UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Control automático de Bluetooth en dispositivos móviles con Android

Proyecto Final de Carrera

Ingeniería Informática

Autor: Julián Zaragoza Bellotti

Director: Miguel Ángel Mateo Pla

19 de septiembre de 2012

Resumen

En la presente memoria, se detalla una manera de abordar el problema de la autonomía de los dispositivos (el cuál definiremos más adelante en el presente documento): El diseño e implementación de una aplicación para el sistema operativo Android que consiga economizar el consumo de la batería mediante la automatización del encendido y apagado del adaptador Bluetooth del dispositivo móvil. La aplicación se basará en un sistema de reglas (básicas o avanzadas), que combinadas entre ellas, harán que la aplicación encienda o apague el Bluetooth.

La idea es implementar un editor de reglas simple, mediante una interfaz gráfica clara e intuitiva, que mediante un asistente pueda crear reglas sencillas. También tendrá una opción de creación avanzada de reglas, basadas en un simple editor de texto y un lenguaje de definición de reglas.

Para poder construir las reglas avanzadas es necesario crear una serie de reglas básicas, las cuales deben ser creadas mediante el asistente anteriormente mencionado. Las reglas se basarán en geolocalización, dispositivos cercanos, tiempo y estado de la batería.

Las reglas se almacenarán en la memoria del dispositivo (Interna o externa), y además también existirá la opción de exportarlas a un archivo .zip, el cuál también se podrá importar posteriormente. De esta manera es posible guardar una copia de seguridad de nuestra base de reglas en caso de borrado de memoria / formateo / cambio de dispositivo.

Para poder llevar a cabo todo esto, se hará uso de guías de diseño para Android, y además, de ayudas en la interfaz gráfica que ayuden al usuario a tener una experiencia con la aplicación lo más intuitiva posible.

Palabras clave: Android, reglas, dispositivo, móvil, exportar.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Objetivos

En el Proyecto de Final de Carrera (descrito en la presente memoria), se va a realizar una aplicación para el sistema operativo Android, el cuál controle el principal problema de los dispositivos móviles avanzados hoy en día: La autonomía de la batería.

La aplicación deberá conseguir alargar la autonomía de la batería mediante un tratamiento automatizado del encendido y apagado del adaptador Bluetooth del dispositivo. Para conseguir automatizarlo, es necesario que el usuario sea capaz de transmitir a la aplicación aquellas situaciones en las que le interese que la conexión Bluetooth esté habilitada, y aquellas en las que no. Definir un conjunto de reglas que el dispositivo sea capaz de identificar, será la manera de conseguir que el usuario se comunique con la máquina, y le transmita sus preferencias a la hora de la activación y desactivación. Ese conjunto de reglas deberá estar diferenciado en dos tipos: **Reglas simples** y **reglas avanzadas**. Las reglas simples sólo podrán ser creadas mediante diálogos y *Activities* de Android (Lo cuál explicaremos más adelante en el presente escrito), los cuales permitirán la introducción de los datos mediante cuadros de texto, selectores y demás herramientas gráficas clásicas. Las reglas básicas que están establecidas como objetivo para la siguiente aplicación son:

- Regla de tiempo: Regla que basa la activación o desactivación en el paso del tiempo. Los parámetros temporales que tendrá en cuenta la regla serán el rango de fechas, el día o días de la semana, y el rango de horas a la que será efectiva. Si se dan todas las condiciones dentro de la regla, el adaptador Bluetooth se activará.

- Regla de proximidad: Regla que basa al activación o desactivación en

la proximidad o lejanía de un punto concreto en el espacio. Es decir, basa su activación en la geolocalización del dispositivo y su posición con respecto a un lugar de referencia.

- Regla de dispositivos conocidos: Regla que basa la activación o desactivación en la detección de dispositivos conocidos alrededor del dispositivo móvil. Si existe algún dispositivo que esté dentro de una lista definida por el usuario, el Bluetooth se activará.

- Regla de batería: Regla que activa o desactiva el Bluetooth según el nivel de batería o si el dispositivo se está alimentando.

El usuario podrá definir reglas basadas en las anteriores definiciones, y éste deberá decidir entre dos maneras de aplicarlas si hay contradicciones entre ellas:

- Prioridad al encendido: Ante conflicto, se dará prioridad al encendido del Bluetooth, es decir, el valor de 'true' (encendido) y 'false' (apagado) de cada una de las reglas es unido mediante 'OR' lógicas.

- Prioridad al apagado: Ante conflicto, se dará prioridad al apagado del Bluetooth, es decir, el valor de encendido/apagado de las reglas es unido mediante 'AND' lógicas.

Si el usuario no desea este tipo de relación entre reglas y desea una combinación más compleja con operadores AND y OR combinados, deberá crear una regla avanzada. Las reglas avanzadas se crearán mediante un editor de texto integrado en la propia aplicación, en el cuál se tendrán que escribir las reglas relacionadas mediante paréntesis y los operadores AND / OR. Si la regla está escrita correctamente con respecto al mini-lenguaje creado para la ocasión, entonces podrá tratarse como una regla más y dará un valor de Bluetooth encendido / Bluetooth apagado. A su vez, las reglas complejas también pueden relacionarse con otras mediante prioridad al encendido y prioridad al apagado. Para evitar que puedan 'molestar' las reglas básicas en estos casos, todas las reglas (Aunque estén creadas) tienen la opción de ser activadas o desactivadas. Esta opción sera efectiva a nivel alto de regla, es decir, si es una regla compleja, y alguna de sus reglas está 'desactivada', las tratará como activadas, pero la regla simple en sí no contará en la prioridad general (Ya que para poder utilizar una regla en una combinación avanzada, la regla ya tiene que existir previamente).

La idea, es que el usuario 'medio' sea capaz de crear reglas sencillas, y el usuario 'avanzado' puede también crear reglas avanzadas según sus necesidades. De todas formas, se intentará que el usuario tenga la mayor ayuda

posible por parte de la aplicación para explicar todos esto conceptos necesarios para su uso, los cuales son complejos de inicio para quien utilice la aplicación. Para ello, se va a tratar de crear una interfaz gráfica de usuario lo más sencilla e intuitiva posible, tomando patrones de diseño de Android, y haciendo hincapié sobre todo en la parte de creación básica de reglas (la que por definición se orienta al usuario medio).

Si los usuarios aún así no son capaces de comprender el funcionamiento del programa, también podrán disponer de una ayuda avanzada accesible en todo momento desde la 'Action Bar' superior (Definición 1 en la página 71). En esta ayuda, la cuál se presentará en forma de "páginas" desplazables mediante gestos, mostrará tanto explicaciones como capturas de pantalla con indicaciones.

También se hará un esfuerzo en conseguir un diseño agradable al usuario, de forma que el aspecto general de la aplicación tenga un aspecto grafico de aplicación para Android, pero alejándose lo más posible del aspecto de componentes 'por defecto', muy común en aplicaciones presentes en 'Google Play' (Definición 2 en la página 71) realizadas por desarrolladores independientes 'amateur'.

Las reglas se almacenarán (en formato XML) dentro del propio dispositivo, o , en caso de existir, en la memoria SD. Pero estas reglas se perderían en caso de borrado de los datos de la aplicación (En las preferencias de Android) o en caso de cambio de dispositivo. Esto es especialmente crítico en el caso de las reglas avanzadas, pues puede que el tiempo de elaboración de una de estas reglas sea alto y no sea agradable volver a realizarla. Para ello, se añadirá una opción de exportar e importar en las preferencias generales de la aplicación: Al exportar, se generará un archivo 'zip' con todos los ficheros XML de la aplicación, y éste podrá ser guardado en cualquier lugar del sistema de archivos (que tenga permisos). Por su parte, al importar, se copiarán los ficheros del .zip importando dentro de la memoria en la cuál la aplicación los guarde (sea la memoria interna o la tarjeta SD).

Una vez definidos los objetivos de la aplicación en sí, hace falta definir la manera en la que se comprobarán las reglas. Para ello, se hará una comprobación cada cierto período de tiempo aplicando la prioridad que el usuario haya seleccionado. Habrá que llegar a un compromiso entre tiempo de comprobación y consumo de batería, ya que no podemos permitir que la propia comprobación de las reglas acabe consumiendo más energía en el dispositivo que el propio uso del adaptador Bluetooth.

1.2. Punto de partida

Inicialmente, este PFC toma como punto de partida la necesidad de ahorrar energía en los smartphones y tablets actuales. La duración de la batería en este tipo de dispositivos es un punto crítico debido al alto consumo de energía como contrapunto de las grandes prestaciones que poseen. Observando las aplicaciones existentes en el universo del sistema operativo Android - el cuál es el escogido para la realización de la aplicación móvil de la que hemos hablado anteriormente - vemos que la mayoría hacen uso de una interfaz y potencia generalista, es decir, no disponen de un control avanzado de ellas mismas debido a que intentan abarcar muchos puntos de ahorro de energia a la vez (Gps, brillo de la pantalla, WiFi... etcétera) y deben simplificar al máximo la posible maraña de opciones como resultado de la cantidad de parámetros posibles dentro de tantas combinaciones posibles. Al contrario que estas aplicaciones, la nuestra se especializará en el adaptador Bluetooth, lo cual hará que su gestión sea más fina, es decir, con mayor detalle que en una aplicación más general.

También debemos tener en cuenta que hay aplicaciones más potentes en cuanto a ahorro de energía, pero que necesitan de permisos de superusuario en el dispositivo (Es decir, como se dice comúnmente, que el dispositivo esté "rooteado") ya que modifican algunos parámetros que por defecto no deberían ser (ni son) accesibles para el usuario medio. Hablamos de parámetros como la frecuencia de la CPU o GPU, la tensión interna... etc. En nuestro software, debemos partir de la base de que el dispositivo en el que se va a ejecutar no tendrá disponible el usuario "root", es decir, no es un dispositivo "rooteado".

Por último, debemos partir de la base que algunos dispositivos no tendrán Bluetooth, por lo que la ejecución del propio software carece de sentido. Entonces, la aplicación deberá avisar al usuario y solicitar su desinstalación.

Capítulo 2

Descripción del entorno

2.1. Bluetooth

En primer lugar, deberíamos definir la tecnología Bluetooth, ya que es el centro de nuestra aplicación.

Tal y como se extrae de [Wikipedia2012] , Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN Wireless Personal Area Network), que posibilita la transmisión de datos entre diferentes dispositivos utilizando el aire como canal. Está diseñado para dispositivos donde se necesite un bajo consumo, una distancia pequeña de cobertura, y unos costes reducidos.

Opera en una frecuencia en el rango entre 2.4 y 2.48 Ghz de la banda ISM (Industrial, Scientific and Medical), la cual está disponible sin licencia a nivel mundial. Para evitar interferencias con otros protocolos y entre dispositivos, se emplea una técnica de saltos de frecuencia de espectro ensanchado, que consiste en dividir la banda en 79 canales (23 en España, Francia y Japón) de 1Mhz de longitud, y realizar entre estos canales 1600 saltos por segundo.

Las características originales son:

- Separación de la banda de frecuencia en saltos. En una conexión se va saltando de un canal a otro, mejorando de esta manera la seguridad.
- Piconet3de hasta ocho dispositivos.
- La señal puede atravesar incluso paredes.
- Los dispositivos no tienen por qué estar uno enfrente del otro, ya que las señales son omni-direccionales.
- Posibilidad de usar aplicaciones síncronas y asíncronas:

- Enlace asíncrono sin conexión (ACL):
- Conexiones simétricas o asimétricas multipunto entre maestro y esclavo.
- Conexión utilizada para tráfico de datos.
- Sin garantía de entrega.
- Máxima velocidad de envío en una dirección: 721kbps , y de 57.6kbps en la otra.
- Enlace síncrono orientado a conexión (SCO):
- Conexiones simétricas punto a punto entre maestro y esclavo.
- Voz en tiempo real y tráfico multimedia.
- Velocidad de transmisión: 64kbps.

Segun [Wireless2000], las especificaciones fueron establecidas formalmente en 1998 por el Bluetooth Special Interest Group (SIG) , que fue creado por Ericsson, IBM, Intel, Toshiba y Nokia, aunque posteriormente se sumaron más compañías.

Las diferentes versiones de los estándares están diseñadas para mantener la compatibilidad hacia atrás y, hasta la fecha, son las siguientes:

- Bluetooth v1.0 y v1.0B:
  - Los dispositivos que implementaban esta versión, tenían dificultades para conseguir compatibilidad hacia otros dispositivos de diferente fabricante.
  - Utilización de canales encriptados.
- Bluetooth v1.1: Ratificado como estándar IEEE 802.15.1-2002.
  - Corrección de erratas en la especificación v1.0b.
  - Soporte para canales no encriptados.
  - Indicador de calidad de señal recibida (RSSI).
- Bluetooth v1.2: Es la primera versión compatible con USB 1.1.
  - Se añadió el 'Discovery',

- AFH (Salto de frecuencia adaptable de espectro ensanchado) mejorando la resistencia a interferencias.
- Velocidad de transmisión hasta los 721 kbit/s.
- Tipo de enlace para aplicaciones de audio ESCO (Conexiones Síncronas Extendidas), mejorando la calidad de voz.
- Mejoras en el HCI (Host Controller Interface), permitiendo una sincronización más rápida de las comunicaciones.
- Control de flujo y modo de retransmisión L2CAP.
- Estándar IEEE 802.15.1-2005.

- Bluetooth v2.0:
  - Incorporación de la tecnología EDR (Enhanced data Rate): Incremento de las velocidades hasta 3Mbps de manera ideal.
  - Reducción del consumo de energía.

- Bluetooth v2.1:
  - SSP (Secure Simple Pairing) : Conexión inicial entre pares simplificada.
  - EIR (Respuesta amplia de de Investigación): Mejor filtrado de los dispositivos a los que conectarse.
  - Posibilidad de utilizar otras formas de emparejado: NFC (Near Field Communication)
  - Reducción de energía en el modo de bajo consumo.

- Bluetooth v3.0 + HS:
  - AMP (alternate MAC/PHY),
  - Unicast de datos sin conexión
  - 802.11 Protocol Adaptation Layer (PAL): Transferencia de datos inalámbrica de hasta 24 Mb/s mediante el protocolo WiFi.
  - Control de energía mejorado.

- Bluetooth v4.0
  - Alcance mayor (+100m).
  - Encriptación AES-128.

- Baja latencia (3ms).
- Interoperabilidad multi-vendor.
- Mejora en el consumo.

Los dispositivos que implementen este protocolo, pueden pertenecer a diferentes clases según su potencia y alcance:

- Clase 1: 100mW de potencia máxima, cobertura de ~100m.
- Clase 2: 2,5 mW de potencia máxima, cobertura de ~10m.
- Clase 3: 1 mW de potencia máxima, cobertura de ~1m.

Las aplicaciones deben implementar una serie de perfiles de Bluetooth, es decir, una serie de interfaces de alto nivel para estandarizar un cierto ámbito de aplicación de la tecnología. Una especificación de un perfil debe cubrir dependencias con otros perfiles, formatos recomendados para la interfaz con el usuario, y partes concretas de la pila Bluetooth que se utilizan.

2.2. Android

Por lo que respecta al sistema operativo que vamos a utilizar, Android, debemos definir sus conceptos básicos.

2.2.1. Definición

Según [ADevelopers2012a], Android es una pila de software para dispositivos móviles, la cual incluye sistema operativo, middleware 4 y aplicaciones clave. El SDK de Android, ofrece las herramientas y APIs necesarias para comenzar a desarrollar aplicaciones para la plataforma usando el lenguaje de programación Java.

Algunas características del sistema Android son:

- Framework de aplicación basado en la reutilización y reemplazo de componentes.
- Máquina virtual Dalvik, la cual no es una máquina virtual de Java estrictamente hablando. Está optimizada para un consumo reducido de memoria y está diseñada para ejecutar varias instancias simultáneamente, de manera que es el sistema operativo el que se encarga del aislamiento de procesos, gestión de memoria e hilos. Aunque las aplicaciones Android se programen en Java, el código intermedio que interpreta Dalvik no es bytecode de Java, aunque la herramienta Dx incluida en el SDK permite la conversión de archivos .class al formato de archivo de Dalvik (.dex).
- Navegador integrado, basado en el motor Webkit.
- Gráficos con librerías propias para 2D , y gráficos en 3D basado en la especificación de OpenGl ES 1.0 (Con aceleración hardware opcional).
- Sistema de gestión de base de datos SQLite.
- Soporte multimedia para audio y vídeo en diversos formatos: (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- Telefonía GSM (Dependiente del hardware).
- Bluetooth, EDGE, 3G, y WIFI (Dependiente del hardware).

Aunque inicialmente fue desarrollado por Android Inc. (firma comprada por Google en el año 2005), actualmente está siendo desarrollado por la Open Handset Alliance (Liderada por Google), un conglomerado de fabricantes y

desarrolladores de hardware, software y operadores de servicio. Está basado en el kernel de Linux, y actualmente la gran mayoría del código de Android está liberado bajo licencia Apache.

2.2.2. Arquitectura

Android está formado por varias capas que facilitan al desarrollador la creación de aplicaciones. La distribución en capas permite acceder a capas más bajas mediante el uso de librerías, evitando la programación a bajo nivel por parte del desarrollador.

Como nos cuenta [Condesa2011], cada una de las capas utiliza elementos de la capa inferior para realizar sus funciones, formando una pila:

Kernel Linux: El núcleo del sistema operativo Android está basado en el kernel 2.6 de Linux, similar al que se puede utilizar en distribuciones de Linux, pero con ciertas modificacioens, como optimización para un mayor rendimiento.

El programador no accede al kernel directamente, sino que debe utilizar librerías disponibles en capas superiores. Se encarga de gestionar los recursos del dispositivo (energía, memoria...) y del sistema operativo en sí (Procesos, comunicación, red... etc). También están incluidos los drivers de los componentes que tenga el dispositivo móvil.

Librerías: Son las bibliotecas nativas de Android. Están escritas en C/C++ y compiladas para la arquitectura hardware específica del dispositivo. Están hechas normalmente por el fabricante, el cuál se encarga de que estén presentes en el dispositivo antes de su venta. Las liberías proporcionan funcionalidad a las aplicaciones para tareas repetidas con frecuencia, evitando tener que codificarlas cada vez y garantizando que se realicen de manera eficiente.

Entre las librerías incluidas habitualmente se encuentran: OpenGL, bibliotecas multimedia (audio, video, e imagen), Webkit, SSL, FreeType, SQLite...

Entorno de ejecución (Runtime): Es una sub-capa formada por librerías. El componente principal es la máquina virtual Dalvik, ya mencionada anteriormente.

Framework de aplicaciones : Formada por las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a recursos de las capas inferiores a través de Dalvik. Las más importantes son:

Activity Manager: Se encarga de administrar la pila de actividades de nuestra aplicación y su ciclo de vida. Una actividad es lo que equivaldría a una 'ventana' en un sistema operativo convencional para PC como Windows, o Linux con escritorio gráfico (Gnome, KDE, XFCE, LXDE... etc), con la

particularidad de ocupar la pantalla entera y ser apilable, pudiendose así mostrar otras actividades relacionadas (o no) posicionándose 'encima'.

Windows Manager: Se encarga de organizar lo que se mostrará en pantalla. Básicamente, crea superficies en la pantalla que posteriormente pasarán a ser utilizadas por las actividades.

Content Provider: Es la librería que encapsula los datos que se compartirán entre las aplicaciones, de manera que tiene el control sobre el acceso a la información.

Views: Una vista es un elemento básico del cual parten todos los elementos básicos que mostrarán las actividades: Botones, cuadros de texto, listas... etc.

Notification Manager: Engloba los servicios para notificar al usuario cuando algo requiera su atención mostrando alertas en la barra de estado. Esta librería también permite la utilización de sonidos, activar la vibración del dispositivo, o utilizar leds en caso de existir.

Package Manager: Permite obtener información sobre los paquetes instalados en el dispositivo Android, y también se encarga de la instalación de nuevos paquetes. Los paquetes de Android tienen la extensión '.apk', y contienen los binarios '.dex' y todos los recursos dependientes de éste.

Telephony Manager: Esta librería permite realizar llamadas o enviar y recibir SMS/MMS, aunque no puede reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.

Resource Manager: Permite la gestión de todos los elementos que están incluidos en la aplicación pero fuera del código.

Location Manager: Permite determinar la posición geográfica del dispositivo Android mediante GPS o redes disponibles y trabajar con mapas.

Sensor Manager: Permite manipular los elementos de hardware del teléfono tales como acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, sensor de presión, sensor de proximidad, sensor de temperatura...

Cámara : Esta librería proporciona la posibilidad de utilizar las cámaras del dispositivo para tomar fotografías o vídeo.

Multimedia: Permite reproducir y visualizar vídeo, imágenes y audio.

Aplicaciones : Es la capa de más alto nivel. En ella se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz gráfica de usuario como las que no, las nativas (programadas en C/C++) y las administradas (En Java). También se encuentra la aplicación principal del sistema: el Home o Launcher, que es la que permite ejecutar otras aplicaciones mostrándolas en forma de lista, y proporcionando al usuario una serie de escritorios donde colocar accesos directos, Widgets de escritorio.... etcétera.

Figura 2.1: Arquitectura del sistema Android

2.2.3. Anatomía de las aplicaciones Android

2.2.3.1. Componentes de una aplicación

Una aplicación Android, tal y como cuenta [ADevelopers2012b], está compuesta por los siguientes elementos:

- Activity (Actividad): Según su definición en Android Developers, es un componente de aplicación que provee una pantalla sobre la cuál el usuario puede interactuar para conseguir un objetivo (por ejemplo, llamar por teléfono, hacer una fotografía, enviar un correo electrónico, ver un mapa... etcétera). Cada actividad proporciona una ventana en la cual dibuja su interfaz de usuario. La ventana, normalmente se adapta a la totalidad de la pantalla.
  Una aplicación se compone, normalmente, de múltiples actividades interrelacionadas entre ellas. Lo común es que exista una actividad principal que se abra en cuanto se ejecute la aplicación.
  Cada Activity puede iniciar otras actividades si la lógica del programa lo requiere. Cada vez que una actividad se inicia, la anterior Activity se para (pasa al estado 'stopped' visto más adelante), pero la preserva dentro de la 'pila de actividades', por debajo de la nueva. Sin embargo, si el usuario pulsa el botón atrás del dispositivo Android, la Activity que estaba en pantalla se desapila, mostrando la que había anteriormente.

Posteriormente en este documento se detallará más acerca del ciclo de vida de una actividad.

- Service (Servicio): Un servicio es un componente que puede realizar operaciones ejecutándose en segundo plano, sin proporcionar ninguna interfaz gráfica. Un servicio puede ser iniciado por otro componente de aplicación y seguirá ejecutándose en segundo plano aunque el usuario cambie a otra aplicación.
  Adicionalmente, un componente puede unirse a un servicio para interactuar con él mediante comunicación interprocesos.
  Un servicio tiene dos formas:

  - Started (iniciado): Ocurre cuando un componente de aplicación (como una actividad) inicia un servicio con la llamada 'startService()' . Una vez iniciado el servicio, éste se ejecuta en segundo plano indefinidamente aunque el componente iniciador sea destruido. Normalmente se utiliza para realizar una sola operación que no devuelva nada al iniciador.
  - Bound (Enlazado) : Ocurre cuando un componente de aplicación inicia un servicio con la llamada 'bindService()'. Un servicio enlazado ofrece una interfaz cliente - servidor que permite a los componentes interactuar con él, enviar peticiones, obtener resultados... etcétera. Un servicio enlazado sólo está en ejecución mientras exista algún componente enlazado con él. Se pueden enlazar múltiples componentes a un servicio, pero en cuanto todos sean destruidos, el servicio también se destruirá.

  Aunque existen dos formas a priori diferenciadas, un servicio también puede funcionar de ambas formas a la vez (Puede ser iniciado con 'startService()' pero puede permitir ser enlazado a posteriori). Esto depende de un grupo de métodos, los cuales pueden ser sobreescritos para propocionar una o ambas funcionalidades.

- BroadcastReceiver (Receptores de difusión): Se trata de un componente destinado a detectar y reaccionar ante eventos, mensajes globales del sistema u otras aplicaciones que manden un mensaje (Mediante 'Intents', el cuál es un mecanismo de paso de mensajes entre procesos además de iniciador de acciones).

- ContentProvider (Proveedores de contenido) : Un Content Provider es el encargado de manejar el acceso a información estructurada. Se encarga de encapsular la información, y ofrecer mecanismos para definir

la seguridad de la información. Son la interfaz estándar que conecta los datos de un proceso con código en ejecución de otro proceso.
Cuando quieres acceder a información por medio de un Content Provideer, puedes usar el objeto 'ContentResolver' dentro de la aplicación.
Android incluye una serie de proveedores de contenido por defecto para manejar audio, vídeo, imágenes e información personal de contacto. Con algunas restricciones, son accesibles por cualquier aplicación de Android.

2.2.3.2. Ciclo de vida de una actividad:

Como hemos indicado anteriormente, Android maneja las actividades como una pila: Cuando se crea una actividad, se coloca en lo más alto de la pila y se convierte en la Activity en ejecución. En la siguiente figura, extraída de [ADevelopers2012], podemos observar como los diferentes métodos del ciclo de vida de una actividad dan lugar a diferentes estados de ésta:

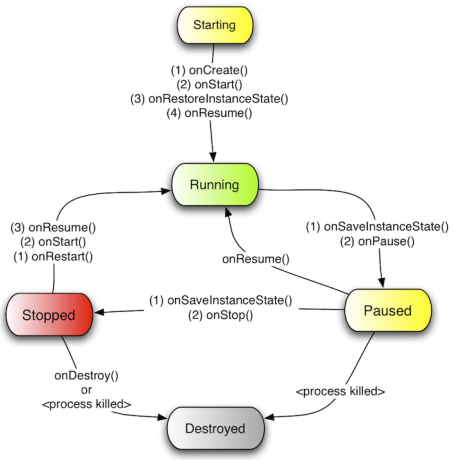


Figura 2.2: Ciclo de vida de las actividades en Android.

Capítulo 3

Definición del problema

3.1. Definición del problema

Durante muchos años, la telefonía móvil y la informática han ido evolucionando a pasos agigantados. Mientras los ordenadores han seguido un camino en el que se desea conseguir cada vez más y mejores prestaciones, los dispositivos móviles se han centrado en ofrecer mejores servicios consiguiendo el menor consumo energético posible. Durante un tiempo, las PDA llegaron a ser las reinas dentro de la informática 'de bolsillo', con una relación consumo / prestaciones aceptables para la época. Sin embargo, la popularidad de los Smartphones, los cuales integran las PDA con los teléfonos móviles, ha favorecido la desaparición de estos dispositivos, quedando en el recuerdo de muchos y muchas. En la actualidad, hay dos grandes familias de sistemas operativos para smartphones y tablets: iOS de Apple (Para iPhone, iPad y iPod), y Android (Sobre el cuál nos centraremos más adelante). Por su parte, Blackberry continúa con su gama de smartphones y su sistema operativo propio, y Microsoft intentó remontar en el mundo de los smartphones con Windows Phone (Un buen producto, pero tardío con un mercado ya copado), y en el de los tablets modernos con la Microsoft Surface y Windows 8 de sistema operativo (Resaltando el hecho de que a principios de la década de los 2000, fue Microsoft la que lanzó su Tablet PC, con más bien poco éxito). Por su parte, Android (Desarrollado por Google, aunque originario de la Open Handset Alliance), ha conseguido hacerse un hueco gracias a su flexibilidad ante las diferentes características de los dispositivos, a costa de aumentar el coste temporal en el desarrollo de aplicaciones. Volviendo al consumo de los dispositivos, las baterías han conseguido aumentar su capacidad durante los últimos años, consiguiendo teléfonos móviles clásicos que consiguen permanecer sin necesidad de carga durante más de una o incluso dos semanas.

Sin embargo, la mayor exigencia de los smartphones en cuanto a potencia, también hacen que la duración de la batería se acorte radicalmente a pesar de los avances comentados anteriormente. Es por ello, que cada vez es más común la utilización de aplicaciones o herramientas que, ya sea automatizando algunos procesos o accediendo a parámetros como control de voltaje, de velocidad de procesador... etc, nos permitan alargar sensiblemente el período hasta la siguiente recarga. En la presente memoria, se detalla esta manera de abordar el problema de la autonomía de los dispositivos: El diseño e implementación de una aplicación para el sistema operativo Android que consiga economizar el consumo de la batería mediante la automatización del encendido y apagado del adaptador Bluetooth del dispositivo móvil.

Para poder resolver este problema, debemos también lidiar con otros directamente relacionados con los dispositivos Android: La fragmentación de dispositivos. Existen dispositivos con Bluetooth, sin Bluetooth, con diferente comportamiento al tratar con el adaptador Bluetooth... etcétera.

Con un poco de experiencia en el desarrollo Android, se puede intuir que, sin disponer de la gran cantidad de dispositivos existentes, es prácticamente imposible hacer funcionar una aplicación en el cien por cien de los dispositivos, por lo menos, de inicio. Siempre habrá algún dispositivo que, por alguna diferencia en el hardware , tratamiento del ciclo de vida... etcétera, el programa fallará o no funcionará como es esperado. Es por ello, que el diseño inicial del programa tiene un peso muy importante como solución, así como mantener el propio software aunque esté publicado ya en la plataforma "Google Play" e ir corrigiendo fallos y bugs.

Capítulo 4

Análisis

En esta parte de la memoria, hablaré del trabajo de diseño de la aplicación. En primer lugar, se presentan los requisitos de la aplicación tanto funcionales, como no funcionales y de implementación. Tras ello, se presentarán los actores que influyen en la aplicación, y posteriormente, los casos de uso de la aplicación. Una vez hecho esto, se mostrará el modelo de casos de uso de la aplicación.

4.1. Requisitos

4.1.1. Requisitos funcionales

- El usuario deberá poder arrancar el servicio de encendido/apagado de Bluetooth. Para ello, bastará con abrir la aplicación y observar el icono de ésta en la bandeja del sistema.
- El usuario deberá poder detener el servicio de encendido/apagado de Bluetooth. Para ello, bastará con presionar la opción 'salir' en el menú principal (dashboard 5) de la aplicación.
- El usuario podrá crear 'reglas de Bluetooth' en el modo 'estándar'. Las reglas que podrá crear serán de tiempo, de lugar, de dispositivo o de batería.
- El usuario podrá crear 'reglas de Bluetooth' en el modo 'avanzado'. Para ello, deberá conocer previamente la sintaxis del lenguaje de especificación de reglas definido en la ayuda de la aplicación.
- El usuario debe poder editar 'reglas de dispositivo' tanto en el modo 'estándar' como en el 'avanzado'.

- El usuario podrá eliminar cualquier regla en el modo 'estándar'. La aplicación deberá poder guardar las reglas en la memoria del dispositivo, sea interna o externa.
- Al crear o editar una regla de tiempo, el usuario podrá introducir tanto fecha inicial y final, como día de la semana como hora inicial y final, el los cuales el dispositivo Bluetooth permanecerá encendido.
- Al crear o editar una regla de batería, el usuario podrá introducir el nivel de batería 'umbral'.
- Al crear o editar una regla de dispositivo, el usuario podrá buscar dispositivos cercanos, y al seleccionar uno de ellos, el sistema deberá ser capaz de almacenarlo como 'dispositivo conocido'.
- Al crear o editar una regla de dispositivo, el usuario deberá poder eliminar dispositivos conocidos.
- Al crear o editar una regla de lugar, el usuario deberá poder eliminar lugares conocidos.
- Al crear o editar una regla de lugar, el usuario podrá localizarse y, si lo desea, mandar al sistema almacenar la ubicación como conocida. El sistema debe ser capaz, en función de las reglas definidas por el usuario, activar o desactivar el dispositivo Bluetooth en concordancia con éstas.
- El usuario deberá escoger entre dar prioridad a las reglas que mandan encender el Bluetooth o las que mandar apagarlo.
- El usuario de la aplicación podrá abrirla directamente desde el escritorio y de forma rápida.
- El usuario, podrá exportar en un fichero todas las reglas u otra información de la aplicación. De esta manera podrá transportarlo o realizar una copia de seguridad.

4.1.2. Requisitos no funcionales

- La aplicación deberá tener una interfaz intuitiva, acompañada de una ayuda detallada en la justa medida para ser fácil de entender y útil.
- La aplicación deberá mejorar el consumo de batería, es decir, no debe consumir más batería de la que ahorra.

- La aplicación deberá consumir poca memoria RAM y ocupar poco espacio en la memoria interna/externa del dispositivo.

4.1.3. Requisitos de implementación

- La aplicación deberá ser desarrollada para el sistema operativo Android, a partir de la versión 2.1 (Eclair). A su vez, la densidad o resolución de pantalla no deberá influir en el funcionamiento de la aplicación.
- La aplicación no deberá ejecutarse en los dispositivos Android que no dispongan de un adaptador Bluetooth.
- La aplicación deberá ajustarse a los patrones de diseño de Android, presentes en el apartado de diseño de Android Developers.

4.2. Actores

Los actores que tienen cabida en los casos de uso de esta aplicación son los siguientes:

Actor	Usuario	**Identificador**	USU1		
Descripción	Persona usuaria de la aplicación móvil Android.				
Características	Usuario de Smartphone o tablet con sistema operativo Android 2.1 o superior, que disponga de la aplicación 'Bluetooth Administrator'.				
Relaciones					
Referencias	ARSE1, DETSE1, ELIREG1, CAMPRIO1, EXPOREG1, IMPOREG1				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1

Atributos		
Nombre	Descripción	Tipo
Vacío	Vacío	Vacío

Comentarios
El usuario será genérico, es decir, no se tiene en cuenta su nivel de conocimiento acerca de Android, Bluetooth... etcétera.

Actor	Usuario Básico	**Identificador**	USUBA1		
Descripción	Persona usuaria de la aplicación móvil Android, con conocimientos básicos.				
Características	Usuario de Smartphone o tablet con sistema operativo Android 2.1 o superior, que disponga de la aplicación 'Bluetooth Administrator'.				
Relaciones	Extiende de USU1				
Referencias	RETEMP1, REBAT1, RELUG1, REDISP1, EDREST1				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1

Atributos		
Nombre	Descripción	Tipo
Vacío	Vacío	Vacío

Comentarios
Se trata de un usuario que, por su nivel de conocimiento, utilizará el modo estándar de la aplicación.

Actor	Usuario Avanzado	**Identificador**	USUAV1		
Descripción	Persona usuaria de la aplicación móvil Android, con conocimientos avanzados.				
Características	Usuario de Smartphone o tablet con sistema operativo Android 2.1 o superior, que disponga de la aplicación 'Bluetooth Administrator'.				
Relaciones	Extiende de USUBA1				
Referencias	REAVA1				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1

Atributos		
Nombre	Descripción	Tipo
Vacío	Vacío	Vacío

Comentarios
Se trata de un usuario que, por su nivel de conocimiento, utilizará tanto el modo estándar de la aplicación como el avanzado.

4.3. Casos de uso

Caso De Uso	Arranque Servicio Principal			**ARSE1**	
Actores	Usuario				
Tipo	Primario				
Referencias					
Precondición	La aplicación no debe estar iniciada				
Postcondición	El servicio de comprobación de Bluetooth estará iniciado en segundo plano.				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1
Propósito					
Iniciar el servicio que comprueba si se debe activar el Bluetooth					
Resumen					
Cuando el usuario inicia la aplicación, si no está ya en marcha el 'servicio Bluetooth', se iniciará en segundo plano. El servicio Bluetooth será en encargado de realizar las comprobaciones de las reglas y la prioridad escogida de activación para activar o desactivar el adaptador Bluetooth.					
Curso Normal (Básico)					
1	El usuario pulsa el icono para iniciar la aplicación				
		2	Se muestra el panel principal de la aplicación		
		3	Se inicia el servicio de Bluetooth		
Cursos Alternos					
3b	Si ya esta el servicio en marcha, no se hace nada				

Caso De Uso	Detención Servicio Principal	DETSE1
Actores	Usuario	

Tipo	Primario				
Referencias					
Precondición	El servicio Bluetooth debe estar iniciado. La aplicación debe mostrar el Dashboard (Menú principal).				
Postcondición	El servicio de comprobación de Bluetooth estará detenido.				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1
Propósito					
Detener el servicio que comprueba si se debe activar el Bluetooth.					
Resumen					
Al detener el servicio Bluetooth, cesarán las comprobaciones de las reglas. El dispositivo Bluetooth permanecerá en el estado en el que se encontraba antes de la detención.					
Curso Normal (Básico)					
1	El usuario pulsa el icono para salir de la aplicación				
		2	Se detiene el servicio Bluetooth		
		3	Se cierra la aplicación		
Cursos Alternos					

Caso De Uso	Creación de regla de tiempo			RETEMP1	
Actores	Usuario Básico				
Tipo	Primario				
Referencias					
Precondición	El servicio de comprobación de Bluetooth estará iniciado en segundo plano.				
Postcondición	La regla definida será almacenada por el sistema y podrá ser comprobada.				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1
Propósito					
Creación y almacenaje en el sistema de una regla de tiempo.					
Resumen					

El usuario introducirá sus preferencias de activación para una regla de tiempo (Fecha inicio - fin, hora inicio-fin, día de la semana)			
Curso Normal (Básico)			
1	El usuario entra en el modo de edición estándar	2	Se muestra el panel de edición estándar
3	El usuario solicita añadir una regla de tiempo	4	La aplicación muestra el diálogo de añadir una regla de tiempo
5	El usuario añade la información de tiempo		
6	El usuario decide añadir la regla	7	La aplicación guarda la regla en el sistema
		8	La aplicación vuelve al panel de edición estándar.
Cursos Alternos			
7b	Si ya hay una regla con ese nombre, vuelve a 4.		

Caso De Uso	Creación de regla de batería			REBAT1	
Actores	Usuario Básico				
Tipo	Primario				
Referencias					
Precondición	El servicio de comprobación de Bluetooth estará iniciado en segundo plano.				
Postcondición	La regla definida será almacenada por el sistema y podrá ser comprobada.				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1
Propósito					
Creación y almacenaje en el sistema de una regla de batería.					
Resumen					
El usuario introducirá el umbral de batería para activar / desactivar el Bluetooth.					
Curso Normal (Básico)					

1	El usuario entra en el modo de edición estándar	2	Se muestra el panel de edición estándar
3	El usuario solicita añadir una regla de batería	4	La aplicación muestra el diálogo de añadir una regla de bateria
5	El usuario añade la información de umbral de batería		
6	El usuario decide añadir la regla	7	La aplicación guarda la regla en el sistema
		8	La aplicación vuelve al panel de edición estándar.
Cursos Alternos			
7b	Si ya hay una regla con ese nombre, vuelve a 4.		

Caso De Uso	Creación de regla de lugar			RELUG1	
Actores	Usuario Básico				
Tipo	Primario				
Referencias					
Precondición	El servicio de comprobación de Bluetooth estará iniciado en segundo plano.				
Postcondición	La regla definida será almacenada por el sistema y podrá ser comprobada.				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1
Propósito					
Creación y almacenaje en el sistema de una regla de lugar.					
Resumen					
El usuario introducirá sus preferencias de activación para una regla de lugar (Longitud y latitud del lugar)					
Curso Normal (Básico)					
1	El usuario entra en el modo de edición estándar		2	Se muestra el panel de edición estándar	

3	El usuario solicita añadir una regla de lugar	4	La aplicación muestra el diálogo de añadir una regla de lugar
		5	La aplicación solicita activar el GPS
6	El usuario marca el lugar donde está como el lugar a guardar	7	La aplicación guarda la regla en el sistema
		8	La aplicación vuelve al panel de edición estándar.
Cursos Alternos			
7b	Si ya hay una regla con ese nombre, vuelve a 4.		
5b	Si no hay GPS, se usará geolocalización por red.		
5c	Si no hay GPS ni geolocalización por red, se lanza un mensaje y se cierra la actividad.		

Caso De Uso	Creación de regla de dispositivo			REDISP1	
Actores	Usuario Básico				
Tipo	Primario				
Referencias					
Precondición	El servicio de comprobación de Bluetooth estará iniciado en segundo plano.				
Postcondición	La regla definida será almacenada por el sistema y podrá ser comprobada.				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1
Propósito					
Creación y almacenaje en el sistema de una regla de dispositivo.					
Resumen					
El usuario introducirá los dispositivos conocidos					
Curso Normal (Básico)					
1	El usuario entra en el modo de edición estándar		2	Se muestra el panel de edición estándar	

3	El usuario solicita añadir una regla de dispositivo	4	La aplicación muestra el diálogo de añadir una regla de dispositivo
5	El usuario selecciona añadir un dispositivo conocido.	6	La aplicación muestra el diálogo de añadir un nuevo dispositivo
7	El usuario selecciona el dispositivo y marca la regla para guardar.	8	La aplicación guarda la regla en el sistema
		9	La aplicación vuelve al panel de edición estándar.
Cursos Alternos			
7b	Si ya hay una regla con ese nombre, vuelve a 4.		
5b	Si ya hay añadido algún dispositivo que sea de nuestro agrado, se pasa al paso 7.		

Caso De Uso	Edición de regla estándar			EDREST1	
Actores	Usuario Básico				
Tipo	Primario				
Referencias					
Precondición	El servicio de comprobación de Bluetooth estará iniciado en segundo plano.				
Postcondición	La regla definida será almacenada por el sistema y podrá ser comprobada.				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1
Propósito					
Edición de una regla del dispositivo.					
Resumen					
El usuario modificará una regla.					
Curso Normal (Básico)					

1	El usuario entra en el modo de edición estándar	2	Se muestra el panel de edición estándar
3	El usuario selecciona la regla que quiera editar, y solicita su edición.	4	La aplicación muestra el diálogo de la regla propia, con los datos actuales de ésta.
5	El usuario modifica los datos que considere.		
6	El usuario solicita guardar la regla.	7	La aplicación guarda la regla en el sistema
		8	La aplicación vuelve al panel de edición estándar.
Cursos Alternos			
6b	Si ya hay una regla con ese nombre, vuelve a 4.		

Caso De Uso	Creación de regla avanzada			REAVA1	
Actores	Usuario Avanzado				
Tipo	Primario				
Referencias					
Precondición	El servicio de comprobación de Bluetooth estará iniciado en segundo plano.				
Postcondición	La regla definida será almacenada por el sistema y podrá ser comprobada.				
Autor	Julián Zaragoza	**Fecha**	05/08/2012	**Versión**	v1
Propósito					
Creación de una regla avanzada del dispositivo.					
Resumen					
El usuario creará una regla avanzada.					
Curso Normal (Básico)					
1	El usuario entra en el modo de edición avanzado		2	Se muestra el panel de edición avanzado	

3	El usuario escribe una regla según la sintaxis definida en la ayuda de la aplicación		
4	El usuario comprueba la regla	5	El sistema reconoce la regla como válida. Marca la regla como validada.
6	El usuario solicita guardar la regla.	7	La aplicación guarda la regla en el sistema
		8	La aplicación vuelve al panel de edición estándar.
Cursos Alternos			
5b	Si la regla no es válida, se marca como no validada.		
6b	Si ya existe una regla con ese nombre, se vuelve a 6		

Caso De Uso	Eliminación de regla			ELIREG1	
Actores	Usuario				
Tipo	Primario				
Referencias					
Precondición	Debe existir en el sistema alguna regla.				
Postcondición	La regla elegida dejará de existir en el sistema.				
Autor	Julián Zaragoza	**Fecha**	06/08/2012	**Versión**	v1
Propósito					
Eliminación de cualquier regla del dispositivo.					
Resumen					
El usuario eliminará una regla del sistema, sea estándar (lugar, tiempo, dispositivo o batería) o avanzada.					
Curso Normal (Básico)					
1	El usuario entra en el modo de edición estándar.		2	Se muestra el panel de edición estándar.	

3	El usuario selecciona la regla que desea eliminar		
4	El usuario pulsa la acción de eliminar	5	El sistema pregunta si desea realmente eliminar la regla
6	El usuario confirma la eliminación de la regla	7	La aplicación elimina la regla del sistema
Cursos Alternos			
6b	Si el usuario no confirma la eliminación de la regla, se pasa a 2.		

Caso De Uso	Cambiar de prioridad			CAMPRIO1	
Actores	Usuario				
Tipo	Primario				
Referencias					
Precondición	El usuario debe estar en el panel de edición estándar.				
Postcondición					
Autor	Julián Zaragoza	**Fecha**	06/08/2012	**Versión**	v1
Propósito					
Cambiar la prioridad de las reglas cambiará de 'prioridad encendido' a 'prioridad apagado'.					
Resumen					
Si la prioridad está a encendido (Es decir, ante un conflicto de dos reglas que se contradigan en un momento del tiempo, se de prioridad a la activación del Bluetooth), se cambiará a apagado o viceversa.					
Curso Normal (Básico)					
1	El usuario pulsa el interruptor de cambio de prioridad		2	Se muestra la prioridad contraria a la que funcionaba anteriormente.	
Cursos Alternos					

Caso De Uso	Exportar reglas			EXPOREG1	
Actores	Usuario				
Tipo	Primario				
Referencias					
Precondición	El usuario debe estar en el Dashboard				
Postcondición					
Autor	Julián Zaragoza	**Fecha**	06/08/2012	**Versión**	v1
Propósito					
Guardar las reglas contenidas en el sistema en un archivo.					
Resumen					
El usuario podrá elegir un lugar en el espacio de archivos del dispositivo para guardar un fichero .zip con la información de las reglas contenidas.					
Curso Normal (Básico)					
1	El usuario entra en las opciones del sistema.				
2	El usuario elige la ruta donde se guardará el fichero		3	El sistema guarda el archivo en ese lugar.	
Cursos Alternos					
3b	Si no hay permisos de escritura en ese lugar, se notifica y se vuelve a 2.				

Caso De Uso	Importar reglas			IMPOREG1	
Actores	Usuario				
Tipo	Primario				
Referencias					
Precondición	El usuario debe estar en el Dashboard.				
Postcondición					
Autor	Julián Zaragoza	**Fecha**	06/08/2012	**Versión**	v1
Propósito					
Importar las reglas desde un archivo hasta la memoria del dispositivo					
Resumen					
Curso Normal (Básico)					

1	El usuario entra en las opciones del sistema.		
2	El usuario selecciona la ruta en la que está el archivo .zip con las reglas.	3	El sistema descomprime las reglas en la memoria interna.
Cursos Alternos			

Caso De Uso	Edición regla avanzada			EDREGAV1	
Actores	UsuarioAvanzado				
Tipo	Primario				
Referencias					
Precondición	El usuario debe estar en el panel estándar.				
Postcondición					
Autor	Julián Zaragoza	**Fecha**	06/08/2012	**Versión**	v1
Propósito					
Importar las reglas desde un archivo hasta la memoria del dispositivo					
Resumen					
Curso Normal (Básico)					
1	El usuario selecciona la regla avanzada				
2	El usuario selecciona 'editar'		3		El sistema muestra la pantalla de regla avanzada con los datos de la regla.
4	El usuario edita la regla y selecciona guardar		5		El sistema guarda la regla
Cursos Alternos					
5b	Si la regla no es correcta, el sistema avisa y pregunta si quiere guardar igualmente.				

4.3.1. Modelo de casos de uso

Una vez presentados los actores y los casos de uso, podemos mostrar el Modelo de Casos de Uso de nuestra aplicación.

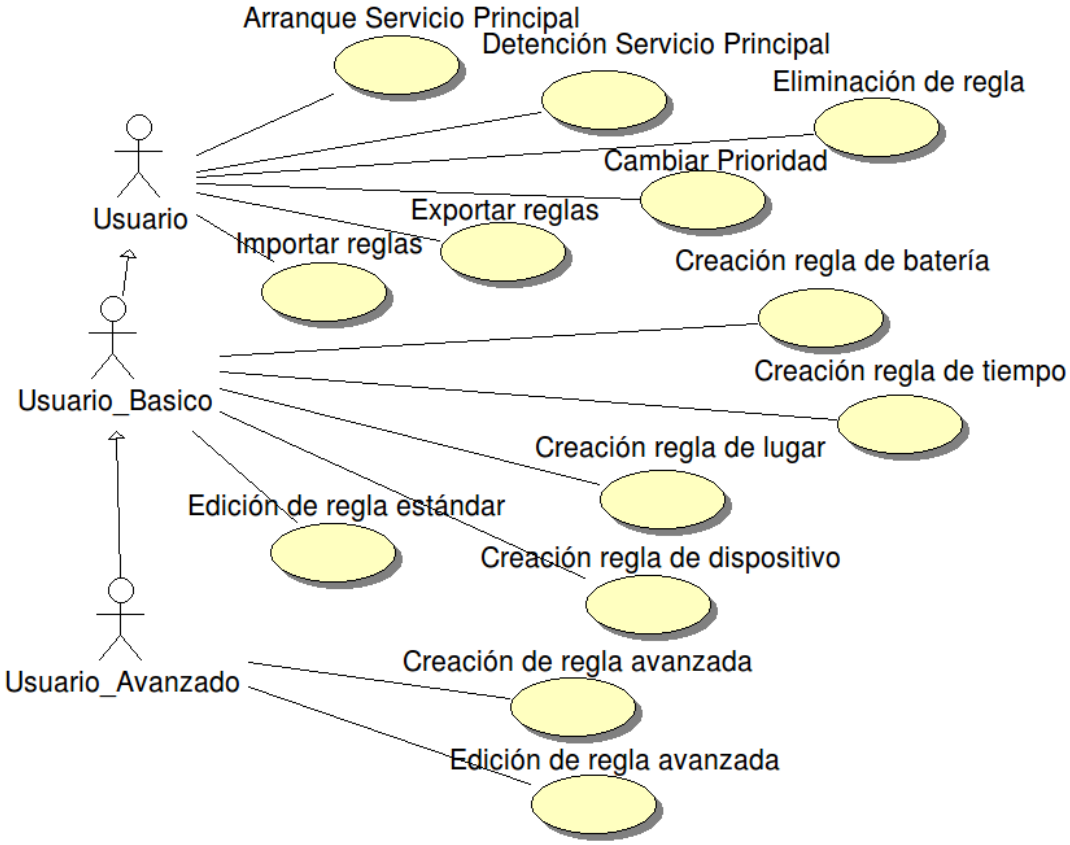


Figura 4.1: Modelo de casos de uso

Capítulo 5

Diseño

En el apartado de diseño vamos a mostrar la arquitectura de la aplicación y su diagrama de clases como paso posterior a la definición de requisitos y casos de uso. También se mostrará un esquema de las pantallas de la aplicación con el flujo lógico entre ellas.

5.1. Diagrama de paquetes

A continuación vamos a mostrar el diagrama de los paquetes que van a componer nuestra aplicación Android. Los paquetes básicos utilizados son: Bluetoothadmin, Bluetoothadmin.libs, Bluetoothadmin.logic, Bluetoothadmin.parser y Bluetoothadmin.services.

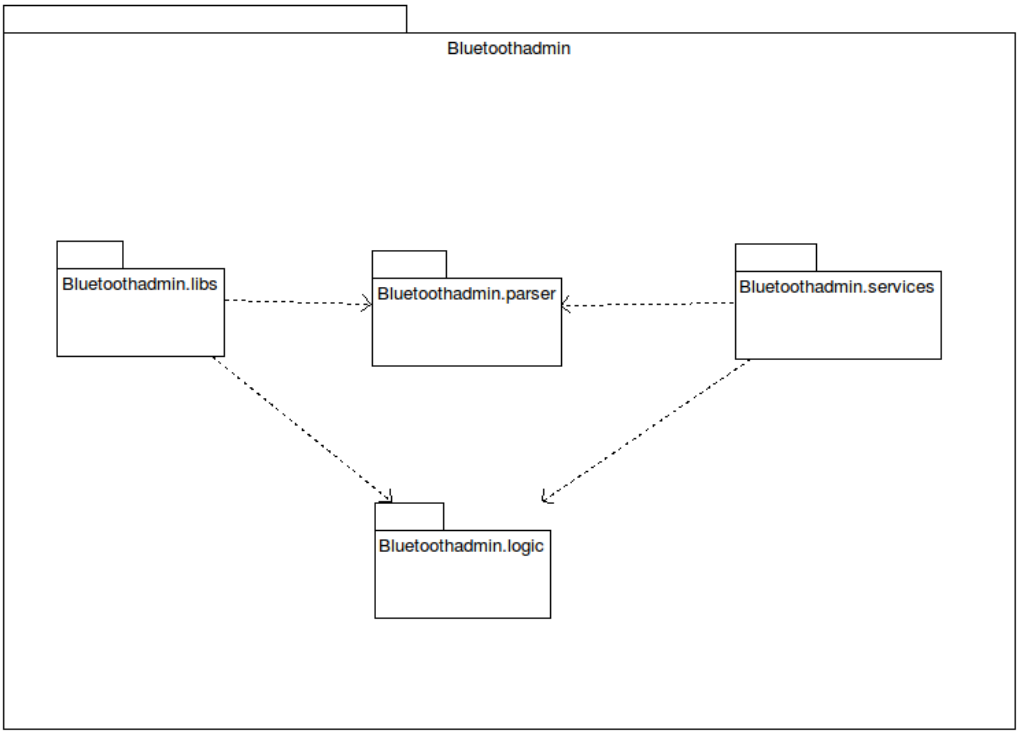


Figura 5.1: Diagrama de paquetes

Vamos a describir los paquetes de la aplicación:

- **Bluetoothadmin** : Es el paquete principal de la aplicación y el resto de paquetes están incluidos en él. Las clases que se encontrarán dentro de este paquete serán las 'Activities', es decir las pantallas gráficas de la aplicación.

- **Bluetoothadmin.libs** : En este paquete se alojarán todas las clases importantes de la aplicación incluyendo reglas, explorador de archivos, listeners de Bluetooth, listeners de posición y librerías para guardar en memoria.

- **Bluetoothadmin.parser** : Aquí estarán las clases del parser del mini-lenguaje del modo avanzado de la aplicación, es decir, el analizador léxico y el sintáctico.

- **Bluetoothadmin.services** : Será el paquete donde se alojen los servicios en segundo plano de la aplicación.

- **Bluetoothadmin.logic** : Aquí se alojarán las clases estáticas de la aplicación, las cuales se encargarán de guardar cierta lógica para el correcto funcionamiento.

Con esta estructura se realizará la implementación de la aplicación, asegurando cierta independencia entre los tipos de clases y mejorando el entendimiento del código fuente del software que se va a realizar.

5.2. Reglas

En esta parte vamos a presentar los diagramas de clases para mostrar la estructura de las reglas de la aplicación.

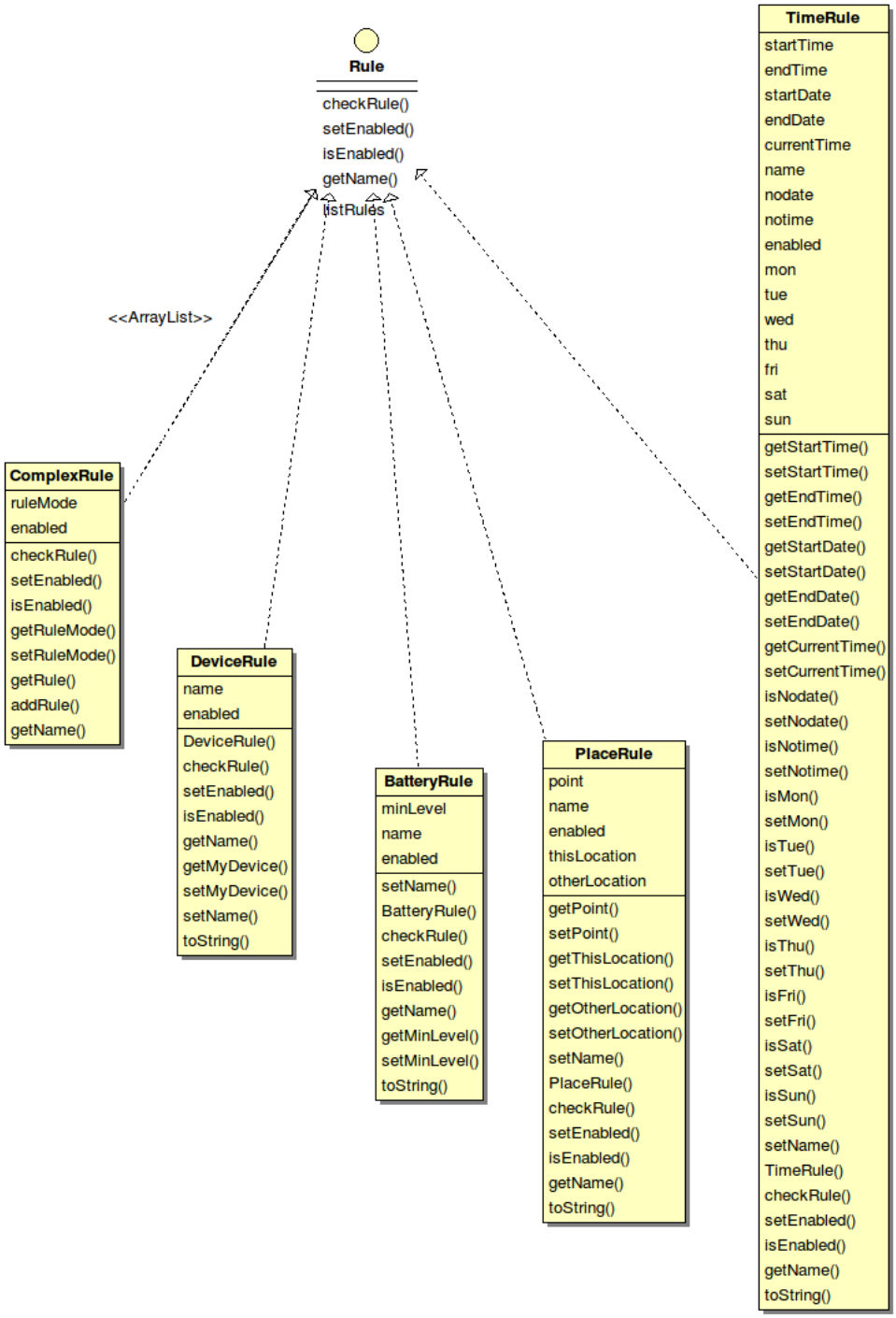


Figura 5.2: Diagrama de clases de las reglas

Como podemos observar en el diagrama de clases de la Figura 3.3, se

creará una interfaz 'Rule' que representará a todas las reglas de la aplicación. De esta manera, cada nuevo tipo de regla implementará esa interfaz.

La interfaz 'Rule' incluye el método 'checkRule', el cuál deberá implementar cada nueva regla del sistema realizando la comprobación de si la regla decide encender el Bluetooth (true) o por el contrario apagarlo (false). Las reglas de la aplicación son las siguientes:

- Rule : Se trata de una interfaz, la cuál implementarán todas las reglas.
- TimeRule : Es la regla de tiempo. En ella se guardarán los intervalos de tiempo tanto de día mensual como de hora, y los días de la semana en los cuáles estará activo el Bluetooth según esta regla.
- PlaceRule: Es la regla de lugar. En ella se almacena tanto la localización donde será efectiva la regla como la actual donde se encuentra el dispositivo.
- BatteryRule: Es la regla de batería. Se almacena el valor umbral de batería.
- DeviceRule: Es la regla de dispositivos. Se almacena el nombre del dispositivo, y los dispositivos 'conocidos' se almacenan en una clase estática para mejor accesibilidad a lo largo de toda la aplicación.
- ComplexRule: Es la regla avanzada. Almacena la cadena del 'minilenguaje', y accede al parser para interpretarla.

5.3. Diseño de la Interfaz Gráfica de Usuario.

Ahora vamos a explicar el diseño elegido para la interfaz gráfica de usuario, tanto a nivel de diseño de Activities como del flujo entre ellas. Para ello, primero vamos a mostrar un diagrama de clases (Figura 3.4) que represente las clases que hagan referencia a la interfaz gráfica del programa. La manera de relacionarlas es mediante una actividad llamada 'ActionBarActivity', la cuál ya contiene métodos para gestionar la selección y el cambio del modo de la aplicación (estándar-avanzado), y a partir de ella se construyen el resto de actividades de la aplicación (Exceptuando las preferencias, que siguen el patrón de diseño de las preferencias de Android).

Antes de explicar el diseño y funcionamiento de cada una de las actividades, vamos a proceder a enumerarlas para posteriormente poder explicar de una manera más directa la lógica entre ellas, y lo podemos ver en la Figura 3.5.

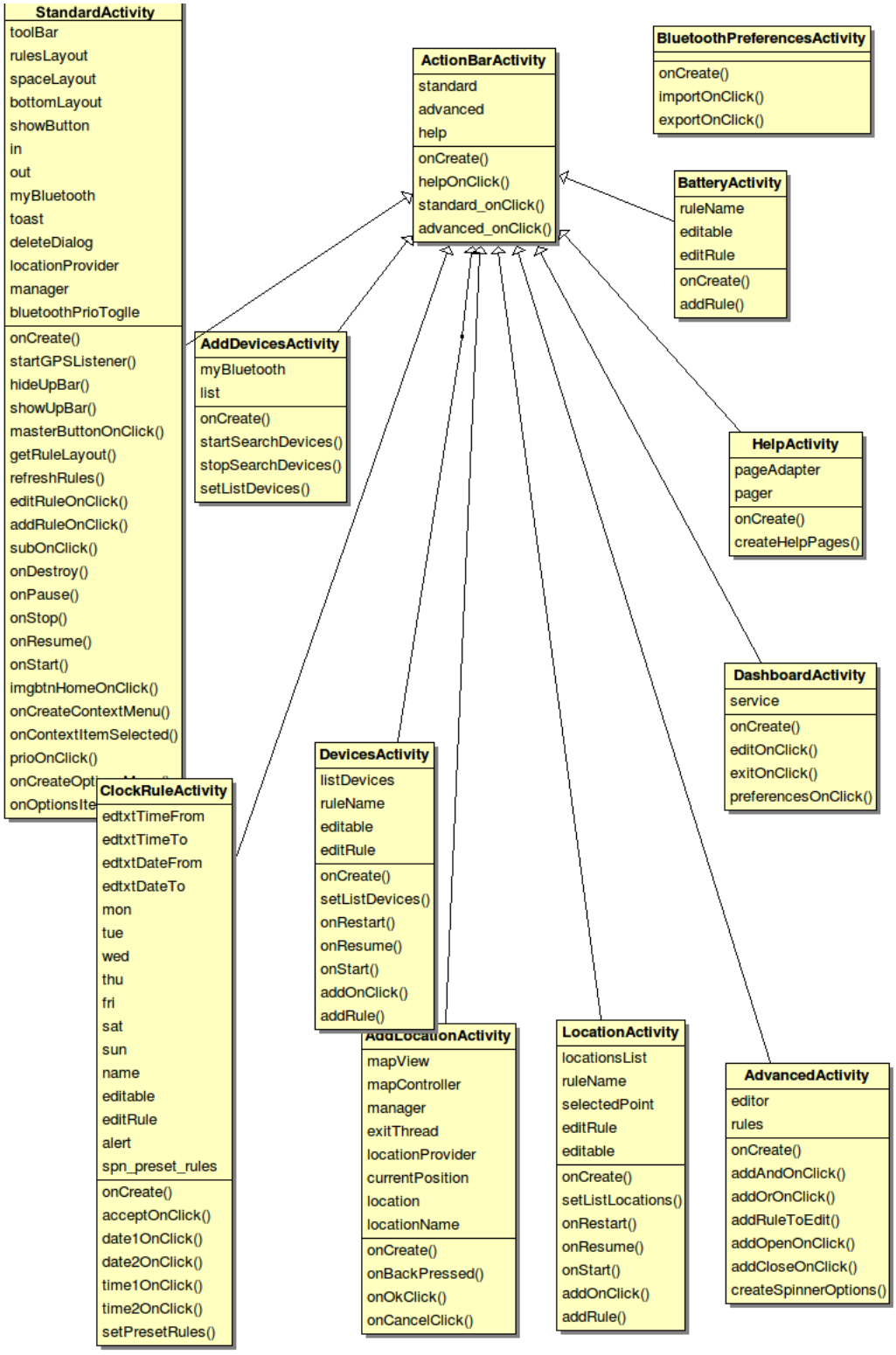


Figura 5.3: Diagrama de clases de Activities

Número	Activity
1	DashboardActivity
2	HelpActivity
3	BatteryActivity
4	AdvancedActivity
5	LocationActivity
6	AddLocationActivity
7	DevicesActivity
8	AddDevicesActivity
9	ClockRuleActivity
10	StandardActivity
11	BluetoothPreferencesActivity
12	AboutDialog

Figura 5.4: Tabla de numeración de Activities

DashboardActivity : Se trata del menú principal de la aplicación, el cuál sigue el patrón de diseño 'Dashboard' 5 de Android. En él tendrá presencia la 'Action Bar' (Otro modelo de diseño de Android) y los accesos a las diferentes partes de la aplicación. El diseño inicial de la correspondiente pantalla se puede observar en la Figura 3.6.

Figura 5.5: Diseño de la interfaz de DashboardActivity

El flujo de control desde esta actividad es el siguiente:

Figura 5.6: Diseño de la interfaz de HelpActivity

- Si se selecciona el interrogante, se procede a ir a 2 (HelpActivity).
- Si se selecciona Edición, se procede a 10 (StandardActivity).
- Si se selecciona Preferencias, se procede a 11 (BluetoothPreferencesActivity).
- Si se selecciona Acerca de, se procede a ir a 12 (AboutDialog).
- Si se selecciona Salir, se procede a salir del programa y apagar el servicio.

HelpActivity : Pantalla donde se muestra la ayuda de la aplicación. Se muestra en forma de páginas deslizables. Podemos ver el diseño en la figura 3.7.

El flujo de control es el siguiente:

- Si se selecciona la casa, se vuelve a 1 (DashboardActivity).
- Si se pulsa la flecha atrás, se vuelve a la pantalla anterior.

BatteryActivity : Es la actividad que define la regla de batería. En ella se puede introducir el nombre de la regla y el umbral de batería. Podemos ver su interfaz en la Figura 3.8.

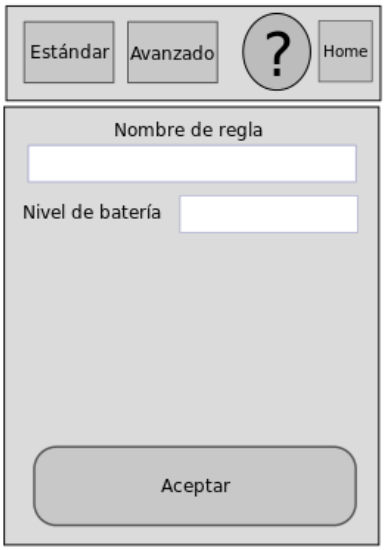


Figura 5.7: Diseño de la interfaz de BatteryActivity

El flujo de ejecución es el siguiente:

- Si se pulsa Estándar, se vuelve a 10 (StandardActivity).
- Si se pulsa Avanzado, se pasa a 4 (AdvancedActivity).
- Si se selecciona el interrogante, se procede a ir a 2 (HelpActivity).
- Si se selecciona Home, se irá a 1 (DashboardActivity).
- Si se pulsa aceptar, se guardará la regla y se volverá a 10 (StandardActivity).

AdvancedActivity : Es la pantalla de creación/edición de reglas avanzadas. En ella aparecen botones de ayuda para escribir el micro-lenguaje de definición de reglas, así como un desplegable con las reglas y un campo para escribir el nombre de la regla compuesta. También puede escribirse el lenguaje directamente con el teclado desde el editor de texto incluido dentro de la actividad. Podemos ver la interfaz en la Figura 3.9.

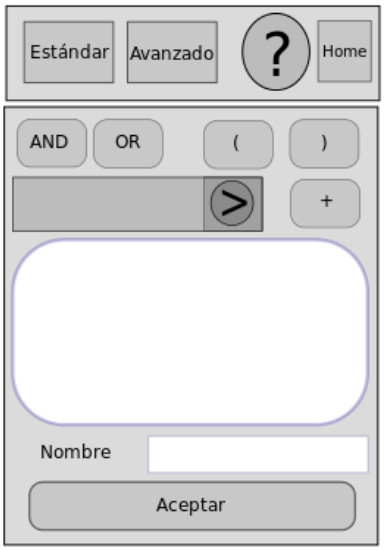


Figura 5.8: Diseño de la interfaz de AdvancedActivity

El flujo de control es el siguiente:

- Si se pulsa Estándar, se vuelve a 10 (StandardActivity).
- Si se pulsa Avanzado, se pasa a 4 (AdvancedActivity).
- Si se selecciona el interrogante, se procede a ir a 2 (HelpActivity).

- Si se selecciona Home, se irá a 1 (DashboardActivity).
- Si se pulsa aceptar, se guardará la regla y se volverá a 10 (StandardActivity).
- Si se pulsa AND, se añade la palabra reservada 'and' al editor de texto.
- Si se pulsa OR, se añade la palabra reservada 'or' al editor de texto.
- Si se pulsa '(' se añade el símbolo reservado '(' al editor de texto.
- Si se pulsa ')' se añade el símbolo reservado ')' al editor de texto.
- Si se pulsa '+' se añade el nombre de la regla del desplegable al editor de texto.

LocationActivity : Es la pantalla en la que se crea la regla de lugar. En ella aparece una lista de lugares conocidos, de la cuál se ha de escoger uno para la regla. Si no hay lugares conocidos, se deberán añadir en AddLocationActivity.

Figura 5.9: Diseño de la interfaz de LocationActivity

El flujo de control de esta actividad es el siguiente:

- Si se pulsa Estándar, se vuelve a 10 (StandardActivity).
- Si se pulsa Avanzado, se pasa a 4 (AdvancedActivity).
- Si se selecciona el interrogante, se procede a ir a 2 (HelpActivity).

- Si se selecciona Home, se irá a 1 (DashboardActivity).
- Si se pulsa aceptar, se guardará la regla y se volverá a 10 (StandardActivity).
- Si se pulsa +, se irá a 6 (AddLocationActivity).

AddLocationActivity : Es la actividad donde puede seleccionarse un lugar y añadirlo a la lista de lugares conocidos.

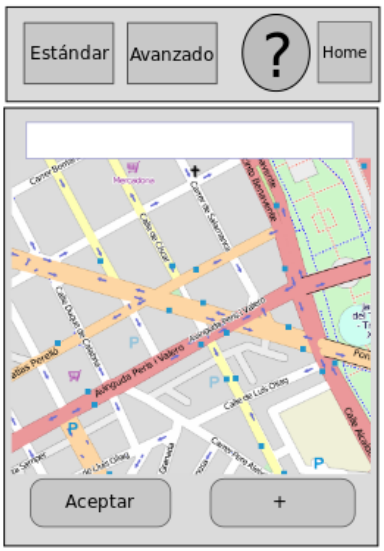


Figura 5.10: Diseño de la interfaz de AddLocationActivity

El flujo de control de esta pantalla es:

- Si se pulsa Estándar, se vuelve a 10 (StandardActivity).
- Si se pulsa Avanzado, se pasa a 4 (AdvancedActivity).
- Si se selecciona el interrogante, se procede a ir a 2 (HelpActivity).
- Si se selecciona Home, se irá a 1 (DashboardActivity).
- Si se pulsa aceptar, se guarda el lugar y se vuelve a 5 (LocationActivity).

ClockRuleActivity : Se trata de la pantalla para crear reglas de tiempo. En ella pueden añadirse rangos de días y/o horas, además de días de la semana donde será efectiva la activación del Bluetooth. También hay configuraciones predefinidas que rellenan las opciones automáticamente.

Figura 5.11: Diseño de la interfaz de ClockRuleActivity

- Si se pulsa Estándar, se vuelve a 10 (StandardActivity).
- Si se pulsa Avanzado, se pasa a 4 (AdvancedActivity).
- Si se selecciona el interrogante, se procede a ir a 2 (HelpActivity).
- Si se selecciona Home, se irá a 1 (DashboardActivity).
- Si se pulsa Aceptar, se vuelve a 10 (StandardActivity) pero habiendo añadido la regla.

StandardActivity : Se trata de la Activity de más importancia de toda la aplicación. Es la pantalla de creación de reglas estándar, y también una de las más complejas. En ella, se pueden crear, editar, y borrar reglas. La interfaz será como muestra la Figura 3.13.

Figura 5.12: Diseño de la interfaz de StandardActivity

La lógica de la actividad es la siguiente:

- Si se pulsa Estándar, se vuelve a 10 (StandardActivity).
- Si se pulsa Avanzado, se pasa a 4 (AdvancedActivity).
- Si se selecciona el interrogante, se procede a ir a 2 (HelpActivity).
- Si se selecciona Home, se irá a 1 (DashboardActivity).
- Si se selecciona el +, se abre un diálogo:
  - Si se selecciona regla de tiempo, se va a 9 (ClockRuleActivity).
  - Si se selecciona regla de lugar, se va a 5 (LocationActivity).
  - Si se selecciona regla de batería, se pasa a 3 (BatteryActivity).
  - Si se selecciona regla de dispositivo, se procede a 7 (DevicesActivity).
- Si se pulsa el -, se eliminará la regla que anteriormente haya sido seleccionada.
- Si se pulsa Editar, se abrirá el diálogo de edición de la correspondiente regla.
- Si se pulsa la flecha, se oculta parte del panel teniendo que utilizar los menús contextuales y de opciones de Android.

- Si se pulsa ON/OFF se cambia la prioridad de las reglas de Bluetooth.

- Si se pulsa Bt, se conecta o desconecta el Bluetooth independientemente de las reglas.

BluetoothPreferencesActivity : Se trata de la pantalla de opciones de la aplicación. Es la única que no dispone de 'ActionBar', y sigue el diseño de las pantallas de opciones de Android.

Figura 5.13: Diseño de la interfaz de BluetoothPreferencesActivity

- Si se selecciona Exportar/Importar, se seleccionará cuál de las dos opciones se realizará y se pasará a la Activity de

- Si se selecciona Refresco, saldrá un diálogo donde poner el tiempo de refresco.

AboutDialog : Se trata del diálogo donde aparecerá toda la información de 'Acerca de'.

Figura 5.14: Diseño de la interfaz de AboutDialog

- Si se pulsa aceptar se vuelve a 1 (DashboardActivity).

5.4. Gramática

Para la implementación de las reglas avanzadas, será necesario crear previamente una gramática válida. La podemos visualizar en la figura 5.15

start -> expr

expr -> expr2 or expr3 | expr2 and expr3 | not expr3 | expr2

expr2 -> ID | '(' expr ')'

expr3 -> expr

and -> 'and' | 'AND' | '&'

or -> 'or' | 'OR' | '|'

not -> 'not' | 'NOT' | '!'

Figura 5.15: Gramática del lenguaje de definición de reglas avanzadas

Capítulo 6

Implementación

En el presente capítulo se pasa a explicar y detallar las partes más importantes y resaltables del software desarrollado, así como las soluciones que he adoptado durante la fase de desarrollo de la aplicación y los problemas surgidos durante la misma.

6.1. Implementación de las reglas

Para implementar las reglas, se ha creado una interfaz 'Regla' (Mirar Anexo I, Rule.java, página 60), de la cuál extenderán todas las demás reglas, teniendo que implementar el método 'checkRule' (Mirar Anexo I, fichero Rule.java, línea 5) , el cuál será el encargado de comprobar que la regla crea que debe activar o desactivar el adaptador Bluetooth. Eso permitirá tratar a todas las reglas de igual forma independientemente de cómo estén desarrolladas.

```
...
4 public interface Rule {
5 public boolean checkRule ( ) throws Exception ;
...
9 }
```

Figura 6.1: Fragmento de la interfaz Rule

En primer lugar vamos a hablar de la regla de batería (Véase en el Anexo I, BatteryRule.java, en la página 60). La implementación es muy sencilla, ya que sólo tomamos como referencia el umbral de batería escogido por el usuario. Entonces, la regla se activaría sólo si el valor de batería está sobre el umbral o está conectado el dispositivo al cargador.

```
...
11 public classBatteryRule implements Rule {
...
34 public boolean checkRule ( ) throws Exception {
...
44 if (GeneralLogic.BATTERY_LEVEL > minLevel | | isCharging ){
45     return true ;
46 } else {
47     return false ;
48 }
...
```

Figura 6.2: Fragmento de la clase BatteryRule.java

Para programar la regla de tiempo (En el Anexo I, correspondiente al fichero TimeRule.java, en la página 66), se comprueba que la fecha y hora actual esté dentro de los rangos establecidos, es decir, dentro del rango de fechas establecido, de horas y a su vez, que se encuentre en un día de la semana activo; se tiene en cuenta si el usuario ha decidido no especificar fechas u horas. Una vez comprobado esto, se mira si está activada la casilla del día de la semana en el que nos encontramos. Si todo eso es positivo, se activa el Bluetooth. Como parte destacable de la implementación, resalto el método checkRule (visible en el Anexo I, fichero TimeRule.java, línea 230).

```
...
7 public class TimeRule implements Rule {
...
230 public boolean checkRule() throws Exception{
...
238 if (nodate && !notime) {
239  if (enabled && compareHours(currentTime, startTime, endTime)) {
240   ret = true;
241  } else {
242   ret= false;
243  }
244 }else if(notime && !nodate){
245  if (enabled && compareDates(currentTime, startDate, endDate)) {
246   ret = true;
247  } else {
248   ret = false;
249  }
250 }else if(!notime && !nodate){
251  if (enabled && compareHours(currentTime, startTime, endTime) &&
compareDates(currentTime, startDate, endDate) ) {
252   ret = true;
253  } else {
254   ret= false;
255  }
256 }
...
```

Figura 6.3: Fragmento de la clase TimeRule.java

Para la regla de lugar (Anexo I , fichero PlaceRule.java en la página 63) hace falta detallar a mayor nivel, dada la complejidad alta con respecto a las otras reglas. Para poder geolocalizar al usuario, se utilizan los dos métodos posibles en Android: Mediante el GPS, más lento pero más preciso, y mediante red (Wifi o 3G) más rápido pero con mayor error. Para geolocalizar puntualmente el lugar que queramos, necesitaremos la geolocalización por GPS ya que el error de posición es menor, y se realiza dentro de la Activity AddLocationActivity.java (Anexo I, página 29), independizando a la regla de cómo se obtienen sus datos.

```
...
22 public class AddLocationActivity extends ActionBarActivity{
...
32 public void onCreate(Bundle savedInstanceState) {
...
/*Obtenemos el manejador*/
42 manager = (LocationManager) getSystemSer-
vice(Context.LOCATION_SERVICE);
...
/*Miramos si está inicializado el de satélite*/
45 if(manager.isProviderEnabled(LocationManager.GPS_PROVIDER)){
46 locationProvider = LocationManager.GPS_PROVIDER;
/*Si no, miramos si está inicializado el de red*/
47 }else if(manager.isProviderEnabled(
        LocationManager.NETWORK_PROVIDER)){
48 locationProvider = LocationManager.NETWORK_PROVIDER;
49 }
50 if(!locationProvider.equals("")){
51 location = new myLocationListener();
/*Tiempo de actualización*/
52 manager.requestLocationUpdates(locationProvider, 30000, 20, location);
53 }else{
/*Si no hay ninguna manera de geolocalizar, lo notificamos al usuario*/
54 Toast t = Toast.makeText(getApplicationContext(), getResources().
    getString(R.string.alert_map), Toast.LENGTH_LONG);
55 t.show();
56 }
...
```

Figura 6.4: Fragmento de AddLocationActivity.java, referenciando la geolocalización

Sin embargo, en el caso de la geolocalización del servicio principal, primero intentaremos utilizar la geolocalización por red, dado que consume menos batería que el GPS, y si por algún motivo no estuviera disponible, se pasaría a geolocalizar usando satélite. Está implementado en la clase StandardActivity.java (Anexo I, página 7).

```
...
43 public class StandardActivity extends ActionBarActivity {
...
114 private void startGPSListener() {
/*Obtenemos el manejador*/
115 manager = (LocationManager) getSystemService(
Context.LOCATION_SERVICE);
116 listener = new myLocationListener();
/*Miramos si está inicializado el de red*/
117 if(manager.isProviderEnabled(LocationManager.
NETWORK_PROVIDER)){
118 locationProvider = LocationManager.NETWORK_PROVIDER;
/*Si no, inicializamos el de satélite*/
119 }else if(manager.isProviderEnabled(LocationManager.
GPS_PROVIDER)){
120 locationProvider = LocationManager.GPS_PROVIDER;
121 }
122 if(!locationProvider.equals("")){
123 manager.requestLocationUpdates(locationProvider, 30000,
5, listener);
124 }else{
125 Toast t = Toast.makeText(getApplicationContext(), getResources().
getString(R.string.alert_map), Toast.LENGTH_LONG);
126 t.show();
}
}
...
```

Figura 6.5: Fragmento de StandardActivity.java, referenciando la geolocalización

En segundo lugar, he decidido añadir un mapa de apoyo que sirva al usuario para comprobar visualmente que ha sido geolocalizado correctamente. Para ello, he escogido la tecnología de OpenStreetMaps dado que ya conocía Google Maps, y eso me permitía aprender una nueva manera de realizar mapas para la plataforma Android.

Ahora, conociendo los datos y la posición del teléfono, podemos calcular la distancia en metros de un punto en concreto (uno solo por cada regla) para conocer su proximidad y activar o desactivar el Bluetooth.

```
...
135 mapController.animateTo(new GeoPoint (GeneralLogiccurrentPosition.getLatitude() , GeneralLogic.currentPosition. getLongitude () ) ) ;
...
```

Figura 6.6: Fragmento de AddLocationActivity donde se localiza el mapa

La regla de dispositivos, por desgracia, no ha sido posible implementarla. El motivo ha sido que para poder hacer un barrido de los dispositivos cercanos (operación, por cierto, muy costosa) es necesario que el dispositivo esté en modo visible, y para ello tenemos que solicitar el permiso de visibilidad, de manera que se pregunta al usuario si desea estar visible, y además es por tiempo limitado (En algunos dispositivos ni llega a los cinco minutos), imposibilitando la forma de hacerlo en segundo plano y de manera continua. Otra opción sería tener un dispositivo con el usuario root activado, pero se descarta pues quedaría en manos del propio usuario y está fuera de nuestros objetivos.

La comprobación de las reglas se hace mediante un servicio en segundo plano, el cuál está gran parte del tiempo dormido (Evitando así consumir energía en exceso), y se despierta periódicamente para comprobar una a una la validez de las reglas. Como se ha ido explicando anteriormente, en caso de tener dos o más reglas que entran en conflicto entre ellas en el mismo momento, existe la prioridad general definida por el usuario, que escoge si activar el adaptador Bluetooth. La presencia de éste servicio, además de comprobaciones condicionales, puede exigir la activación y desactivación del GPS si no está disponible la geolocalización por red. El lugar donde se realiza todo esto es en BluetoothService.java (Anexo I, página 169).

```
...
25 public class BluetoothService extends Service implements Runnable {
...
96 public void run() {
...
105 switch (GeneralLogic.ruleMode) {
/*Modo de prioridad al apagado*/
106 case Constants.RULE_MODE_AND:
107 BluetoothState = false;
/*Recorremos las reglas*/
108 for (Rule r : GeneralLogic.rules) {
/*Si la regla está habilitada la tenemos en cuenta*/
109 if (r.isEnabled()) {
/*Si es la primera regla, guardamos su valor*/
110 if(i==0){
111 BluetoothState = r.checkRule();
112 set = true;
113 }else{
114 BluetoothState = BluetoothState && r.checkRule();
115 set = true;
116 }
117 i++;
118 }
119 }
120 break;
121 case Constants.RULE_MODE_OR:
122 BluetoothState = false;
123 i = 0;
/*Recorremos las reglas*/
124 for (Rule r : GeneralLogic.rules) {
/*Si la regla está habilitada la tenemos en cuenta*/
125 if (r.isEnabled()) {
/*Si es la primera regla, guardamos su valor*/
126 if(i == 0){
127 BluetoothState = r.checkRule();
128 set = true;
129 }else{
130 BluetoothState = BluetoothState || r.checkRule();
131 set = true;
132 }
133 i++;
134 }
135 }
136 break;
137 }
...
```


Figura 6.7: Fragmento de BluetoothService.java

6.2. Exportar e importar reglas.

Otro problema surgido fue en la exportación de reglas en formato .zip. El SDK de Android no incluye ningún diálogo ni ninguna herramienta de navegación de archivos como los típicos abrir y guardar cómo, de manera que fue necesario programar un navegador de archivos propio. Para ello, utilizando de base el tutorial de [Android-er2010], se creó un explorador de archivos adaptado para las necesidades de la aplicación. De esta manera, se puede exportar a la tarjeta SD un zip que incluye tanto las reglas como los lugares guardados.

6.3. Reglas avanzadas

Las reglas avanzadas (Anexo I, fichero ComplexRule.java, página 76) se basan en un mini-lenguaje de definición de reglas, el cuál se basa en interconectar las reglas existentes mediante precedencia por paréntesis, AND lógicos y OR lógicos. Para crear el intérprete del mini-lenguaje, se ha optado por utilizar la herramienta ANTLR, debido a su buena integración con el lenguaje Java y con el sistema Android, y el entorno de desarrollo ANTLRWorks. Para crear la gramática, hay que tener en cuenta que no debe ser recursiva a izquierdas haciendo las transformaciones pertinentes en su caso.

Las reglas compuestas pueden ser guardadas en cualquier caso, pero para funcionar deberán ser validadas como correctas.

Para el uso de ANTLR es necesario añadir un entorno de ejecución especial para sistemas Android (obtenido de la web oficial de ANTLR), ya que las librerías tradicionales son demasiado pesadas. Para ello, obtenemos el código fuente y lo incluimos dentro del proyecto.

Lo que se guarda en el fichero XML de definición de regla, es el texto tal cuál del editor, procesándolo cuando sea necesario.

```
...
9 public class ComplexRule implements Rule {
...
24 public boolean checkRule() throws Exception {
...
31 ANTLRInputStream input = new ANTLRInputStream(
32 new ByteArrayInputStream(rule.getBytes("UTF-8")));
/*Creamos el analizador léxico*/
33 BluetoothRuleLexer lexer = new BluetoothRuleLexer(input);
34 CommonTokenStream tokens = new CommonTokenStream(lexer);
/*Creamos el parser sintáctico*/
35 BluetoothRuleParser parser = new BluetoothRuleParser(tokens);
/*Recogemos el valor obtenido por el parser sintáctico*/
36 ok = parser.start();
...
```

Figura 6.8: Fragmento de ComplexRule.java donde se lanza el procesador de lenguajes

6.4. Interfaz gráfica

Para crear la interfaz gráfica, se ha utilizado tanto material propio como de otros diseñadores bajo licencia Creative Commons. Para el material propio (El diseño general de la aplicación) se ha utilizado tanto GIMP como Inkscape. Los iconos (Excepto el de la aplicación) han sido obtenidos de IconFinder, filtrando la licencia como "Permitido para uso comercial, sin necesidad de poner el enlace".

Para el aspecto se ha utilizado tanto gráficos png como estilos creados en formato xml, de manera que se consigue ahorrar el peso de la imagen pero se obtienen degradados, bordes redondeados... etcétera. Un ejemplo de esto sería el fichero dashboard_background.xml (Anexo I, Página 177).

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <shape xmlns:android="http://schemas.android.com/apk/res/android" >
3<gradient
4 android:angle="90"
5 android:centerY="0.5"
6 android:endColor="#04B4AE"
7 android:startColor="#0B615E" />
8 <corners android:radius="15dp" />
9
10 <padding
11 android:bottom="7dp"
12 android:left="7dp"
13 android:right="7dp"
14 android:top="7dp" />
15
16<margin
17 android:bottom="3dp"
18 android:left="3dp"
19 android:right="3dp"
20 android:top="3dp" />
21
22</shape>
```

Figura 6.9: dashboard_background.xml

6.5. Sección de ayuda.

Para la sección de ayuda, se ha escogido utilizar un 'ViewPager', un tipo de componente que muestra diferentes contenidos divididos en páginas, a las cuales se puede acceder deslizando el dedo/puntero hacia la izquierda o la derecha. A la hora de utilizar este componente, se han presentado dos problemas principales: La compatibilidad y el diseño de la interfaz.

En lo que respecta a la compatibilidad, esta vista no es compatible de por sí con el SDK de Android, y es necesario añadir una librería de compatibilidad (proporcionada por Google). Esta librería se llama 'android-support-v4.jar', y una vez incluida en el proyecto ya podremos utilizar el ViewPager. Su uso se puede ver en el Anexo I , fichero help.xml, página 221.

```
1 <?xml version="1.0" encoding="utf-8"?>
...
19 <android.support.v4.view.ViewPager
20 android:id="@+id/vwpg_help"
21 android:layout_width="fill_parent"
22 android:layout_height="fill_parent" >
23 </android.support.v4.view.ViewPager>
...
```

Figura 6.10: help.xml, con la vista ViewPager

El segundo problema, y más laborioso de solucionar, es el hecho de diseñar e implementar la interfaz del ViewPager. El principal problema que tenemos es que el ViewPager no está integrado con el diseñador de interfaces gráficas del entorno de desarrollo, y es necesario definir a código java todo aquello que queramos que esté incluido dentro del componente. Un ejemplo de ello lo podemos ver en la figura 6.11, la cuál es un extracto de la clase HelpActivity.java (Anexo I, página 53).

```
...
20 public class HelpActivity extends ActionBarActivity {
...
107 /*
108 * Creamos el layout1
109 */
110 LinearLayout layout = new LinearLayout(this);
111 LinearLayout scrollChild = new LinearLayout(this);
...
120 /*Creamos una vista de scroll, para adaptarlo a pantallas pequeñas*/
121 ScrollView scroll1 = new ScrollView(this);
/*Creamos el texto a mostrar*/
122 TextView explanation1 = new TextView(this);
123 explanation1.setTextSize(TypedValue.COMPLEX_UNIT_DIP, 17);
124 explanation1.setText(Html.fromHtml(
125 "<b>"+getResources().getString(
R.string.header_explanation1)+"</b><br>"));
...
138 layout.addView(explanation1);
139 scrollChild.addView(textOne);
...
142 scroll1.addView(scrollChild);
143 layout.addView(scroll1);
144 layouts.add(layout);
...
```

Figura 6.11: Fragmento de HelpActivity.java

Capítulo 7

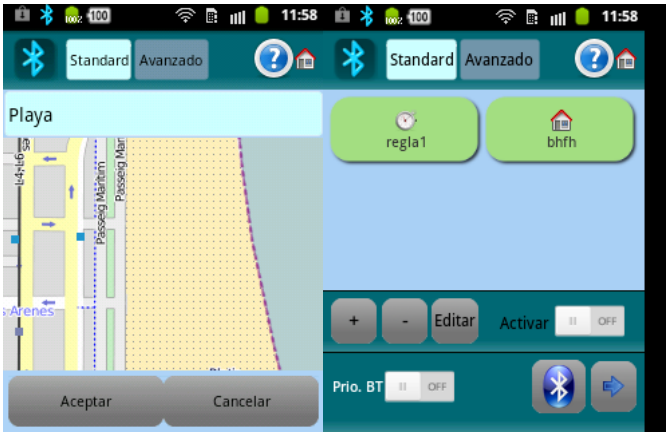
Ejemplos de funcionamiento

A continuación, mostraremos ejemplos de funcionamiento de nuestra aplicación, mostrando su correcto funcionamiento y la interfaz definitiva.

Probaremos a añadir una regla de tiempo, y guardarla:

Figura 7.1: Añadir regla de tiempo

Pasamos a realizar lo mismo con una regla de lugar:



Cuadro 7.1: Añadir regla de lugar

Ahora, añadimos una regla de batería:

Figura 7.2: Añadir regla de batería

Tras añadir las tres reglas, se comprueba que en prioridad a apagado, el Bluetooth está activo hasta pasada la hora establecida. Una vez hecho esto, se cambia a prioridad de encendido, y se observa que el adaptador Bluetooth vuelve a activarse.

Ahora, desactivaremos todas las reglas, y haremos una regla avanzada relacionándolas:

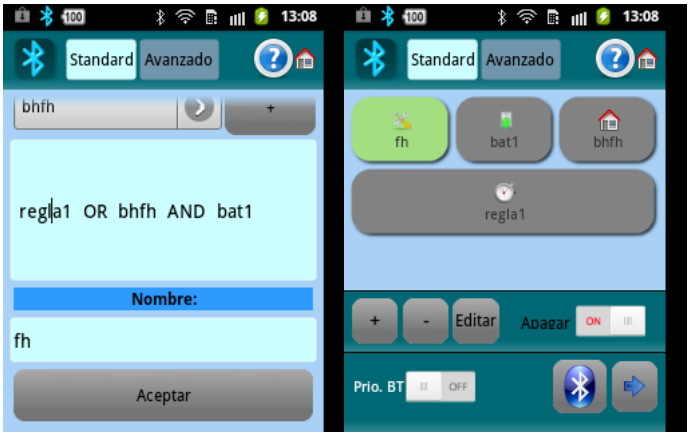


Figura 7.3: Añadir regla avanzada

El comportamiento es el esperado, ya que la batería es superior al umbral, y además, estamos en el lugar seleccionado.

Ahora vamos a proceder a comprobar el funcionamiento del explorador de archivos, así como de la importación y exportación de las reglas. Para ello, primero exportamos a un archivo en formato zip, para borrar todas nuestras reglas, y después importarlo de nuevo. Observamos que, efectivamente, las reglas se han restaurado.

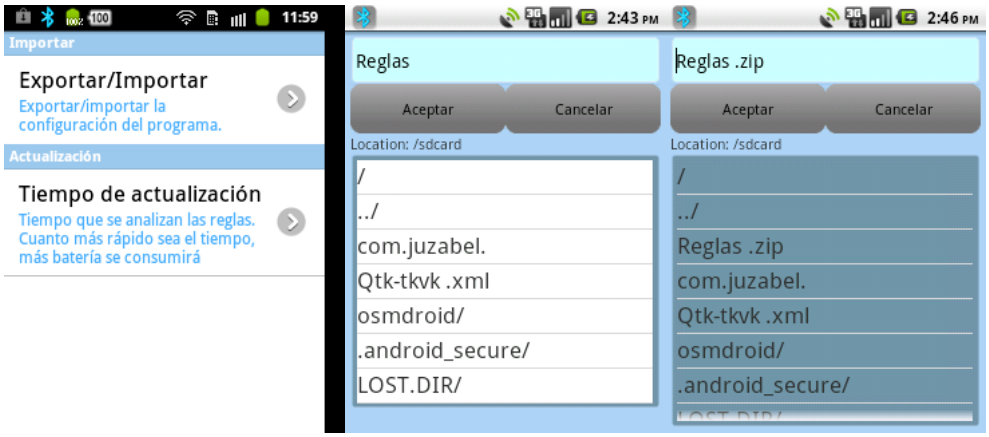


Figura 7.4: Exportación e importación de reglas

Para terminar la comprobación del funcionamiento de las partes del programa más importantes, pasamos simplemente a mostrar la ayuda de la aplicación en el orden en el cuál se muestra al usuario.

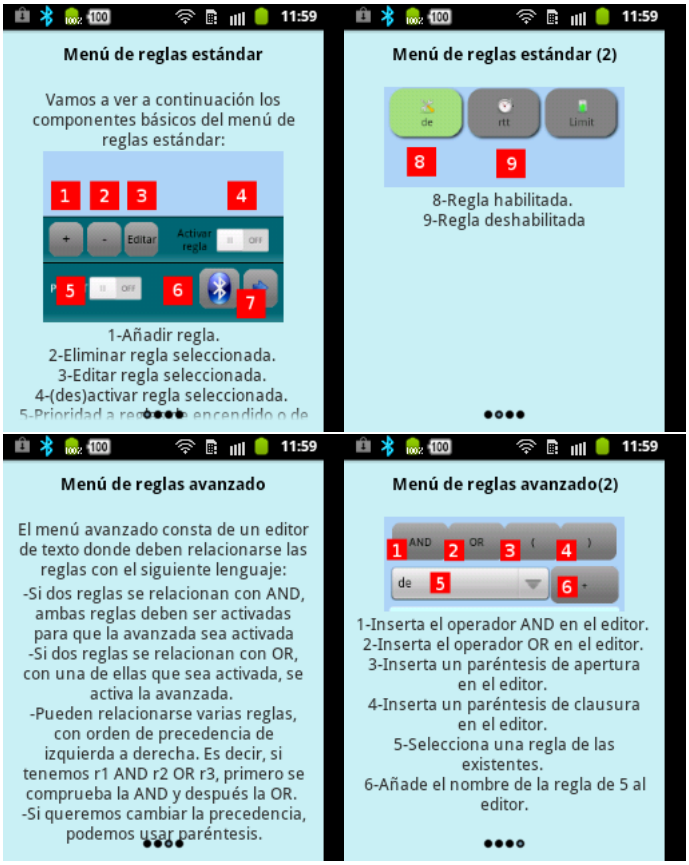


Figura 7.5: Ayuda de la aplicación

Capítulo 8

Conclusiones

Para finalizar, podemos observar que hemos conseguido realizar una aplicación de gestión de Bluetooth basada en reglas. El usuario, puede ser capaz de transmitir la manera en la que la aplicación mediante reglas simples o complejas, las cuáles se evalúan en conjunto propiciando una activación o desactivación del adaptador Bluetooth del dispositivo Android.

Las reglas básicas que se han conseguido implementar correctamente han sido:

- Regla de tiempo: Es capaz de comprobar los intervalos de tiempo definidos por el usuario, tanto de fechas como de horas.
- Regla de proximidad : Consigue añadir ubicaciones propias, así como detectar la proximidad a estas.
- Regla de batería: Detecta el umbral de batería , y si el dispositivo se encuentra conectado a la corriente y cargando.

Por otra parte la regla de dispositivos conocidos no pudo ser desarrollada, ya que los permisos necesarios para ello eran propios del superusuario.

En las reglas avanzadas, se ha conseguido crear un lenguaje de definición de reglas basado en conexiones lógicas entre los resultados de éstas. Para ello, he aprendido a utilizar la herramienta ANTLRWorks, entorno de desarrollo para ANTLR. Si la regla no está escrita según el lenguaje de definición de reglas, se detecta que existe un error y no es activada.

También se ha implementado un sistema de prioridades, en el cuál el usuario puede elegir si ante un conflicto de reglas, se da prioridad al encendido o al apagado.

Por lo que respecta a la IGU, se ha conseguido una interfaz limpia, la cuál aprovecha las opciones que ofrece Android (Menús contextuales y

de opciones) ofreciendo además un panel donde puede realizase igualmente cualquier acción (Favoreciendo a los usuarios con poco conocimiento del propio sistema operativo).

También se ha incluido un apartado de ayuda en forma de páginas, para que el usuario pueda consultar cualquier duda acerca del software.

Uno de los puntos fuertes de la aplicación puede ser el diseño, el cuál se ha alejado de un aspecto 'por defecto' de las aplicaciones Android, tanto utilizando material propio como iconos de libre utilización.

Para el almacenamiento de las reglas, se ha conseguido crear un formato diferente para cada una utilizando XML, y también se ha conseguido exportarlas e importarlas a un archivo externo como copia de seguridad.

Debido a motivos de tiempo, hay objetivos que se han quedado total o parcialmente fuera del alcance del presente proyecto. Estos objetivos son:

- Un estudio más a fondo del ahorro de batería conseguido por la aplicación.
- Realización de casos de test y pruebas en la aplicación, utilizando herramientas como JUnit o similares.

También podría haber sido interesante, haber ampliado los objetivos:

- Un sistema de prioridades de regla más avanzado.
- Almacenamiento de las reglas en un servidor ("en la nube").

Capítulo 9

Definiciones

1. Action Bar: Patrón de diseño para aplicaciones Android basado en la utilización de una barra superior común a toda la aplicación, dentro de la cual hay elementos de navegación como botones, iconos... etcétera.

2. Google Play: Aplicación presente en los dispositivos Android (Anteriormente llamada Android Market) en la cuál es posible descargar, gratuitamente o bajo pago, aplicaciones desarrolladas por Google, otras empresas, o desarrolladores independientes. A partir de su cambio a Google Play, además de aplicaciones, también pueden encontrarse libros y películas, y es posible que en el futuro puedan añadirse nuevos productos.

3. Piconet: Según [Wikipedia2012b], una piconet es una red de dispositivos informáticos conectados mediante la tecnología Bluetooth. Una piconet puede constar de dos hasta ocho dispositivos, en los que uno será el maestro y el resto esclavos.

4. Middleware: Según [Rfidpoint2012], el middleware es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

5. Dashboard : Patrón de diseño para Android que se basa en una actividad principal de la aplicación con botones/iconos de acceso a las partes más importantes de la misma.

Bibliografía

[Condesa2011] Condesa. *Arquitectura de Android.* (Julio 2011). Disponible en: http://androideity.com/2011/07/04/arquitectura-de-android/

[ADevelopers2012] Android Developers. *Activities.* (Julio 2012). Disponible en: http://developer.android.com/guide/components/activities.html

[ADevelopers2012a] Android Developers. *Android, the world's most popular mobile platform.* (Julio 2012). Disponible en: http://developer.android.com/about/index.html

[Wikipedia2012] Wikipedia. *Bluetooth.* (Junio 2012). Disponible en: http://developer.android.com/about/index.html

[Wikipedia2012a] Wikipedia. *Bluetooth profiles.* (Julio 2012). Disponible en : http://en.wikipedia.org/wiki/Bluetooth_profile

[Wireless2000] Wireless. *Bluetooth Range in Relation to Different Power Classes.* (Marzo 2000). Disponible en: http://www.palowireless.com/infotooth/knowbase/general/10.asp

[ADevelopers2012b] Android Developers. *Application Fundamentals.* (Julio 2012). Disponible en : http://developer.android.com/guide/components/fundamentals.html

[Android-er2010] Android.er. *Implement a simple File Explorer in Android* . (Enero 2010). Disponible en: http://android-er.blogspot.com.es/2010/01/implement-simple-file-explorer-in.html

[Wikipedia2012b] Wikipedia. *Piconet.* (Marzo 2011). Disponible en: http://es.wikipedia.org/wiki/Piconet

[Rfidpoint2012] RFIDPoint. *Middleware.* (Septiembre 2012). Disponible en : http://www.rfidpoint.com/fundamentos/middleware/