16] es un servidor web con soporte para servlets y JSPs. Es una implementación software de código abierto de las tecnologías de Servlets de Java. Esto se combina con el servidor Apache para la ejecución de aplicaciones web desarrolladas.

Para ejecutar una aplicación con tomcat, deberemos iniciar el servidor, exportar como un archivo comprimido .war nuestra aplicación y ejecutarla desde un navegador normal.

4.1.3. OPENSHIFT

Para probar el proyecto en un entorno real, necesitaba encontrar un servidor que pudiera alojar mi aplicación. Como contiene servlets y es necesario un servidor Tomcat, no es válido cualquier servidor. Después de hacer una búsqueda, encontré Openshift.

Openshift [17] es una plataforma que permite el alojamiento gratuito de las aplicaciones de sus usuarios. Su función principal es que los usuarios puedan diseñar, desarrollar y probar sus aplicaciones en un entorno real de forma gratuita.

Ofrece aplicaciones para su instalación directa como entornos para PHP, Ruby, WordPress, etc., o incluso un entorno llamado *Do It Yourself* (Hágalo usted mismo), para la instalación de cualquier ámbito que se necesite y no venga por defecto. Este último es el que se ha utilizado en el proyecto y en el que se ha instalado el servidor Tomcat.

También se ha utilizado una instalación de MySQL que permite la creación de bases de datos para contar con un almacenamiento persistente.

4.1.4. JERSEY

Jersey [18] es un conjunto de librerías de código abierto extensión de Java para ayudar en la creación de servicios web de tipo REST. La palabra REST proviene del inglés Representative State Transfer o Transferencia de Estado Representacional y define un conjunto de métodos arquitectónicos gracias a los que se pueden diseñar servicios web que se centran en los recursos del sistema.

Cualquier implementación de un servicio web REST consta de cuatro principios de diseño:

- Utiliza los métodos HTTP de forma explícita.

En estos servicios web, se usan los métodos HTTP como métodos explícitos en la clase. Se realiza una asociación entre las operaciones CRUD (CREATE, READ, UPDATE, DELETE), que se usan para el almacenamiento persistente, y las HTTP (POST, GET, PUT, DELETE).

- No mantiene ningún estado.

Cualquier petición dirigida a un servicio web REST, se procesará sin almacenar nada en ese mismo servicio. Esto hace que este tipo de servicios tengan mucho mejor rendimiento y simplifica su diseño e implementación. Si es necesario almacenar alguna sesión de usuario o alguna variable, se tendrá que utilizar otro método como una base de datos o un archivo de texto.

- Se utilizan URIs con forma de directorios.

La estructura de las URIs de un servicio web REST es jerárquica con una ruta raíz y subrutas que exponen las áreas y recursos del servicio. Un posible ejemplo podría ser:

```
http://www.servicio.com/discusion/temas/{tema}
```

La raíz, discusion tiene un nodo temas después, para acceder a un tema en concreto de este servicio web, tendríamos que escribirlo en la última parte de la URI, después de temas

- Transfiere XML, JavaScript Object Notation (JSON), o ambos.

El último principio tiene que ver con el formato de los datos que el cliente y el servicio intercambian en la petición HTTP. Puede solicitarse la devolución de los datos en un formato específico como XML utilizando la cabecera HTTP.

En este proyecto se ha utilizado esta forma de implementar servicios web porque es mucho más intuitiva y fácil que las otras opciones como puede ser SOAP.

- Los recursos son los extremos de una interfaz REST. Cada recurso tiene su propia URI.

4.1.5. VLCJ

El proyecto vlcj es un proyecto de código abierto que proporciona librerías para crear proyectos java incluyendo el reproductor multimedia VideoLAN.

El lenguaje Java, tiene un conjunto de librerías que soportan reproducción multimedia, el el entorno JMF. Para este proyecto, decidí buscar una alternativa, ya que, JMF lleva mucho tiempo sin ser actualizado y no funciona bien con algunos formatos. A utilizar estas librerías, llegamos a una solución muy parecida a lo que podría llegarse con JMF pero sin ningún problema añadido de formato o versión.

4.2. Aplicación Android

Para el desarrollo de la aplicación Android, que será el punto de entrada de todo el sistema, necesitamos un lector de códigos QR.

Para conseguir esto vamos a usar ZXing [21], una biblioteca Java de procesamiento de imágenes multiformato de código abierto. El propósito principal de estas bibliotecas es utilizar la cámara de los dispositivos móviles para decodificar códigos de cualquier tipo en el propio dispositivo, sin comunicación con ningún servidor.

Este proyecto ofrece una forma de integración de su sistema en cualquier aplicación Android que se desarrolle, llamada "Escaneo por medio de Intent" [22]. Consiste en una clase que incluimos en nuestro proyecto y a la que hay que llamar para que lance el lector de códigos.

Este lector no está totalmente integrado en nuestra aplicación si no que hace falta instalar una aplicación externa. Está aplicación se llama BarCode Scanner y es a la que llama la clase que incluimos en nuestro proyecto.

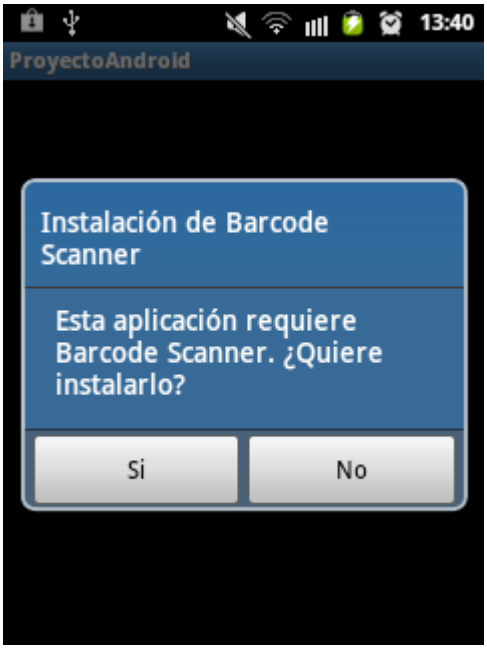

Figura 6. Aviso de instalación de Barcode Scanner.

Si no se tiene instalada esta aplicación lectora la primera vez que ejecutamos el programa, aparecerá un aviso indicando que se va a proceder a instalar y que pide permiso al usuario para hacerlo.

Sería posible también hacer una integración completa para evitar este paso y no tener otra aplicación instalada a parte, pero realmente no implica tanta molestia, ya que el proceso es automático, la aplicación es gratuita y sólo ocupa 412KB.

La pantalla principal de nuestra aplicación consta de un mensaje que dice "Pulsa el botón para empezar a capturar tu código QR" y un botón con la leyenda "Capturar". Cuando lo pulsamos, se lanza el intent de la aplicación BarCode Scanner y procedemos a la lectura del código.

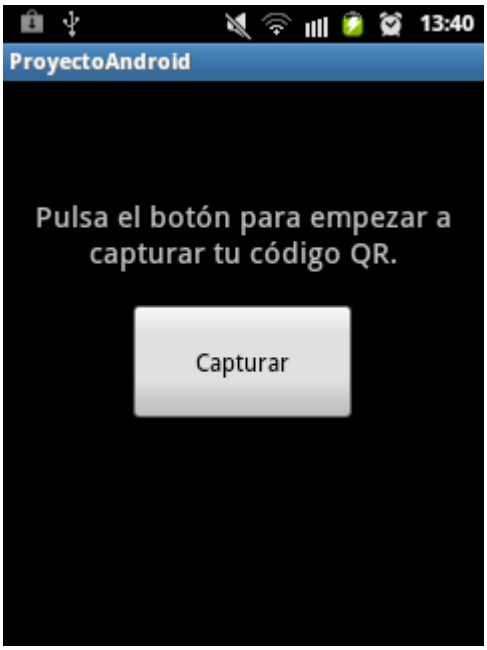

Figura 7. Pantalla principal de la aplicación.

Al pulsar el botón, pasamos a la vista de la cámara, que contiene una línea roja horizontal en el medio de la pantalla. El dispositivo deberá estar en posición horizontal. Centramos el código en el centro de la pantalla para su lectura.

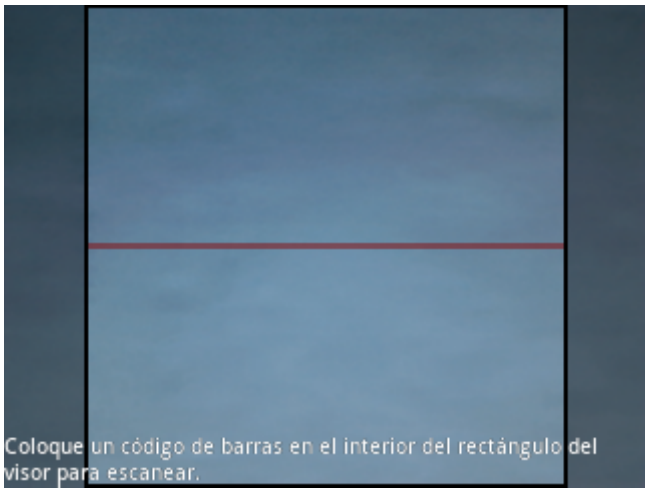

Figura 8. Pantalla de captura.

Una vez se detecta el código que enfocamos con el dispositivo aparece el mensaje "URL encontrada".

Figura 9. Pantalla de URL encontrada.

La aplicación integrada es capaz de leer un gran número de códigos diferentes, pero en este caso se ha limitado la funcionalidad para que sólo lea códigos QR que enlazan a una web.

Una vez leído el código, la URL que contenía se devuelve al intent principal de la aplicación y se lanza un navegador con esa URL obtenida que, una vez cargada, será la que contenga los controles del aparato del que hayamos leído el código.

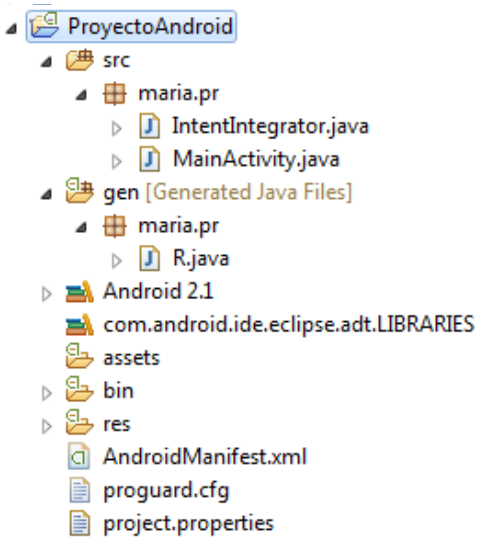

Figura 10. Estructura del proyecto en Eclipse

Según vemos en esta imagen, este proyecto consta de dos clases: IntentIntegrator.java, que es la clase que nos ofrece la funcionalidad de lectura de códigos, y MainActivity.java, que es nuestra clase principal.

A continuación se van a explicar los métodos de los que se compone la actividad principal del programa.

- `onCreate()`: Método por el que comienza la ejecución en toda actividad de Android. Crea la interfaz gráfica de la pantalla.

- `onClick()`: Recibe los eventos de pulsado sobre el botón. Lanza la aplicación de lectura de códigos QR.

- `lanzaNavegador()`: Abre un navegador con la URL obtenida del código.

- `onActivityResult()`: Recibe el resultado de la actividad de lectura de códigos, que en este caso será una URL, el formato del código y otros datos sobre él. A partir de este método llamamos a `lanzaNavegador()`.

4.3. Solución 1: Servlets y Sockets

Para la primera solución implementada, se utilizan íntegramente Servlets [23] y comunicación UDP. Para ello, creamos un proyecto web dinámico en el entorno Eclipse.

Los servlets son una clase de lenguaje de programación Java que se usan para ampliar las capacidades de los servidores que alojan aplicaciones accedidas por medio de un modelo de petición y respuesta. Todos los servlets deberán implementar la interfaz `Servlet`, que define los métodos de su ciclo de vida. Si se quiere implementar un servlet básico, se puede extender la clase `HttpServlet` del API de Java que ofrece los métodos `doGet` y `doPost` para manejar las peticiones HTTP que se reciban.

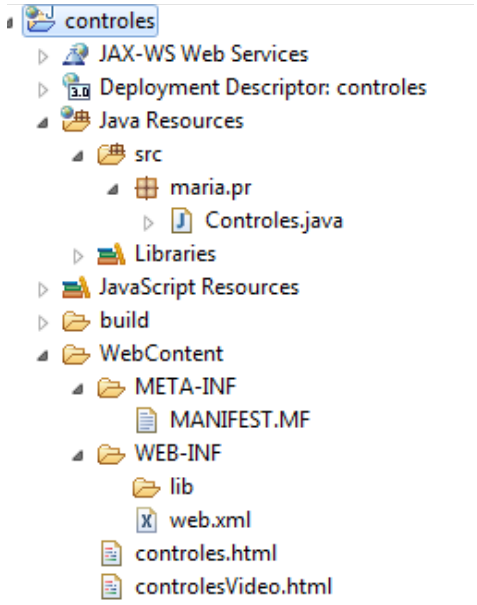

Figura 11. Estructura del proyecto

En este caso, el servlet implementado está formado por tres archivos principales: La clase Controles.java, que es el servlet en sí mismo, el que tiene los métodos que manejan las peticiones; el archivo controles.html que es el que envía peticiones hacia el servlet por medio de una interfaz de botones; y el archivo web.xml que sirve para mapear el archivo .java al directorio al que enviamos la petición.

Como se puede ver en la figura, tenemos dos archivos html, esto es porque deberemos tener tantos archivos HTML como códigos QR referentes a electrodomésticos. Cada uno tiene la interfaz de usuario de un aparato distinto con sus botones correspondientes.

4.3.1 AJAX

Se va a proceder a explicar la tecnología AJAX ya que se ha utilizado en esta parte del proyecto y es necesario para poder seguir con la explicación del desarrollo de la aplicación.

AJAX (Asynchronous JavaScript and XML) [24] es una tecnología para crear páginas web dinámicas que permite que las páginas web se actualicen de manera asíncrona intercambiando pequeñas cantidades de datos con el servidor. Es posible actualizar partes de la página sin volver a cargarla entera. Las páginas que no utilizan AJAX deben volver a cargar la página entera si el contenido cambia.

Para hacer una petición al servidor con AJAX necesitamos un objeto XMLHttpRequest que se usa para intercambiar datos con el servidor. Para enviar una petición al servidor usamos los métodos `open()` y `send()` de este objeto.

El método `open()` tiene un parámetro que representa el tipo de petición, que puede ser GET O POST. GET se usará en la mayoría de los casos, pero hay que usar peticiones POST cuando enviamos grandes cantidades de datos al servidor, ya que POST no tiene tamaño límite, o cuando enviamos datos introducidos por el usuario, porque se ocultarán los datos introducidos por el usuario. En nuestro caso hemos utilizado una petición POST, por las razones enumeradas.

Además, podemos tener una función que se ejecute cuando la respuesta esté lista. El evento onreadystatechange se dispara cada vez que la variable readyState cambia. Cuando readyState es igual a 4 quiere decir que la petición ha finalizado y la respuesta está disponible.

Si se necesita más de una petición AJAX, se crea solamente una función que crea el objeto XMLHttpRequest y se la llama con diferentes parámetros.

4.3.2. CONTROLES.HTML

La entrada del servlet, donde se realizan las peticiones, es el archivo controles.html que es al que va a enlazar el código QR explicado anteriormente. Este archivo html, está formado por una interfaz con botones y un código JavaScript que controla su pulsación.

Figura 12. Interfaz de bombilla.

Cada botón tiene una etiqueta *name* que identifica el control que representa. Cuando pulsamos alguno de los botones, esta etiqueta se envía a la función de control de eventos de pulsado de un botón en JavaScript: `onClick`. Junto con JavaScript, vamos a utilizar la tecnología AJAX para poder hacer peticiones asíncronas.

```
function onClick(nombreBoton){
        /*Al pulsar el botón obtengo el comando del botón (nombre) y
        se lo paso al dispositivo en un post*/

        var url = "pulsado";
        var xmlhttp = new XMLHttpRequest();

        xmlhttp.open("POST", url, true); //true: Comunicación asíncrona
        xmlhttp.setRequestHeader("Content-type",
                                 "application/x-www-form-urlencoded");
        xmlhttp.send("boton="+nombreBoton);
 }
```

Código 1. JavaScript de controles.html

La variable url es la dirección a la que vamos a hacer la petición. En este caso se utiliza una url relativa a la que ya tenemos, solo se añadiría la palabra pulsado al final de ella. Este parámetro podría ser cualquier tipo de archivo: xml, txt...

En este caso, no se espera ningún tipo de respuesta por parte del servidor, porque como no es necesaria, no se ha implementado. Podemos decir que la comunicación es en un único sentido.

4.3.2. WEB.XML

Este archivo tiene la función de hacer la asociación de la clase Java que compone el servlet con la dirección a la que hacemos la petición. De esta manera, al hacer una petición, será la clase Java la que la reciba.

```
<?xml version="1.0" encoding="utf-8"?>
<web-app>
   <servlet>
      <servlet-name>controlservlet</servlet-name>
      <servlet-class>maria.pr.Controles</servlet-class>
    </servlet>

   <servlet-mapping>
      <servlet-name>controlservlet</servlet-name>
      <url-pattern>/pulsado</url-pattern>
   </servlet-mapping>
</web-app>
```

Código 2. Archivo web.xml

Podemos ver que se mapea la clase Controles.java a la url `/pulsado` a la que enviábamos las peticiones dentro del JavaScript.

Por lo tanto, en un entorno local, accederemos a los controles con la URL:

`http://localhost:8080/controles/controles.html`

Y la petición se hará a: `http://localhost:8080/controles/pulsado`

4.3.3. CONTROLES.JAVA

Clase que extiende de HttpServlet y que, por lo tanto, implementa los métodos de las peticiones doPost() y doGet(). Lo que hacemos dentro de esta clase es obtener el nombre del botón que hemos pulsado y que se ha enviado en el JavaScript.

Además, tenemos un método enviar al que le pasamos como parámetro el nombre del botón una vez obtenido y que lo envía por medio de sockets a una aplicación simuladora creada.

Se han utilizado sockets en esta primera solución de la aplicación, porque su implementación es sencilla. Es posible también, elegir entre sockets TCP o UDP. Veamos las características de cada uno de ellos.

En los sockets TCP [25] es necesario establecer una conexión. El servidor espera hasta que se conecta el cliente. Es un protocolo fiable y garantiza que todos los mensajes van a llegar bien. Aunque la fiabilidad es una característica deseable, en este caso no tenemos una conexión en la que el cliente se va a conectar para pedir la información, porque ésta viene por otro lado.

En los sockets UDP no se establece conexión. El servidor y el cliente se ponen a escuchar en un puerto y en cualquier momento cualquiera de ellos pueden enviar un mensaje al otro. Además, este protocolo sólo garantiza que si el mensaje llega, llega bien, sin fallos. Para la aplicación que se está desarrollando, como el mensaje enviado es pequeño y, por lo tanto tarda poco en llegar, no nos importa que este protocolo tenga errores en el envío, porque no se van a notar y se puede volver a enviar el mensaje sin causar molestias.

Por todo lo explicado, se han elegido los sockets UDP para que el servlet web envíe el nombre del botón hacia la aplicación simuladora.

La clase que estamos tratando, Controles.java, será el servidor que envíe la información hacia el cliente. Para ello tenemos que instanciar la clase DatagramSocket que creará el canal

de comunicación y también la clase DatagramPacket que es la que se va a enviar o recibir como paquete, el objeto que creemos de esta clase lleva dentro un array de bytes que será la información que enviamos, y los datos sobre el destino, el puerto y la IP.

```
public void enviar(String nombreBoton){
	byte[] dato = nombreBoton.getBytes(); //Paso el nombre a bytes para enviarlo
	DatagramSocket clientSocket = null;

	try {
		clientSocket = new DatagramSocket();
		DatagramPacket paquete = new DatagramPacket(dato, dato.length,
				InetAddress.getByName(HOST_SERVIDOR), PUERTO_SERVIDOR);

		clientSocket.send(paquete);

	} catch (Exception e) {
		e.printStackTrace();
	} finally {
		if(clientSocket.isConnected()){
			clientSocket.disconnect();
		}
		if(!clientSocket.isClosed()){
			clientSocket.close();
		}
	}
}
```

Código 3. Socket de envío

4.3.4. SIMULADOR BOMBILLA

Para probar este programa, se ha creado una aplicación Java que simula el funcionamiento de una bombilla.

La clase SimuladorBombilla.java crea una interfaz gráfica compuesta por una imagen de una bombilla que está apagada cuando se ejecuta la aplicación y va cambiando de estado según que mensajes le lleguen.

A continuación se van a explicar los métodos de los que se compone esta clase.

- `SimuladorBombilla()`: Es el constructor, inicializa los componentes de la interfaz gráfica.

- `creaInterfaz()`: Añade los elementos de la interfaz a un JFrame y lo hace visible.

- `escucha()`: Es el cliente UDP. Escucha paquetes UDP un puerto y una vez recibe uno, lo decodifica y se lo pasa al método `realizaAccion()`.

- `realizaAccion()`: Según el dato que le llegue cambia la interfaz gráfica.

- `main()`: Método que comienza la ejecución de la aplicación. Llama a los métodos `creaInterfaz()` y `escucha()`.

Para poder recibir mensajes, el socket cliente tiene que instanciar la clase DatagramSocket indicando el puerto en el que escucha, que es el mismo al que envía el servidor y también instanciar la clase DatagramPacket con un paquete vacío, que es donde se introducirá el paquete que llega. Una vez que llega, como está en bytes se extrae la información a un String y se le pasa al siguiente método.

Figura 13. Estados de la aplicación simuladora

4.4. Solución 2: Servicio Web

En la segunda solución implementada, se ha desarrollado una aplicación basada en servicios web.

Antes de comenzar con los detalles del desarrollo, voy a hacer una introducción breve a los servicios web, qué son y cómo funcionan.

Un servicio web es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones software, desarrolladas en lenguajes de programación diferentes y ejecutadas sobre cualquier plataforma pueden utilizar los servicios web para intercambiar datos en redes como Internet.

En este caso, el servicio web que se ha implementado es de tipo REST utilizando el API Jersey de Java, que ya se ha explicado anteriormente. Se ha elegido esta forma de implementación porque es muy sencilla e intuitiva y además, debido a las características de los servicios REST, la comunicación entre las partes es más rápida.

Esta solución tiene 3 partes muy diferenciadas: El servicio web, un cliente web y un cliente simulador.

El cliente web es el que contiene la interfaz de botones a la que se llama desde el dispositivo Android, el cliente simulador es el que representa el aparato electrónico y el servicio web es el que hace posible la comunicación entre todos los elementos de la red. A continuación se explicará cada uno de ellos en detalle.

4.4.1. SERVICIO WEB

Este servicio web está formado por una clase Java llamada WSControles. Su función es recibir las peticiones de los dos clientes y responderlos. Toda la comunicación entre ellos es HTTP, es decir, está en el nivel de aplicación.

HTTP (Hyper Text Transfer Protocol) es un protocolo que se utiliza para transmitir información en la web. Se denomina *sin estado* porque cada petición se ejecuta independientemente, sin conocer ninguna de las peticiones ejecutadas anteriormente. También es un protocolo sin conexión lo que quiere decir que no hace falta establecer una conexión entre las dos partes que se van a comunicar si no que se envían los mensajes directamente.

A continuación se explicarán los métodos de los que se compone este servicio web.

- `guardaEstado()`: A este método le llega como parámetro el nombre del botón que hemos pulsado en la interfaz web y se lo pasa a set`Nombre()` para guardarlo.

- set`Nombre()`: Guarda el nombre que le pasan en una base de datos.

- get`Nombre()`: Obtiene el nombre de la base de datos.

- `leeEstado()`: Este método es llamado por el simulador para saber que estado tienen los botones de la interfaz. Se llama a su vez a `getNombre()` para leer el nombre guardado y devolverlo al simulador.

Los métodos `guardaEstado()` y `leeEstado()` son los métodos reales del servicio web, los otros dos se usan para hacer funciones dentro de los métodos principales. El método `guardaEstado()` se asocia a una petición GET y `leeEstado()` a una petición POST.

Normalmente un servicio web solamente tiene una URI asociada a él, pero como estamos haciendo uso de los servicios tipo REST [26] [27], tenemos URIs personalizadas para cada acción a realizar. Veamos como es el archivo de mapeado web.xml para poder explicar esto.

```
<display-name>WSControles</display-name>
<servlet>
    <servlet-name>Controles</servlet-name>
    <servlet-class>
        com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>maria.pr</param-value>
    </init-param>
</servlet>
<servlet-mapping>
        <servlet-name>Controles</servlet-name>
        <url-pattern>/controles/*</url-pattern>
</servlet-mapping>
```

Código 4. web.xml

Sólo se ha extraído la parte importante de este código. Vamos a ver lo que significa cada parte.

- `<display-name>`: se refiere al nombre del proyecto y será el primer directorio al que habrá que acceder para hacer una petición al servicio.

- `<servlet-name>`: Debe ser exactamente igual que el que aparece más abajo, solo se utiliza para realizar la asignación entre el servicio y la clase .java.

- `<servlet-class>`: Es la clase contenedora de servlets de las librerías de Jersey que estamos usando.

- `<param-name>`: Es el nombre del parámetro de la clase contenedora de servlets. Como estamos utilizando el paquete de Jersey, estos dos parámetros van a ser siempre igual

- `<param-value>`: El valor del parámetro anterior. El paquete donde tengamos nuestro archivo .java del servicio web.

- `<url-pattern>`: Siempre empieza con '/' y termina con '/*' y puede ser cualquier nombre sin espacios. Va a ser la segunda parte de la URL , después del nombre del servicio web.

Por lo tanto, para hacer peticiones a nuestro servicio web en un entorno local tenemos, de momento, la URL base: `http://localhost:8080/WSControles/controles`

Para acceder a algún método del web service de la clase .java, tendremos que ampliar esa url.

```
@Path("/pulsado")
public class WSControles {

	@POST
	@Path("/boton")
	public void guardaEstado(String n){
		setNombre(n);
	}
	...
}
```

Código 5. WSControles.java

Este es un extracto de código de la clase WSControles.java. Podemos ver que la clase tiene un parámetro `@Path` que significa que es una clase recurso, y se puede acceder a ella por ese directorio. Además, el método `guardaEstado()` tiene otro parámetro `@Path` que indica que para llamar a este método tendremos que hacerlo añadiendo ese directorio a la URI que ya teníamos. También se puede ver que a este método se accede con una petición POST.

Por todo esto, nuestra URL completa para hacer una petición POST a este método es:

`http://localhost:8080/WSControles/controles/pulsado/boton`

Como es una petición POST, el nombre que estamos enviando no se ve de manera explícita como sería en una petición GET.

4.4.2. CLIENTE WEB

El cliente web es, como se ha explicado antes, por el que accedemos a toda la aplicación, por medio de los botones de la interfaz gráfica.

El cliente web de esta segunda solución no tiene muchas diferencias con el de la primera. Se compone de un archivo html que tiene una interfaz de botones y un código JavaScript con una función que es la encargada de lanzar la petición al servicio web.

La única diferencia que nos encontramos es que la URL a la que hacemos la petición la tenemos que escribir completa, no podemos escribir una dirección relativa como antes porque ahora el cliente web y el servicio web se encuentran separados.

4.4.3. CLIENTE SIMULADOR

El cliente simulador es, recordando lo ya explicado, una aplicación java que simula el funcionamiento de un electrodoméstico real interactuando con la aplicación que se ha desarrollado en este proyecto.

Al igual que pasa con el cliente web, esta parte tampoco ha cambiado sustancialmente con respecto a la primera solución propuesta. La única parte diferente es que antes realizábamos el envío mediante sockets UDP y ahora todas las comunicaciones son por HTTP.

De esta manera, nos evitamos tener que enviar los mensajes a puertos concretos, lo que implicaba tener que hacer una redirección de puertos desde nuestro router que no todos los usuarios estarían preparados para llevar a cabo.

4.4.4. FUNCIONAMIENTO GENERAL

En este apartado se va a explicar el funcionamiento general de esta segunda solución como una lista de pasos diferenciados para que el entendimiento sea mayor.

- Escenario de comunicación Cliente web - Servidor web

1. El usuario pulsa una tecla en el cliente web.

2. El cliente web lanza una petición POST hacia el servidor, incluyendo el nombre del botón pulsado.

3. El servidor recibe la petición y se almacena el nombre del botón.

- Escenario de comunicación Cliente simulador - Servicio web

1. El simulador realiza una petición POST al servidor web.

2. El servidor web recibe la petición, recoge el nombre que tiene almacenado y se lo devuelve al simulador.

3. El simulador comprueba si es el mismo estado que tenía antes, si lo es no hace nada, si es distinto al anterior, cambia la interfaz gráfica en consecuencia.

4. Repite en bucle los pasos anteriores.

Como ya hemos dicho, este tipo de servicio web es *sin estado* por lo que si queremos mantener alguna variable o dato de una petición a otra deberemos guardarla en otro sitio. En este caso se ha hecho guardando la variable en una base de datos MySQL. Al haber sólo una variable que guardar, utilizar una base de datos está un poco sobre dimensionado, pero es la mejor manera de tener un almacenamiento persistente, que es lo que necesitamos.

4.4.5. BASE DE DATOS MYSQL

Como ya hemos dicho, se ha hecho uso de una base de datos para guardar los datos necesarios de la aplicación ya que, por sus características, ésta no podía guardarlos por sí sola. A continuación se va a hacer una breve introducción de lo qué es MySQL.

MySQL es un sistema de gestión de bases de datos relacionales. Es la base de datos de código abierto más popular en todo el mundo porque es gratuita y está disponible para la mayoría de plataformas. Se usa para aplicaciones web porque proporciona una buena velocidad y es muy segura. Pueden almacenar datos de todo tipo.

Una base de datos contiene tablas de datos. Una tabla se compone de filas y columnas, cada fila corresponde a un elemento de esa tabla y cada columna es una característica de todos los elementos. Por ejemplo, una tabla que contenga nombres y apellidos tendrá a cada persona en una fila y las columnas serán el nombre y los dos apellidos.

Una vez hecha esta introducción, se puede pasar a explicar lo que hace la base de datos en la aplicación desarrollada. En este caso tenemos una base de datos que se llama tomcat y una tabla nombre con solamente una columna acción. Lo que hacemos es que cada vez que se pulsa un botón de la interfaz comprobamos si está creada la tabla y si no la creamos, borramos la información que contenga la tabla de antes, si la tiene, e insertamos el nombre del botón que acabamos de pulsar.

De esta manera, cuando la aplicación simuladora llama al método `leeEstado()`, éste lo que hace es ir a la base de datos y sacar el nombre del botón que se encuentra en esos momentos en ella, que será la última acción que hayamos pulsado. Una vez le llega a la aplicación simuladora, actuará en consecuencia según la acción que le haya llegado.

Con respecto a esta configuración podemos decir que la aplicación simuladora llama al servicio web cada cierto tiempo, y si pulsamos os botones más rápido de lo que se actualiza la aplicación no se producirá un acumulamiento de acciones, si no que como sólo se guarda el último pulsado, sólo se realizará esa acción.

4.5. Simulador Vídeo

Para realizar más pruebas sobre la aplicación y su funcionamiento se ha desarrollado otra aplicación simuladora que imita lo que podría ser un reproductor de vídeo con varias funciones básicas. Para lograr esto, se ha utilizado la colección de librerías que proporciona el proyecto vlcj.

Java tiene un entorno propio para la reproducción y captación multimedia, JMF (Java Media Framework) pero su última versión es antigua y existen muchos problemas con los formatos de los archivos multimedia. Como alternativa, encontré este proyecto vlcj que es el que he utilizado finalmente en el proyecto.

Para utilizar estas librerías es necesario instalar el reproductor VideoLAN.

La interfaz de botones correspondiente al simulador de vídeo se representa en la siguiente imagen.

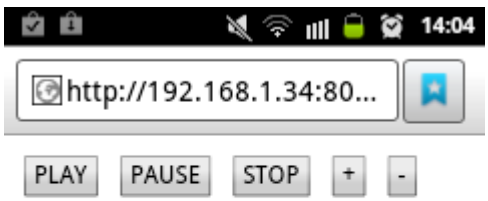

Figura 14. Interfaz de vídeo.

4.5.2. SIMULADOR

El simulador es una aplicación Java muy parecida al ya explicado anteriormente simulador de bombilla. Tiene los mismos métodos, sólo que esta vez incluimos el código para la creación del reproductor de vídeo y las opciones de la interfaz de botones son más amplias.

Los estados que tiene esta aplicación son: reproducción, pausa y parar, además de un control básico de volumen.

Este simulador está implementado tanto para la primera solución propuesta, donde se utilizan sockets, como para la segunda, en la que utilizamos la comunicación a través de un servicio web.

La interfaz del simulador es un JFrame con un color de fondo negro mientras que se encuentra parado y que contiene el vídeo cuando se encuentra en reproducción. No se ha implementado ninguna opción de elección de un archivo de vídeo, así que el archivo que se reproduce viene directamente indicado en el código.

Cuando le damos a cualquiera de los botones de la interfaz, podremos visualizar una etiqueta en la pantalla con la acción que se está llevando a cabo, para que sea más intuitivo de utilizar y sepamos lo que se está haciendo en cada momento. Después de unos segundos, el mensaje desaparecerá y solo nos quedará la pantalla de vídeo.

Como ya se ha explicado, esta aplicación pide al servicio web el nombre del último botón pulsado y, una vez obtenido, se comprueba si es igual a la última acción realizada y si lo es, no cambia nada en la pantalla. Esto es así con las tres funciones básicas, reproducir, pausar y parar, pero no con el control de volumen, porque éste si va a poder pulsarse de una manera continua para realizar un cambio de nivel del audio.

5. PRUEBAS

Para realizar las pruebas en un entorno real, se ha utilizado la plataforma openshift que ya se ha explicado anteriormente. En esta plataforma ha sido necesario instalar, para poder ejecutar el proyecto, un servidor tomcat y la aplicación para bases de datos MySQL.

Para acceder a los archivos de programación y poder poner en funcionamiento el servidor, lo hacemos desde el programa PuTTY mediante una conexión ssh cuyo host nos proporciona la plataforma openshift.

Las aplicaciones web se han subido al servidor tomcat mediante su pantalla de gestión, que se encuentra accesible desde http://tomcat-proyecto.rhcloud.com/manager. Desde eclipse se exporta el proyecto en un archivo comprimido .war y, para poder ejecutarlo en el servidor remoto tenemos que desplegarlo en la página mencionada.

La URL principal de la instalación es: http://tomcat-proyecto.rhcloud.com/ y a partir de ahí se puede acceder a las aplicaciones desarrolladas, para no tener que acceder desde un dispositivo móvil mientras que duren las pruebas.

La generación de los códigos QR se ha llevado a cabo mediante una página web [28] a la que proporcionas una URL y obtienes el código asociado a esa dirección en particular. Las direcciones vienen dadas por la URL principal de openshift, añadiendo el nombre y los parámetros de tu aplicación, como puede ser:

http://tomcat-proyecto.rhcloud.com/controles/controles.html

Al desarrollar la primera solución me encontré con un problema cuando empecé a probar el sistema desde la web, no podía acceder a mi ordenador por un puerto porque no se encontraba abierto. Después de buscar información, descubrí que es necesario hacer un port forwarding si se quiere que algún puerto del ordenador esté disponible desde alguna máquina desde fuera de la red local. Esta configuración incluye hacer cambios en el router y como no es muy básico y cambiaría para cada máquina en la que quisiéramos ejecutar el simulador, decidí que no era la mejor solución para el problema.

Sin embargo, la segunda solución, que utiliza comunicación HTTP, no necesita ninguna configuración especial ni diferente y las pruebas ejecutadas han sido todas positivas.

6. CONCLUSIONES Y LÍNEAS FUTURAS

Uno de los principales motivos que me llevó a interesarme por este proyecto era la posibilidad de programar en la plataforma para móviles Android. Creo que en este momento de auge de los dispositivos móviles inteligentes como se dijo en la introducción, es una de las tecnologías más usadas, más útiles y que más van a desarrollarse en los próximos años.

Finalmente, al contrario de lo que me esperaba, esta tecnología no ha conllevado una parte tan grande del proyecto, pero esto me ha permitido conocer una gran variedad de tecnologías diferentes con las que nunca había trabajado.

La dificultad más grande con la que me he encontrado en este proyecto ha sido, por lo tanto, el empezar casi desde cero a aprender qué eran y cómo había que desarrollar cada una de las piezas que forman este trabajo.

Se han podido cumplir todos los requisitos propuestos al inicio del trabajo y además, se ha hecho algo que no estaba planteado en un principio, como ha sido la aplicación simuladora de un reproductor de vídeo.

Además, creo que se ha conseguido que el uso de esta aplicación por parte de usuarios de cualquier tipo sea bastante intuitivo y no muy complicado de seguir.

En general, el trabajo realizado en este proyecto ha sido una experiencia muy enriquecedora ya que me ha servido para aprender, o comenzar a aprender muchas tecnologías y sistemas con los que nunca había trabajado anteriormente.

6.1. Líneas Futuras

Este proyecto tiene mucho margen de ampliación. En este apartado se explicarán los posibles nuevos desarrollos que se me ocurren serían buenas opciones para la evolución del sistema.

Lo primero en lo que creo que se debería ampliar el sistema es en la aplicación Android en sí misma, ya que en este momento no tiene muchas funciones. Una posible mejora es que se pudieran guardar los códigos QR en una base de datos, asignándoles un nombre descriptivo, para si tenemos que usar dos interfaces de una manera muy seguida, poder volver a la otra de una manera rápida.

Este proyecto está enfocado en un escenario futuro en el que todos los aparatos eléctricos tengan una dirección IP que les identifique, pero para utilizarlo ahora, podemos usar

los aparatos que ya tengan identificación, como ordenadores. Sería posible modificar el sistema para que funcionase con alguna aplicación de escritorio como las simuladoras que se han implementado para este proyecto.

Además de su utilización para manejar dispositivos ya existentes, otra posibilidad sería usar el dispositivo móvil con la interfaz obtenida, como un mando de algún videojuego. Aquí habría dos opciones, hacerlo con un juego existente o con uno que implementemos nosotros.

Estas ideas son algunas que se podrían implementar de manera sencilla a partir del desarrollo ya hecho en el trabajo.

7. RECURSOS UTILIZADOS

[1] The Evolution of Internet of Things

http://www.casaleggio.it/pubblicazioni/Focus_internet_of_things_v1.81%20-%20eng.pdf [3 septiembre 2012]

[2] M2M Technology: Challenges and Opportunities

http://www.techmahindra.com/Documents/WhitePaper/M2MTechnology_ChallengesandOpportunities_Sept10.pdf [3 septiembre 2012]

[3] Machine-to-Machine. Let your machines talk

http://www.nokia.com/NOKIA_COM_1/About_Nokia/Press/White_Papers/pdf_files/m2m-white-paper-v4.pdf [3 septiembre 2012]

[4] What Makes a Smartphone Smart?

http://cellphones.about.com/od/smartphonebasics/a/what_is_smart.htm [3 septiembre 2012]

[5] Sistemas Operativos

http://forja.rediris.es/frs/download.php/1922/SSOO-0_5_0.pdf [3 septiembre 2012]

[6] StatCounter Global Stats

http://gs.statcounter.com/ [3 septiembre 2012]

[7] Sistemas Operativos Móviles

http://www.elchecibernetico.com/compras/sistemas-operativos-moviles-ios-android-windowsphone [3 septiembre 2012]

[8] Symbian OS

http://www.developer.nokia.com/Community/Wiki/Symbian_OS [3 septiembre 2012]

[9] OS Móviles

http://jjprince.wordpress.com/2010/11/08/os-moviles-android-symbian-blackberry-y-muchos-mas/ [3 septiembre 2012]

[10] About iOs Development

http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/IPhoneOSOverview/IPhoneOSOverview.html [3 septiembre 2012]

[11] Google Android

http://www.roseindia.net/android/index.shtm [3 septiembre 2012]

[12] QR Code

http://qrbcn.com/imatgesbloc/Three_QR_Code.pdf [3 septiembre 2012]

[13] The Java Language Environment

http://www.oracle.com/technetwork/java/index-136113.html [3 septiembre 2012]

[14] Página oficial de Eclipse IDE

http://www.eclipse.org/ [3 septiembre 2012]

[15] Eclipse IDE Tutorial

http://www.vogella.com/articles/Eclipse/article.html [3 septiembre 2012]

[16] Apache Tomcat

http://tomcat.apache.org/ [3 septiembre 2012]

[17] Openshift

https://openshift.redhat.com/app/ [4 septiembre 2012]

[18] Jersey

http://jersey.java.net/ [3 septiembre 2012]

[19] RESTful Web services: The basics

http://www.ibm.com/developerworks/webservices/library/ws-restful/ [4 septiembre 2012]

[20] Página del proyecto vlcj

http://www.capricasoftware.co.uk/vlcj/ [4 septiembre 2012]

[21] Web del proyecto ZXing

http://code.google.com/p/zxing/ [3 septiembre 2012]

[22] Scanning Via Intent

http://code.google.com/p/zxing/wiki/ScanningViaIntent [3 septiembre 2012]

[23] What is a Servlet?

http://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html [3 septiembre 2012]

[24] AJAX Tutorial

http://www.w3schools.com/ajax/default.asp [4 septiembre 2012]

[25] Ejemplo de socket UDP en Java

http://www.chuidiang.com/java/sockets/udp/socket_udp.php [3 septiembre 2012]

[26] Companion Notes on RESTful Web Services with Java and Jersey

http://www.javahotchocolate.com/tutorials/restful.html [4 septiembre 2012]

[27] Restful Java with Jax-RS - Bill Burke

http://books.google.es/books?id=_jQtCL5_vAcC&printsec=frontcover [4 septiembre 2012]

[28] Web oficial de MySQL

http://www.mysql.com/ [4 septiembre 2012]

[29] Generador de códigos QR

http://qrcode.kaywa.com/ [4 septiembre 2012]