Proyecto de Sistemas Informáticos.
Curso 2008-2009.

INTELIGENCIA AMBIENTAL EN DISPOSITIVOS MÓVILES

Componentes del grupo:
Jesús Díaz Artiaga
Ezequiel Lara Gómez
Federico Gabriel Mon Trotti

Directores del proyecto:
Marco Antonio Gómez-Martín
Manuel Prieto Matías

Facultad de Informática.
Universidad Complutense de Madrid.

Inteligencia Ambiental en Dispositivos Móviles

Memoria de Proyecto Fin de Carrera presentada por Jesús Díaz Artiaga, Ezequiel Lara Gómez y Federico Gabriel Mon Trotti en la Universidad Complutense de Madrid, realizado bajo la dirección de Marco Antonio Gómez-Martín y Manuel Prieto Matías.

Madrid, a 3 de Julio de 2009.

Prefacio

En este proyecto desarrollamos una aplicación que introduce inteligencia ambiental en dispositivos móviles, esto es, hacer que el teléfono utilice la información del contexto en que se halla (hora, localización, conectividad, etc.) para automatizar tareas cotidianas que el usuario realiza cuando se cumplen esas condiciones. En primer lugar hacemos un estudio del estado del mercado y de las diferentes plataformas disponibles para desarrollar, discutiendo la mejor opción para llevar a cabo el proyecto, para decidirnos finalmente por Google Android. A continuación analizamos las diferentes técnicas de Inteligencia Artificial que pueden ser aplicadas para mejorar el funcionamiento del sistema, las ventajas que se obtienen con cada una, y las dificultades de aplicar este tipo de algoritmos en dispositivos móviles. Posteriormente detallamos las fases de análisis, diseño e implementación de nuestro proyecto, y el funcionamiento del prototipo final. Finalmente exponemos el alcance del proyecto y las futuras líneas de trabajo que quedan abiertas.

Abstract

In this project we develop an application that introduces ambient intelligence in mobile devices, which means to have the phone use the information on the context in which it is present (time, location, connectivity, etc.) in order to automate everyday tasks that the user does when in those conditions. First, we study the contemporary market and the different platforms available in which to develop, discussing the best option for completing the project, to finally choose Google Android. Afterwards we analyze diferent Artificial Intelligence techniques that can be implemented to improve system performance, the advantages to be gained with each one, and the difficulties of implementing such algorithms on mobile devices. Later, we detail the different stages of analysis, design and implementation of our project, and the operation of the final prototype made. Finally we present the project scope and future work lines to pursue.

Keywords

Google Android, mobile, positioning, GPS, context, ambient intelligence, association rule learning

El código fuente del proyecto se encuentra disponible bajo la licencia GPL versión 2, descargable de `http://www.gnu.org/licenses/gpl-2.0.txt`. Por su parte, la memoria del mismo está disponible bajo la licencia GFDL versión 1.3, a su vez descargable de `http://www.gnu.org/licenses/fdl-1.3.txt`.

Sin perjuicio de lo anterior, Jesús Díaz Artiaga, Ezequiel Lara Gómez y Federico Gabriel Mon Trotti, los autores del proyecto y abajo firmantes autorizamos además a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Madrid, a 3 de Julio de 2009.

Jesús Díaz Ezequiel Lara Federico Mon

Agradecimientos

A Linus Torvalds y Richard Stallman, porque sin ellos nada de esto habría sido posible.

A Eclipse, ese editor con personalidad, y a Bazaar, causa y a la vez solución de tantos problemas.

A nuestros familiares, amigos y allegados, por soportarnos durante tantas horas de desarrollo.

A Tino por ayudar con la memoria.

A Elena, Maribel y Luis por echar una mano revisando la memoria.

A la vida, que nos ha dado tanto.

Índice general

1. Introducción **1**

1.1. Objetivo inicial 1

1.2. Objetivos alcanzados 2

1.3. Estructura del documento 4

2. Estado del Arte **7**

2.1. Avances en el hardware 7

2.2. Avances en el software 8

2.3. Análisis de la competencia 11

3. Análisis **13**

3.1. Casos de uso básicos 13

3.2. Análisis de las plataformas 14

4. Diseño **21**

4.1. Arquitectura preliminar 21

4.2. Arquitectura final 23

4.2.1. Conceptos y funcionamiento básicos 23

4.3. Inteligencia Artificial 25

4.3.1. Inferencia de reglas 25

4.3.2. Algoritmos bayesianos 27

5. Implementación **29**

5.1. Herramientas utilizadas 29

5.1.1. Entorno de desarrollo - Eclipse 29

5.1.2. Control de versiones - Bazaar 32

5.1.3. Memoria del proyecto 32

5.2. Desarrollo sobre la plataforma Android 34

5.3. Estructura de la aplicación 36

5.3.1. Estructura de componentes 36

5.3.2. Estructura de clases de Regla 38

5.3.2.1. Eventos 39
5.3.2.2. Reacciones 41
5.4. Funcionamiento del prototipo 42

6. Conclusiones **51**
6.1. Alcance 51
6.2. Líneas futuras de trabajo 52
6.2.1. Mejoras en la estructura del código 52
6.2.2. Mejoras en la gestión de reglas 53
6.2.3. Otras mejoras y pruebas 54
6.2.4. Técnicas de Inteligencia Artificial 54

A. Plataforma Android **I**

Bibliografía **VII**

Índice de figuras **XV**

Índice de tablas **XVII**

Capítulo 1

Introducción

1.1. Objetivo inicial

El teléfono móvil se ha convertido en un periodo de pocos años en una parte fundamental de la sociedad, llegando incluso al hecho de que en España existen más líneas de telefonía móvil que habitantes [1]. Desde sus primeras funciones básicas de realizar llamadas y poder estar localizable, ha pasado a ser un aparato indispensable en la vida diaria de la mayor parte de la población, con funcionalidades como agenda personal, calendario, o incluso reproductor de música y cámara de fotos. Asimismo, de un tiempo a esta parte la tecnología ha posibilitado capacidades en los terminales hasta ahora inimaginables, como conexión a Internet de alta velocidad, geoposicionamiento, o la propia capacidad de cálculo con la que cuentan.

Sin embargo, el software presente en los terminales no ha evolucionado al mismo ritmo que el hardware; aunque hay avances en el campo de la usabilidad [2], es un campo mayormente aún por explorar.

En nuestro proyecto partimos de la idea de aprovechar este objeto tan presente en nuestras vidas, para dotarle de mayor *inteligencia ambiental*, esto es: hacer al software del teléfono más consciente de su entorno para permitirle integrarse de un modo transparente en la vida del usuario, evitando causar situaciones indeseadas (que el teléfono suene en el cine o en el centro de trabajo, por ejemplo), o ayudando al usuario en su vida diaria, programando acciones rutinarias y potenciando con ello las capacidades de organizador personal que tiene el terminal (por ejemplo, recordando realizar cierta llamada en un momento determinado).

Para conseguir este propósito, la idea es usar el contexto disponible en el teléfono móvil, es decir: la hora que es, el lugar en el que nos encontramos, la lista de contactos

y aquellos que el móvil detecta mediante *bluetooth*, etc. Con esa información se puede hacer que el teléfono realice tareas automáticas programadas por el usuario como, por ejemplo, silenciar el teléfono al llegar al trabajo, o guardar la localización GPS cuando el teléfono se desconecta del manos libres del coche (para saber dónde hemos aparcado).

También consideramos que sería muy interesante que el propio teléfono deduzca estas respuestas en base al comportamiento cotidiano del usuario, ya que también conoce el contexto en que se halla en el instante en que el usuario interactúa con él. Al usar nuestro teléfono realizamos continuamente tareas repetitivas según ciertas condiciones de contexto, por lo que la aplicación podría aprender los patrones de comportamiento del usuario y a partir de ellos sugerirle que el propio terminal realice esa tarea repetitiva por él, como si fuera otra de las acciones automáticas que el usuario ha programado. Un claro ejemplo de esto, basado en los que hemos expuesto anteriormente, sería el de un usuario que cada mañana cambia el modo de teléfono a silencio al llegar a su oficina. La aplicación podría detectar que ese patrón se repite al llegar a un determinado lugar (con un cierto margen de tolerancia), y sugerirle quitar el sonido siempre en esas condiciones.

Durante nuestra fase de definición de la idea inicial observamos también que en varias empresas se empezaba a tener en consideración el concepto citado de inteligencia ambiental, o teléfonos que respondan mejor al contexto que detectan: Intel, por ejemplo, cuenta con varios grupos de investigación y desarrollo [3], incluyendo un acuerdo con Nokia [4], que persiguen objetivos similares a los nuestros. Sus casos de uso son tan ambiciosos como, por ejemplo, integrar el teléfono móvil con proyectores disponibles en una sala de reuniones, de modo que al llegar a la misma, si hay una reunión planificada en la agenda al depositar el teléfono sobre la mesa sincronizase y autenticase directamente contra el proyector vía inalámbrica para mostrar una presentación.

Descubrir esto nos confirmó que el camino a seguir era prometedor y además estaba relativamente inexplorado, lo que hacía posible el impacto de nuestro proyecto siempre y cuando lográsemos un tiempo al mercado adecuado.

1.2. Objetivos alcanzados

Como se ha visto, partimos de unos objetivos iniciales muy ambiciosos, puesto que nuestra idea incluye diferentes frentes de desarrollo, desde el funcionamiento

básico de recogida de datos del teléfono, que no sabíamos cuánto trabajo nos podría llevar en función de la plataforma elegida, a la inclusión de algoritmos de inteligencia artificial para completar la aplicación, pasando por el resto de aspectos necesarios para que ésta sea funcional: una interfaz de usuario, almacenamiento de los datos, etc.

Una vez analizados los objetivos, es necesario también considerar las dificultades que se presentan ante un proyecto de estas características. En primer lugar, estamos ante tecnologías relativamente nuevas, en terminales móviles de última generación, por lo que hay poca o nula experiencia de desarrollo acumulada, tanto por parte de los propios diseñadores de la plataforma como por parte de la comunidad de desarrollo, lo que sin duda dificulta encontrar la solución a problemas comunes de implementación. Además, la novedad de las plataformas afecta a la inestabilidad de los interfaces de programación contra los que se programa, que pueden cambiar sustancialmente entre versiones. Son también muy reseñables las propias limitaciones que conlleva trabajar sobre un teléfono móvil: menor capacidad de procesamiento, memoria, importancia de reducir el consumo de batería, etc; lo que implica tener especial cuidado a la hora de implementar cualquier algoritmo costoso, como es nuestro caso en relación a la inteligencia artificial. Finalmente podemos destacar también la incertidumbre inicial al desconocer cuál es el punto de partida, ya que dependiendo de la plataforma elegida podíamos encontrarnos con que quizá nos veríamos en la obligación de implementar funcionalidades básicas en la recogida de información y de más bajo nivel del deseado, como por ejemplo los datos del GPS.

Afortunadamente, la elección de Android como plataforma de desarrollo (como se verá en detalle en la sección 5.2) nos facilitó enormemente el trabajo, ya que cuenta con un API –o Interfaz de Programación de Aplicaciones– muy completo para la recogida de la información de contexto del teléfono. Sin embargo, nuestra inexperiencia con esta plataforma de desarrollo, así como nuestro tiempo limitado para desarrollar el proyecto, sólo nos han permitido establecer un *framework* o marco de desarrollo básico sobre el que continuar, y que por tanto tiene un alcance e impacto más limitado del deseado.

En el prototipo hemos implementado la estructura fundamental de la aplicación, junto con los casos mínimos de prueba para soportar reglas estáticas y definidas por el usuario (esto es, reacciones que debe efectuar el sistema ante determinadas situaciones). La aplicación cuenta asimismo con una interfaz de usuario y un sistema de almacenamiento de esas reglas cuando el teléfono está apagado, para que la funcionalidad sea completa. Dentro de los contextos que se capturan hemos implementado los más representantivos (hora, localización, etc.) como ejemplos de uso de la aplicación, y hemos tenido siempre en cuenta la facilidad de añadir nuevos tipos de situaciones a

la hora de realizar el diseño, prefiriendo que fuese genérico en la medida de lo posible.

Por último, también hemos implementado la recogida de información del contexto en cada acción del usuario. Con ello posibilitamos las futuras extensiones (tratadas con más detalle en la sección futuras líneas de trabajo [6.2]), que serían las que utilizasen técnicas de inteligencia artificial para proponer al usuario nuevas reglas automáticamente generadas. El análisis de las diferentes técnicas que se podrían aplicar en nuestro proyecto se describen en el capítulo 4.3.

1.3. Estructura del documento

Este documento se divide en las siguientes secciones:

En el capítulo 2 tratamos el punto desde el que partimos en el desarrollo, examinando las plataformas de telefonía móvil existentes, tanto respecto del punto de vista del *hardware* existente como del *software* y mencionamos los diversos entornos de desarrollo o SDK existentes entonces. Asimismo incluimos un análisis de otras aplicaciones de similares características existentes en el momento en que comenzamos este desarrollo, así como sus diferencias con nuestros objetivos.

En el capítulo 3 detallamos los diferentes casos de uso a partir de los cuales diseñamos la aplicación. Además incluimos un análisis detallado de las plataformas de desarrollo contempladas en las primeras fases del proyecto, así como los motivos que nos llevaron a optar por Google Android en detrimento del resto.

En el capítulo 4 exponemos en primer lugar la especificación de la arquitectura de la aplicación, a alto nivel e independientemente de la plataforma elegida y, a continuación, cómo se amoldan esos conceptos preliminares a la plataforma de desarrollo concreta, con los cambios que conllevan en la arquitectura final. Incluimos además la investigación realizada sobre las diferentes técnicas de inteligencia artificial aplicables al proyecto.

Posteriormente, en el capítulo 5, detallamos las herramientas utilizadas, así como las particularidades del desarrollo sobre Android. Esto se verá ampliado en su anexo correspondiente (anexo A). A continuación entramos en detalle en la implementación del proyecto, con un análisis de más bajo nivel de las partes más relevantes de la arquitectura final, y una descripción final del prototipo y de su funcionamiento.

En el capítulo 6, el lector podrá encontrar un breve resumen final del proyecto y el alcance del mismo en el momento de la redacción del presente documento. También se detallan las líneas de trabajo que quedan abiertas para futuras ampliaciones de la aplicación.

Finalmente, se detalla la bibliografía utilizada (6.2.4), así como un anexo sobre la plataforma Android (anexo A).

Capítulo 2

Estado del Arte

2.1. Avances en el hardware

Desde hace algunos años [5] ha aparecido en los terminales la categoría de *smartphone*: teléfonos de funcionalidad similar a la de ordenadores personales, lo que implica una importante capacidad de cálculo y memoria. Tipicamente son *System-on-Chip* específicos, implementando el conjunto de instrucciones ARM y con diversos coprocesadores gráficos, de señal, etc. Esta capacidad de proceso, poco antes solo al alcance de ordenadores personales, posibilita el desarrollo de aplicaciones de usuario en los teléfonos verdaderamente complejas. La ley de Moore demuestra también en el campo de los dispositivos embebidos, una vez más, su imparable efecto en el avance de la tecnología.

También las capacidades de comunicación y transmisión de datos han mejorado mucho, con la expansión de las redes 3G y posteriores, que permiten un acceso rápido a Internet (tipicamente de hasta 1MB/s) [6] en cualquier parte . También las tarifas se han hecho más asequibles con la aparición de tarifas planas a precios razonables ([7] 10 euros + IVA, [8] 12 euros + IVA). Incluso algunos terminales comienzan a incorporar tecnologías como 802.11b/g (*wifi*), para un acceso de muy alta velocidad (en torno a 4 MB/s) [6] a la red.

Asimismo, en los terminales comienza a hacer una entrada tímida el GPS, hoy prácticamente en todos los terminales de gama alta e introduciéndose rápidamente en la gama media [9]. Esto permite al teléfono funcionalidades, además de determinar su localización geográfica, antes sólo presentes en dispositivos especializados, como por ejemplo creación de mapas geográficos, seguimiento de rutas o planificación de las mismas.

Otras características cada vez más frecuentes son pantallas grandes y táctiles, que simplifican la interacción con el usuario, acelerómetros y brújulas integradas para determinar gestos y la posición del teléfono (sobre una superficie, agitándose, en vertical u horizontal…), etc.

Todas estas capacidades presentes en hardware bastante asequible abren un mundo previamente inexistente de posibilidades para el desarrollo y nuevos posibles modelos de negocio y oportunidades a capitalizar.

Como ejemplo de las capacidades habitualmente presentes en un dispositivo Android [10] [11] sirva esta tabla:

	HTC G1 	**HTC Magic** 
Procesador	Qualcomm® MSM7201A™, 528 MHz	Qualcomm® MSM7200A™, 528 MHz
Memoria ROM	256 MB	512 MB
Memoria RAM	192 MB	288 MB
Dimensiones	117.7 x 55.7 x 17.1 mm	113 x 55.56 x 13.65 mm
Peso	158 gramos	116 gramos
Pantalla	Táctil, de 3.2 pulgadas	Táctil, de 3.2 pulgadas
Resolución	320 x 480 píxeles	320 x 480 píxeles
Redes	7.2Mbps(HDSPA), 2Mbps(HSUPA)	7.2Mbps(HDSPA), 2Mbps(HSUPA)
Teclado	Tipo *Slide*, QWERTY	
GPS	Navegación GPS con Google Maps	Antena GPS interna
Conectividad	Bluetooth 2.0, IEEE 802.11b/g	Bluetooth 2.0, IEEE 802.11b/g
Cámara	3.2 Mpx con autoenfoque	3.2 Mpx con autoenfoque
Batería	Ion-Litio, 1150mAh	Ion-Litio 1340mAh
Extra	Brújula digital, sensor de movimiento	Brújula digital, sensor G

TABLA 2.1: Modelos HTC con Android

2.2. Avances en el software

Los avances citados y la gran diversidad de plataformas hardware existentes no han tenido una correspondencia directa en la usabilidad y capacidades de los

sistemas operativos presentes en los teléfonos móviles. Aparte de aquellos específicos para teléfonos de gama baja/media, la categoría de *smartphone* cuenta con un número más limitado de plataformas de desarrollo que la enorme diversidad de modelos que hay, todas ellas fundamentalmente incompatibles entre sí. Partiendo de aquellas que soportasen adecuadamente las capacidades de hardware de las que precisamos, analizamos las diversas plataformas preexistentes con vistas a sus proyecciones de mercado.

Por un lado, bastante asentadas ya en sus respectivos segmentos de ventas:

Windows Mobile, de Microsoft [12]: la plataforma que introdujo el propio término *smartphone*, una auténtica veterana con un boyante mercado de aplicaciones para todos los nichos, muy integrada en entornos empresariales y con gran diversidad de terminales que la soportan. Sin embargo, su usabilidad es bastante mejorable.

Blackberry, de RIM [13]: cuenta con una fuerte presencia en el mundo empresarial. La descartamos a priori por desconocimiento de la plataforma, y por no ofrecernos ventajas respecto a alternativas similares como Windows Mobile, que presuponíamos más flexible por la veteranía. Por lo demás, su usuario tipo es un usuario de negocios como móvil de empresa, para quien no es tan útil nuestra aplicación como para alguien en su tiempo de ocio.

Symbian, de Nokia entre otros [14]: la primera en cuota de mercado [15] [16], propiedad de un consorcio controlado por Nokia, introducida inicialmente para terminales de gama alta pero hoy día presente en gran número de terminales mucho más destinadas al usuario casual o de ocio. Mientras estudiábamos la viabilidad de la plataforma se anunció la liberación del código fuente de la plataforma [17], lo que en potencia apuntalará su relevancia futura.

Java 2 Mobile Edition (J2ME), de Sun Microsystems [18]: presente en la amplia mayoría de terminales, esta plataforma tenía además la importante ventaja de no ser desconocida para nosotros. Sin embargo la funcionalidad expuesta, así como su pobre integración con todas las plataformas en las que está presente nos hizo descartarla rápidamente.

Maemo 4, de Nokia [19]: la plataforma basada en Linux de Nokia para sus Internet Tablets ofrecía un potente entorno de desarrollo (*matchbox*) basado en tecnologías estándar Linux (*GTK*, *dbus*) [20]. Sin embargo su proyección de mercado era la más reducida: sin planes por parte de Nokia de extenderlo a teléfonos móviles, estaba muy limitada, y además no nos ofrecía capacidades de comunicación con el exterior y limitaba por tanto sus casos de uso.

Además, en paralelo a nuestra investigación o muy poco antes fueron anunciadas varias plataformas prometedoras, tanto por sus proyecciones de mercado como por sus capacidades de base:

iPhone, de Apple [21]: Las previsibles ventas de mercado de un teléfono-iPod hacían deseable esta plataforma, así como los primeros análisis sobre su facilidad de desarrollo una vez anunciado el SDK (que sin embargo no estaba disponible desde el lanzamiento), y el concepto de tienda de aplicaciones. Además, el anuncio del iPod Touch, compatible en software con la plataforma, hacía muy amplio el posible mercado.

Android, de Google [22]: El anuncio del primer sistema operativo de Google, en conjunto con el consorcio Open Handset Alliance [23] que conforman muchos fabricantes de telefonía de importancia como HTC, LG, Motorola, Sony-Ericsson o Siemens, así como pesos pesados del hardware como ARM, Atheros, Broadcom, Intel, Marvell, Nvidia, Qualcomm o Texas Instruments [24]. Esto indudablemente garantizaba su impacto y la atención de la industria hacia la plataforma. Sin embargo ha sido la última en llegar de todas, existiendo antes de su versión 1.0 bastantes dudas sobre las posibilidades de cumplir plazos por parte de Google, como finalmente lograron. En esta plataforma, si bien el uso de bytecode no permite a priori acceso nativo al sistema Linux sobre el que se basa, su proyección de crecimiento y validez como plataforma crecen, al poder ejecutarse las aplicaciones también sobre procesadores x86 [25], omnipresentes en ordenadores personales. De hecho existien varios fabricantes de portátiles como Asus o Acer que han presentado recientemente prototipos que usan Android como su sistema operativo ([26], [27]).

Qtopia / Qt extended, [28]: Originalmente de Trolltech, durante nuestro desarrollo la empresa fue adquirida por Nokia, lo que también probaba su relevancia como plataforma. Además, su soporte preliminar multiplataforma (Windows CE y Symbian) lo hacía muy deseable como plataforma base. Sin embargo, pese a su antiguedad no tenía teléfonos que la utilizasen directamente y por tanto estaba más destinada a embebidos de propósito específico, y limitada en su alcance. Recientemente no obstante se ha anunciado su abandono por parte de la compañía [29].

Openmoko, [30]: Un proyecto de una implementación libre de un *framework* o marco de desarrollo de telefonía sobre un hardware relativamente abierto –de especificaciones disponibles para la mayoría de componentes, con los esquemáticos a disposición del público, etc. Lamentablemente el teléfono se retrasó bastante respecto de su fecha prevista de lanzamiento y presentaba algunos defectos re-

lativamente severos de hardware en sus primeras versiones [31]. Ello, junto con la mala gestión de la comunidad por parte de la compañía, hizo que llegara al mercado con un software sin terminar y una excesiva diversidad de plataformas, lo que condicionaba sus posibilidades reales. Recientemente se confirmó además que la compañía abandona el desarrollo de futuros modelos [32] y reducía plantilla, confirmándose con ello el fracaso último del proyecto.

2.3. Análisis de la competencia

Durante la investigación acerca de la viabilidad encontramos varias aplicaciones que realizaban funciones similares a nuestros casos de uso.

Por un lado, sobre la plataforma Symbian existen multitud de pequeñas aplicaciones gratuitas y de pago que realizan subconjuntos de la misma, que principalmente se apoyan sobre los datos de localización que pueden obtenerse triangulando a partir de las torres de telefonía a las que está conectado el teléfono (pues no todos los móviles que soportan esta plataforma tienen GPS). Por ejemplo, *Aspicore GSM Tracker* [33], para deducir localización respecto de las torres GSM a las que está conectado, o incluso *Epocware Handy Profiles* [34], que permite configurar reglas de distinto tipo ("de la localidad", "de los intervalos temporales" y "del calendario") y cambiar los perfiles del móvil (silencio, manos libres, etc.) como respuesta.

Adicionalmente, nos encontramos con *Locale* [35]. Esta aplicación, finalista del primer concurso de desarrollo de aplicaciones de Google, y desarrollada en paralelo a la propia versión final del SDK de Android, permite también al usuario definir manualmente respuestas a condiciones preestablecidas como silenciar el teléfono; básicamente las mismas funcionalidades que las de nuestra aplicación, sin contar con la parte de la inteligencia artificial. Esta aplicación era por tanto la competencia más directa a la nuestra, pero podía acortar mucho el desarrollo de ser posible que partiésemos desde su código. Con este fin nos pusimos en contacto con los desarrolladores, para consultarles sobre el API que anunciaban disponible para extender su aplicación y añadir la funcionalidad que habíamos proyectado. Sin embargo, la aplicación sólo era accesible desde la tienda de aplicaciones (y por tanto desde un terminal físico, del que no disponemos). Además nos confirmaron que dicho API estaba planificado pero aún no disponible, y que no planeaban liberar el código, siendo el mismo la única forma de ampliar funcionalidad.

A pesar de todo, consideramos interesante la continuación del proyecto con el fin de alcanzar los objetivos que quedan fuera del resto de alternativas. La imposibilidad de partir desde Locale nos confirmó la necesidad de reimplementar parte de esa funcionalidad por nuestra cuenta para dar soporte a nuestros objetivos. Sin embargo el tiempo invertido en ello nos ha limitado nuestra capacidad de alcanzar dichos objetivos.

Capítulo 3

Análisis

3.1. Casos de uso básicos

Concretada la idea general de qué queríamos que hiciera nuestra aplicación, el número de casos de uso posibles es relativamente ilimitado, ya que cada vez son más las acciones que un usuario puede realizar con un teléfono móvil. Aquí exponemos los principales casos de uso que hemos contemplado:

- Entrar en la biblioteca de la Facultad y el teléfono cambia automáticamente a modo silencio.
- Al llegar a cierta localización el teléfono envía un mensaje de texto avisando a un contacto de que ya he llegado; por ejemplo, después de un viaje.
- A cierta hora el teléfono me avisa de que debo llamar a un contacto.
- A partir de cierta hora de la noche, y hasta la mañana siguiente, el teléfono se pone en modo silencio.
- Al detectar que me he conectado a Internet, el teléfono descarga automáticamente el correo electrónico.
- Al detectar que me desconecto del *bluetooth* del coche, el teléfono guarda automáticamente su localización por GPS para recordar dónde lo he aparcado.
- Al recibir una llamada de teléfono en cierta circunstancia (hora, lugar, etc), el teléfono cuelga o silencia automáticamente la llamada. Este caso de uso puede ser ampliado para tener en cuenta grupos de contactos o ciertas excepciones:
    - Silenciar o colgar sólo si el número entrante es desconocido.

  - Silenciar o colgar sólo si el número entrante es cierto contacto o pertenece a cierto grupo.
  - No silenciar o colgar si el número de teléfono es cierto contacto o pertenece a cierto grupo.

Además contamos con otros casos de uso que requieren una interacción más compleja por parte del programa, con ciertos aspectos de Inteligencia Artificial que detallaremos más adelante.

- Detección de patrones de comportamiento: el teléfono es capaz de detectar un hábito del usuario que describe alguno de los casos de uso explicados anteriormente. En ese momento el teléfono pregunta al usuario si desea que haga automático ese patrón de comportamiento (se detallará más adelante como sugerencia de reglas en la sección 4).
- El teléfono detecta que tengo una cita en cierta localización a una hora determinada. Por aprendizaje previo, el teléfono es capaz de saber que es probable que dada la hora actual y mi localización, no llegue a la cita; entonces me avisa de que llego tarde y propone enviar un mensaje de texto al contacto con el que tengo la cita, informando de que me retrasaré.
- Al detectar un contacto de *bluetooth* en un radio cercano, con el cuál tengo una cita en la agenda, el teléfono me avisa de que este contacto está cerca y me muestra la información pertinente.
- El teléfono detecta que tengo una reunión con ciertos contactos en cierta localización. Al llegar a ese lugar el teléfono envía automáticamente la documentación de la reunión al resto de asistentes.

3.2. Análisis de las plataformas

Al margen de las consideraciones de mercado, nuestro análisis técnico de las plataformas fue el siguiente:

- Windows Mobile (familia Windows CE)
  - *Lenguaje:* C / C++ / .NET / Flash lite
  - *Pros:* Veteranía de la plataforma (documentación, comunidad de desarrolladores, etc).
  - *Contras:* Licencias propietarias. Coste del entorno de desarrollo. La diversidad de dispositivos, si bien facilita el impacto, dificulta el desarrollo por la diversidad del hardware y de las características soportadas por cada uno.

  - *Licencia:* propietaria.

- Blackberry
  - *Lenguaje:* Java contra API propietario.
  - *Pros:* Penetración de mercado de la plataforma.
  - *Contras:* Segmento de mercado de la plataforma poco interesado en potencia en el proyecto. Apariencia mejorable de las aplicaciones [36]. Problemas de usabilidad. Ausencia de hardware sobre el que probar. Entorno de desarrollo con ciertas carencias.
  - *Licencia:* propietaria.

- Symbian
  - *Lenguaje:* C / C++, Python.
  - *Pros:* Plataforma veterana. Alta penetración de mercado.
  - *Contras:* Dificultad de inicio en el desarrollo para la plataforma. Coste del desarrollo: aunque existen versiones gratuitas de los entornos de desarrollo, la segmentación podía implicar que las ediciones gratuitas (pero no libres) de los SDK e IDEs tuviesen limitaciones fundamentales.
  - *Licencia:* propietaria, anunciada su liberación futura.

- J2ME
  - *Lenguaje:* Java contra API específico.
  - *Pros:* Máxima penetración de mercado (soporte en prácticamente cualquier teléfono). Sencillez de desarrollo. Herramientas de desarrollo maduras (*plugins* sobre Eclipse y emuladores con soporte para *profiling* o análisis de rendimiento).
  - *Contras:* Mejorable integración con cada plataforma. Obligada la depuración de comportamiento y funcionalidad sobre la plataforma destino. Poca funcionalidad necesaria preimplementada, y soporte de APIs de GPS opcional sobre cada teléfono concreto.
  - *Licencia:* propietaria.

- Maemo
  - *Lenguaje:* múltiples (C, C++, Python, Lua, Java...).
  - *Pros:* Similaridad de la plataforma a las capas de software estándar Linux (está basado en una distribución *Debian* y un *toolkit GTK* modificado llamado *Hildon*). Software libre. Facilidad para la portabilidad entre aplicaciones de escritorio normales y específicas de Maemo. Entorno de desarrollo potente, con emulador incorporado. Comunidad activa de desarrollo.

- *Contras:* APIs de demasiado bajo nivel en algunos casos. Falta de funcionalidad por implementar (funcionalidades avanzadas de ubicación, capa de abstracción de la comunicación *Empathy* aún en desarrollo). Carencias del hardware: en esencia, no tener telefonía no permitía obtener posicionamientos aproximados con poco consumo de energía.
- *Licencia:* GPLv2.

- iPhone
  - *Lenguaje:* Objective-C contra Cocoa.
  - *Pros:* Rápido crecimiento de mercado. Facilidad de desarrollo, con un potente entorno para ello (*Xcode*) [37]
  - *Contras:* Sin soporte oficial de aplicaciones corriendo en segundo plano, algo necesario para nuestra aplicación. Coste de desarrollo muy elevado (registro obligatorio como desarrollador + desarrollo obligatoriamente sobre *OSX*, solo soportado sobre portátiles Apple).
  - *Licencia:* Propietaria.

- Android
  - *Lenguaje:* Basado en Java, contra API propio.
  - *Pros:* Facilidad de desarrollo. Potencia del SDK (plugin sobre Eclipse integrado con un emulador del hardware que admite conexiones para depuración, compatible plenamente con Linux y soportando la misma funcionalidad sobre terminales físicos). API muy extenso y de alto nivel, con gran parte de la funcionalidad necesaria preimplementada. Disponibilidad de herramientas (NDK, Herramientas de Desarrollo Nativo [38]) para compilar aplicaciones en C ó C++ contra cabeceras nativas del sistema y enlazarlas con código en Java mediante *Java Native Interface*, para secciones del código más intensivas en rendimiento. Posibilidad de sustituir componentes del propio sistema operativo en código (lo que nos facilitaría extensiones al mismo, en caso de ser necesarias, sin tener que bajar demasiado de nivel de abstracción). Proyección de mercado de Google. Multiplataforma por estar basado en Java: soporta por ejemplo *ARMv5TE* [39] y *x86* sobre el mismo *bytecode*, lo que permite que el software pueda ejecutarse sobre múltiples dispositivos distintos, ampliando el posible mercado objetivo de los programas desarrollados.
  - *Contras:* Especificidad de la mayor parte del código - ya que no sigue estándares Linux como *X*, solo utiliza el kernel. Modelo de aplicación distinto que obliga a diseñar específicamente para la plataforma, no pudiendo

hacerse el código portable. Plataforma cerrada: si bien puede modificarse el sistema operativo por ser software libre, la mayoría de teléfonos solo cargan *firmware* firmado y por tanto no permiten cargar código modificado por el usuario, aunque éste haya trabajado sobre las fuentes del sistema [40]. API muy cambiante en muy poco tiempo por su inmadurez. Dispositivos relativamente lentos, algo exarcerbado por la carencia de compilación *Just-In-Time* de la máquina virtual.

- *Licencia:* Originalmente cerrada cuando tomamos la decisión, pero estaba anunciada - y finalmente se produjo - la liberación de todo el código bajo licencia Apache 2.0, estableciéndose además protocolos de cooperación con terceros desarrolladores para la publicación de sus parches.

- Qtopia / Qt Extended

  - *Lenguaje:* C++
  - *Pros:* Madurez del entorno. Su similaridad al Qt estándar (de aplicaciones de escritorio) permite que sean muy aplicables a posteriori los conocimientos adquiridos y fácilmente portables. Ports en desarrollo para Windows CE y Symbian: un mero recompilado del software haría que se ejecutase bajo estos sistemas operativos sin demasiados cambios.
  - *Contras:* Mayor tiempo de desarrollo de C++ respecto de Java. Plataforma no diseñada para accesos a bajo nivel, aunque los soporta. Ausencia de *X*. Limitaciones de alcance de la plataforma. IDEs menos elaborados inicialmente.
  - *Licencia:* inicialmente dual (GPL para proyectos GPL), posteriormente GPL.

- Openmoko

  - *Lenguaje:* múltiples (C, C++, Python, Lua, Java...)
  - *Pros:* Similaridad de la plataforma a la pila de software estándar Linux. Apertura total de la plataforma. Comunidad activa de desarrollo. Posibilidad de impacto real (si el software es bueno y no hay alternativas anteriores es posible que formase parte del proyecto).
  - *Contras:* Inmadurez de la plataforma. Ausencia de IDEs y documentación alguna. Volatilidad del proyecto y dudas sobre su viabilidad. Ausencia de gran parte de la funcionalidad esencial de telefonía. Problemas en la gestión del proyecto. Limitacion de la proyección de mercado. Coste de adquirir el dispositivo y problemas encontrados para usar el emulador que existe para el mismo.
  - *Licencia:* varias (principalmente GPLv2).

Inicialmente nuestras plataformas candidatas excluían de partida Blackberry y Windows Mobile por exigir un desarrollo en Windows. Aunque eso descartaba en principio también a Symbian, el disponer de varios dispositivos basados en diversas versiones de su sistema operativo nos permitía hacer pruebas sobre dispositivo real, algo a valorar, y además su proyección de mercado y la posibilidad de programar en Python hizo que fuese de las últimas en ser descartada, principalmente por la dificultad de iniciarse en el desarrollo que nos encontramos.

Otra plataforma que no descartamos hasta el final fue Openmoko, en particular el dispositivo físico (GTA02, [41]), ya que la apertura de la propia plataforma hardware, si bien no carente de problemas, permitía la máxima flexibilidad posible a la hora de elegir una plataforma [42] –cuando tomamos la decisión estaba en proceso un *port* de Android, ahora ya casi completamente operativo, y también soportaba su propia distribución basada en *Openembedded* [43] con varias capas de software alternativas sobre ella, así como *Debian* sobre ARM [44], con su amplio espectro de paquetes disponibles. Otras ventajas eran el cargador de arranque o *bootloader* abierto, que permitían cargar cualquier sistema operativo sin restricción alguna, la existencia de una placa adicional de depuración de hardware *JTAG* [45] -lo que permite depurar incluso el kernel de ser necesario-, y el y soporte hardware de todo lo que pudiéramos necesitar: acelerómetros, GPS, wifi... Finalmente no llegamos a adquirir el dispositivo por su coste, y por la importante inmadurez del kernel y del propio hardware, con problemas en el GPS [46], en el consumo de energía del aparato y en la carga de la batería [47], problemas entre el kernel y el hardware para suspender el teléfono [48], carencias del chip gráfico *Glamo* integrado [49], problemas de eco en las llamadas [50], etc. La mayoría de estos problemas han sido solventados ya o están siendo investigados, pero en su momento descartaron definitivamente al hardware.

Las distribuciones alternativas a Android en Openmoko fueron deshechadas también, principalmente por la dificultad de poner un entorno de desarrollo en marcha, por los retrasos del proyecto en conseguir resultados y por las carencias del software existente, lo cual, si bien nos permitía el impacto de nuestro desarrollo en el software debido a ser libre, nos alejaba mucho de los objetivos deseados.

La otra posibilidad que consideramos hasta el último momento fue Qt, ya que la adquisición de la compañía por Nokia nos hacía dudar de que se fuera a abandonar, así como su portabilidad a distintos operativos como Windows CE y Symbian era una ventaja indudable. Sin embargo los idiomas específicos de C++ que componen Qt, unido a la mayor dificultad de desarrollar en C++ –debido a la gestión específica de la memoria– nos suponían un inconveniente (en particular, Qt cuenta con implementaciones propias de la mayoría de funcionalidad de la Librería Estándar

de Tipos *STL* [51] con la que éramos familiares los desarrolladores, y el diseño de señales y zócalos [52] también es específico). Además, la proyección de mercado lo limitaba a ejecutar sobre la plataforma hardware de Openmoko, con sus problemas antes citados.

Por todo ello finalmente nos decantamos por Android por considerar que nos permitiría un desarrollo más rápido, y contaba con varios puntos a favor con respecto al resto de alternativas. Principalmente, su API nos daba más funcionalidad preimplementada, sobre todo en lo referente a la información de posicionamiento mediante el *LocationManager*, además de otros servicios que nos permitiría usar con facilidad el organizador personal, la agenda de contactos, etc. Consideramos pues que Android nos permitiría alcanzar más objetivos de los previstos inicialmente. A posteriori, consideramos muy acertada nuestra decisión por las brillantes proyecciones de crecimiento que posee esta plataforma y el impacto que está causando en la industria.

Capítulo 4

Diseño

4.1. Arquitectura preliminar

Con los objetivos generales (1.1) en mente nos dispusimos a estructurar la aplicación y a planificar el desarrollo. Sin haber aún decidido la plataforma sobre la que desarrollaríamos (y por tanto sin poder asumir qué funcionalidad proporcionaría más que a grandes rasgos), esbozamos una estructura en grandes bloques para analizar la información necesaria en cada parte de la aplicación, y las funcionalidades principales. Con ello llegamos al siguiente resultado:

Recogida de información (*bloque 1*)
: Este módulo estaría encargado de recoger toda la información necesaria del teléfono móvil, desde los cambios del entorno (hora, lugar, llamadas recibidas, conectividad, etc), a las acciones que realiza el usuario (colgar una llamada, poner el móvil en silencio, etc). Ésta información se dividiría en tres partes, una para cada una de los bloques fundamentales de la aplicación, como explicaremos más adelante. La forma en que se obtiene toda la información es dependiente de la plataforma elegida, limitado por los servicios disponibles en el API en el caso de Android, y de la complejidad de implementarlo o de cuánto acceso a bajo nivel permitieran otras plataformas.

Gestión de reglas (*bloque 2*)
: Este módulo implementaría la gestión de las reglas predefinidas por el usuario. Para ello, necesita recibir la información de los eventos que se producen (que parten del contexto, como por ejemplo "entrar en una localización"), y efectúa la reacción correspondiente. Para completar esta funcionalidad se hace necesario incluir un interfaz gráfico de usuario o GUI para permitir al usuario definir estas reglas, así como de un sistema de almacenamiento (base de datos o similar) para poder mantener la información cuando el teléfono móvil se apaga.

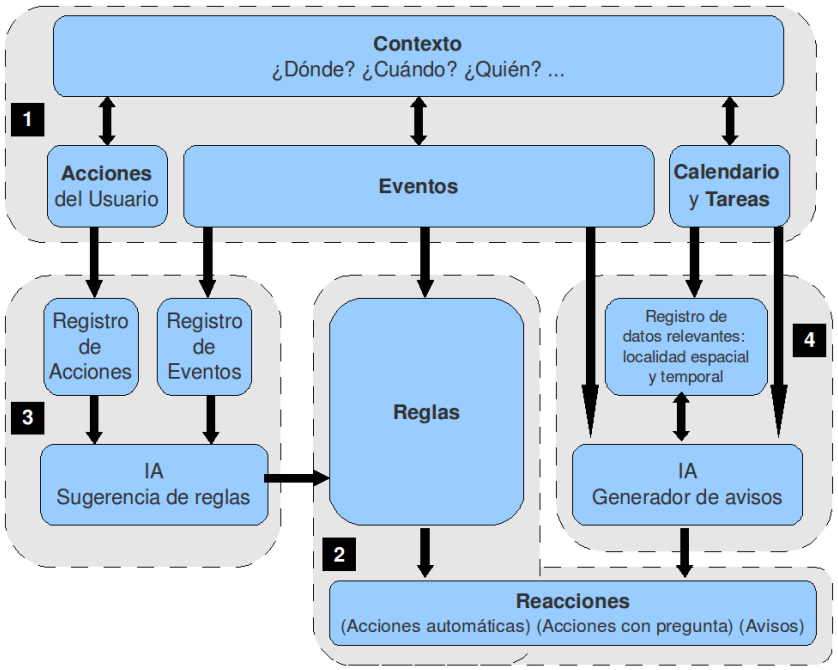

FIGURA 4.1: Diagrama de la arquitectura de la aplicación

Sugerencia de reglas (*bloque 3*)

Para sugerir nuevas reglas al usuario, registraría esos mismos eventos y las acciones del usuario y las almacenaría. De esta forma, y mediante técnicas de Inteligencia Artificial, la aplicación podría deducir qué acciones del usuario se repiten en el mismo contexto las suficientes veces para inferir una nueva regla. Ésta regla se mostraría por pantalla al usuario, que podría confirmarla o rechazarla. En el caso de que la aceptase, pasaría al registro de reglas de la aplicación como si fuera otra de las definidas por el usuario, y comenzaría a funcionar como éstas a partir de ese momento. Este bloque es por tanto funcionalidad añadida al sistema de gestión de reglas definido en el punto anterior.

Generador de avisos, agenda inteligente (*bloque 4*)

Por último definimos otro bloque de funcionalidad que se encargaría de lanzar avisos al usuario en función de su comportamiento habitual, que el móvil ha ido registrando a lo largo del tiempo. Se encargaría de analizar las costumbres habituales del usuario, tanto las definidas directamente por él en su agenda (Calendario, Agenda, horarios de trabajo/clase, Notas, etc) como la información que el propio teléfono debería deducir y almacenar (desplazamientos habituales del usuario y el tiempo que tarda en realizarlos, hora a la que comienza a utilizar el teléfono, etc). De esta forma el teléfono podría asumir una funcionalidad de

agenda inteligente más allá de lo que el propio usuario especificase, y avisarle cuando lo considerase necesario. Por ejemplo, si tuviera una cita registrada en la agenda con cierto contacto en una localización a la que tarda habitualmente cierto tiempo en llegar desde la actual, y dada la hora fuese improbable que llegase a tiempo, se podría avisar al usuario y proponer enviar un mensaje a dicho contacto para avisarle del retraso. Esta funcionalidad, al contrario que las anteriores, es independiente de la gestión de reglas, por lo que se puede tratar aparte.

4.2. Arquitectura final

Una vez esbozado el esquema general de la aplicación y elegida la plataforma, en este caso Google Android, analizamos la mejor forma para comenzar a trabajar en el desarrollo del proyecto. Estudiadas las facilidades y limitaciones del API para la recogida de la información de contexto, y vistas las propias dependencias de nuestro diseño, decidimos comenzar con un desarrollo por fases que sigue el orden lógico en el que se han analizado los bloques, centrándonos principalmente en las dos funcionalidades básicas de gestión y sugerencia de reglas, y dando menos importancia al generador de avisos.

El proceso de desarrollo del proyecto quedaría de la siguiente forma:

1. Inicialmente, desarrollo de la parte básica de la aplicación, con la recogida y tratamiento de la información, así como la creación, edición y eliminación de reglas mediante GUI, y su almacenamiento.

2. Desarrollo del módulo de recogida de la información de las acciones del usuario y su análisis mediante Inteligencia Artificial (IA) para sugerir reglas al usuario.

3. Finalmente, y considerándolo como un bloque con una funcionalidad independiente al resto del proyecto, y con un tratamiento lógico y de IA más complejo, el generador de avisos.

4.2.1. Conceptos y funcionamiento básicos

En primer lugar, definimos un glosario de términos importantes que utilizamos en toda la aplicación y el funcionamiento que tendrán tanto de cara al usuario como a nivel de programación. El concepto base de la aplicación es la Regla. Aquí definimos sus elementos y su funcionamiento básico:

Regla: es el conjunto de condiciones de entorno (o ambiente en el que se encuentra el teléfono) que el usuario establece como relevantes en cierta situación, y las reacciones o cambios de estado que el teléfono realizará cuando esas condiciones

se cumplan. Éstas pueden ser un número ilimitado de *"Contextos"*, uno o ningún *"Evento disparador"*, y un conjunto también ilimitado de *"Reacciones"* que el teléfono debe llevar a cabo.

Evento: un evento es una situación que sólo tiene lugar en un momento determinado, puntual e instantáneo. De cara a la lógica de las reglas puede tener dos funciones: la primera, como parte de un *Contexto* (como veremos más adelante), ya sea como inicio o como fin de éste; y la segunda, como un *"evento disparador"* que activa la regla. Los diferentes tipos de eventos implementados en la aplicación se verán más adelante (sección 5.3.2.1).

Contexto: un contexto consiste en una situación que tiene lugar entre un *"Evento de activación"* y un *"Evento de desactivación"*, y que se considera *"activo"* en el tiempo que transcurre entre ambos.

Reacción: cada una de las acciones o cambios de estado que realiza el teléfono en respuesta a una regla que ha quedado activada. Dependiendo del tipo de reacción es posible que cuente con una *"contrarreacción"* o reacción opuesta, que se lanza cuando la regla correspondiente queda desactivada, notificándolo al usuario o devolviendo el teléfono al estado anterior a la activación de la regla. También los distintos tipos de reacción se tratan más adelante (sección 5.3.2.2).

Es importante notar la diferencia entre contextos (sucesos que se alargan en el tiempo) y eventos (puntuales). Algunos ejemplos de contextos podrían ser *"mientras estoy en la biblioteca"* o *"desde las 9:00 hasta las 17:00"*, mientras que eventos serían *"cuando entro en la biblioteca"*, *"a las 22:00"* o *"cuando me llama algún compañero de trabajo"*.

Esta definición de regla hace que contemos con dos tipos implícitos de las mismas: las *reglas de contexto*, que son aquellas que no cuentan con *"evento disparador"*, y las *reglas de evento*, que son aquellas que sí lo tienen.

- Las primeras, las **reglas de contexto**, quedarán activadas (y por tanto lanzarán las reacciones correspondientes) a partir del momento en que todos sus contextos estén activos, y dejará de estarlo –lanzándose si procede las contrarreacciones– cuando alguno de ellos se desactive. Por ejemplo, *"mientras estoy en el centro de trabajo, el teléfono permanecerá en silencio todo el tiempo, y al abandonarlo, volverá al modo de teléfono anterior"*.
- Por el contrario, las **reglas de evento** no se activarán directamente cuando todos los contextos se activen, sino cuando esto ocurra *y además* tenga lugar el *evento disparador*. En ese momento se lanzan las reacciones, que serán

de carácter instantáneo y puntual; y, por sentido común, en este tipo de reglas no tienen cabida contrarreacciones, ya que no se produce un cambio de estado de larga duración. Un ejemplo de regla de evento puede ser *"desde las 00:00 hasta las 07:00, silencia cualquier llamada entrante"*. El evento disparador sería *"llamada entrante"*.

Para el bloque de sugerencia de reglas añadimos un nuevo concepto:

Acción: es todo aquello que el usuario puede realizar con el teléfono móvil y que puede quedar registrado como posible reacción de una futura regla inferida; por ejemplo, cambiar el modo del teléfono, realizar una llamada, etc.

Estos conceptos y funcionamiento son independientes de la plataforma elegida, aunque pueden variar para adecuarlos mejor a ella en el momento de la implementación, como finalmente ocurrió. En la sección 5.3 se detalla la implementación de la arquitectura final.

4.3. Inteligencia Artificial

En un principio pospusimos la investigación sobre inteligencia artificial por ser una parte separada de nuestro proyecto, que dependía de una infraestructura de recoger contexto y actuar conforme a él que tendríamos que implementar previamente. Nos centramos en la faceta de sugerir reglas aplicables al usuario, ignorando, por estar fuera de ámbito, los aspectos del código relativos al tratamiento de la agenda. Aparecieron entonces varias posibilidades: algoritmos de inferencia de reglas, o algoritmos bayesianos.

4.3.1. Inferencia de reglas

Primeramente consideramos, por ser la más sencilla –tanto de implementación como computacionalmente– basarnos en ajuste de patrones predefinidos: por ejemplo, preguntar por añadir una localización siempre que se detectase que se ha realizado algún evento cierto número de veces en un radio predeterminado respecto de cierta localización.

Como extensión de esta idea investigamos también la generalización del problema, a partir de publicaciones como [53] [54] [55] [56], si bien no tuvimos tiempo de implementar ninguna de las ideas.

Resumiendo esta línea de investigación: en general, nuestro problema se define como uno de inferencia de reglas de asociación , dentro del campo del *Data Mining* o

minería de datos [57]. Estas, en resumen, son una expresión $X \Rightarrow Y$, donde X e Y son conjuntos de items cualesquiera, habitualmente llamados pre y postcondiciones de la regla, de un conjunto mayor D, y el significado de las mismas es que, dada una base de datos de transacciones –donde cada una es también un subconjunto de D–, cuando una transacción contiene X también contiene Y con cierta probabilidad. La *confianza* de la regla es su probabilidad condicionada: $P(Y \subseteq T|X \subseteq T)$, para $T \subseteq D$, y nos da información sobre la relevancia de la regla. Otro modo de definir esta medida es mediante el *soporte* de un subconjunto I de D, que es igual a

$$\frac{\| \{t \in D|I \subseteq t\} \|}{\| \{t \in D\} \|}$$

. Esta fórmula equivale al porcentaje de transacciones de D que contienen I. Asimismo se define el soporte de una regla como el soporte $(X \cup Y)$, que nos da información sobre cuántas transacciones apoyan la existencia de una regla. Por tanto, la confianza de una regla es también igual al soporte de la misma entre el soporte de sus precondiciones.

Esta serie de problemas se suelen también llamar *problemas de la cesta de la compra*, ya que su aplicación más directa es tratando los conjuntos de elementos como productos de un supermercado, siendo las transacciones compras individuales. Con ello se intenta extraer información relevante y no trivial sobre los patrones de compra de los usuarios (por ejemplo, sobre productos que se compran juntos habitualmente [58]). De la información existente, se puede extraer o bien todas las reglas posibles, o bien aquellas que cumplan unos requisitos mínimos de confianza y soporte.

Aplicándolo a nuestro caso concreto, la base de datos estaría formada por acciones del usuario y contextos, y una transacción consistiría en una acción del usuario, unida a su contexto relevante (o al contexto que fuera factible recoger sin excesivo consumo de batería, ya que los propios algoritmos pueden determinar la parte del mismo que es relevante); por tanto, los items posibles son la unión de todas las acciones que registramos, con todos los contextos y también las reacciones. Entonces, la acción del usuario desencadenaría el procesamiento, y el resto de items que formasen parte del conjunto de la transacción serían contextos. Además, el lado derecho de la regla consistiría en reacciones genéricas, asociadas siempre con la acción (esto es: si en cierto sitio siempre se pone el teléfono en silencio, la reacción sugerida es siempre poner el móvil en silencio, y es al sugerir la regla al usuario cuando éste podría añadir más notificaciones de tipos que no se pueden inferir de la información que tenemos, como un aviso).

Existen varios algoritmos para resolver este tipo de problemas: *Apriori* [59], *Close* [54], *Eclat* [60], *Max-Miner* [61] y *FP-growth* [62], si bien estos algoritmos [57] se encargan solo de hallar los conjuntos de ítems frecuentes (con un soporte al menos igual a una constante predefinida), prerrequisito para hallar después todas las reglas sobre ellos con una confianza también mayor o igual que otra constante. Este problema de hallar subconjuntos relevantes de items tiene una complejidad a priori igual a $2^{|D|}$ (ya que son los posibles conjuntos de items que existen). Existen también otros algoritmos que resuelven el problema completo, como *One-Attribute-Rule* (que halla reglas con una única precondición).

Una aproximación alternativa, presente en [55], trata el problema bajo el campo del análisis formal: la tarea de encontrar conjuntos frecuentes se describe como, dado un conjunto G de objetos, otro M de atributos, una relación binaria $I \subseteq G \times M$ (donde $(g, m)\exists I$ significa que "un objeto g tiene un atributo m"), buscar todos los subconjuntos de M (llamados *patrones*) que tengan un soporte mayor que el mínimo. Aquí soporte se define como la cardinalidad de X' entre la de G, para $X' = \{g \exists G | \forall m \in X : (g, m)\exists I\}$. En general, tratar el problema como uno de análisis formal de conceptos permite reducir el número de reglas inferidas sin pérdida de información, algo que lo haría particularmente útil para nosotros.

4.3.2. Algoritmos bayesianos

Otra línea de desarrollo sugerida por Marco Antonio, uno de los directores, fue usar *algoritmos bayesianos*, especializados en clasificación (su aplicación de uso más típica es decidir si un correo electrónico es no deseado). En nuestro caso, los estados serían equiparables a reacciones que deseamos del teléfono (como por ejemplo estar en silencio), y la información de entrada que tomaría es el distinto contexto que se pueda recopilar con coste razonable en el momento de la acción del usuario.

Otra alternativa sería basarlo en un subtipo de algoritmos bayesianos llamados *redes de decisión* [63], donde aquí los distintos nodos de decisión sí serían directamente las reacciones posibles (que tienen una función de utilidad esperada asociados), y de funcionamiento parecido a los anteriores: se evalúa esa función de utilidad esperada para los posibles nodos de decisión, aplicándoles la entrada del contexto presente, y se opta por el que tenga mayor valor.

Con este modo de funcionamiento basado en algoritmos bayesianos el móvil asignaría probabilidades a los diversos estados y cambiaría entre ellos directamente cuando el algoritmo decidiese, sobre el contexto. Sin embargo, finalmente lo descartamos por su dificil aplicación a nuestra estructura de reglas predefinidas, que consideramos en principio más usable por ser más previsible, y ha quedado como una posible expansión o rediseño del proyecto, en el que se activa un modo de funcionamiento autónomo del teléfono tal como se menciona en la sección 6.2.

Capítulo 5

Implementación

5.1. Herramientas utilizadas

En esta sección detallamos las herramientas usadas durante el desarrollo del proyecto, así como algunas de las ventajas e inconvenientes que hemos encontrado en ellas.

5.1.1. Entorno de desarrollo - Eclipse

Para nuestro desarrollo hemos empleado principalmente las herramientas que proporciona el SDK de Android, en particular su versión 1.5r1, durante la mayor parte del desarrollo. Utilizamos el *plugin* compatible con el propio SDK del entorno integrado de desarrollo Eclipse [65], versión 3.4.2 sobre Linux: *Debian Testing* [66] de 64 bits, *Ubuntu Hardy Heron* [67] de 32 bits y *Ubuntu Karmic Koala* de 64 bits, respectivamente. Las pruebas las realizamos sobre el emulador incluido en el SDK.

El código es compatible con la versión 1.1 del API , contra la que se puede también desarrollar con el SDK de la versión 1.5 (que en particular permite desplegar varias versiones de *firmware* en el emulador, incluso y *firmware* compilados por el propio usuario desde las fuentes de Android). Finalmente nos vimos obligados a utilizar la versión 1.1 al encontrar un bug crítico en la versión 1.5 del *firmware* del emulador [68] que provocaba que no se generasen avisos de cambio de posición después del primero lanzado, un problema notable dadas las características de nuestra aplicación.

La utilización de estas herramientas de desarrollo fue muy positivo por las facilidades que nos ofrecían, especialmente a la hora de realizar pruebas, ya que el emulador permite simular llamadas entrantes y salientes, cambios de localización,

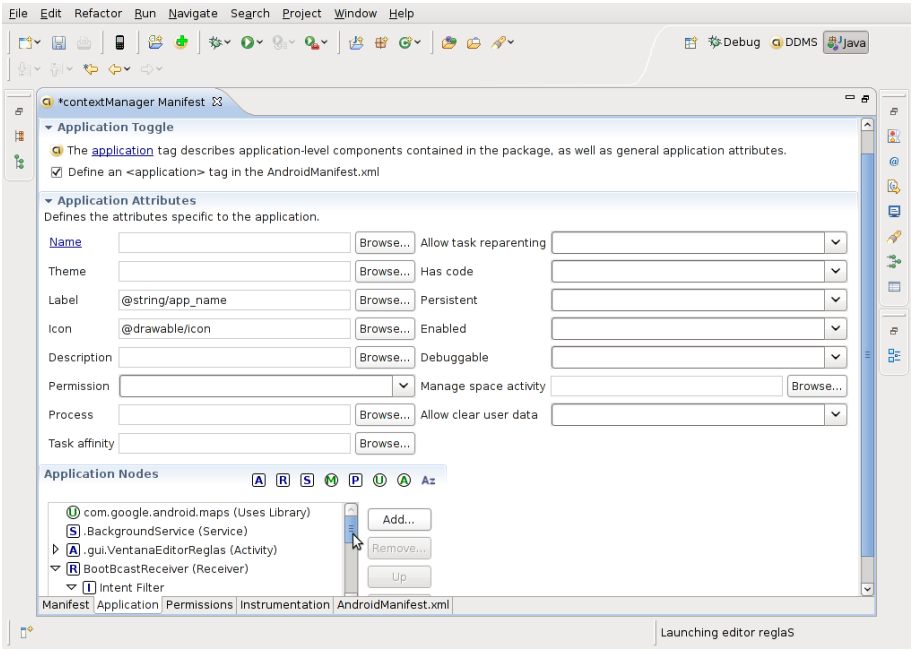

FIGURA 5.1: Entorno de desarrollo Eclipse – Manifest.xml

recepción de mensajes cortos y multimedia, etc. En definitiva, nos permitía probar casi todos los casos de uso que finalmente fueron implementados de un modo muy directo. Además, el uso de Eclipse como entorno de programación, en parte gracias a la experiencia previa de los miembros del grupo, facilitó también las labores de desarrollo.

Aun así, encontramos diferentes problemas, sobre todo debido que tratar con una plataforma nueva que está actualmente en constante evolución. A continuación detallamos algunos de estos problemas:

En primer lugar, el uso de sistemas operativos de 64 bits dio algunos problemas adicionales, ya que algunas herramientas incluidas en el SDK (como por ejemplo *ddms*, el servicio de monitor de depuración de la máquina virtual *Dalvik*) están enlazadas contra bibliotecas incluidas (parte del *SWT - Standard Widget Toolkit*, un API de gráficos de Java sobre el que se basa el propio Eclipse), y compiladas para 32 bits. Pudo solventarse mediante las herramientas incluidas directamente en el *plugin* de Eclipse, así como el uso del paquete *ia32-java-sun-bin* ([69] y [70]).

Por su parte también fue problemática la introducción de coordenadas de GPS al sistema. Desde Eclipse hay otro *bug* cuando el idioma del sistema no es inglés que

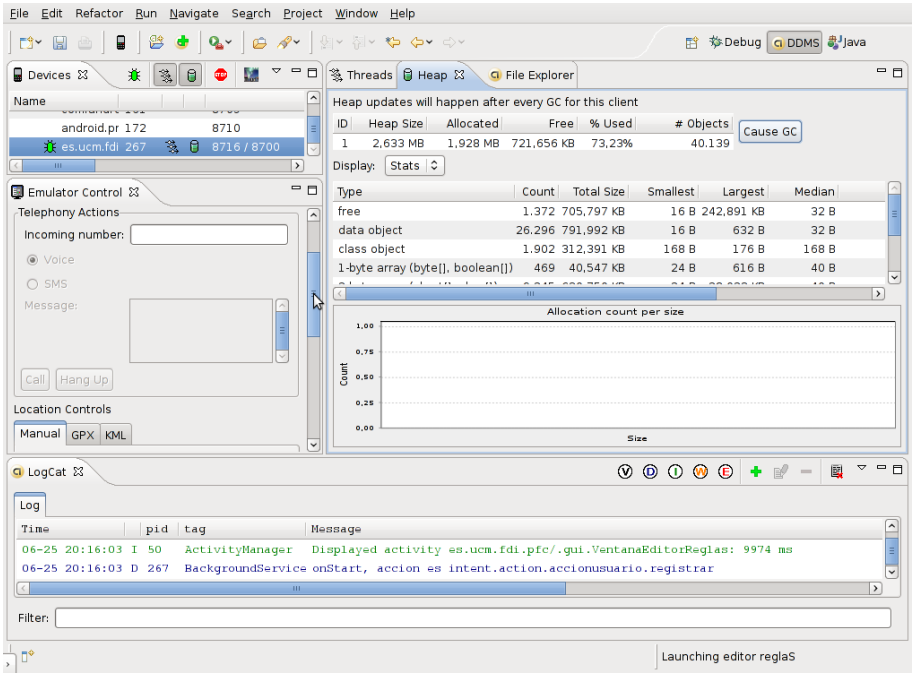

FIGURA 5.2: Entorno de desarrollo Eclipse – Plugin DDMS

evita que funcione correctamente [71], solventado fácilmente invocando el programa con la variable *LC_ALL* exportada al valor *"C"*. La alternativa, conectar mediante el programa de terminal *telnet* al emulador [72] e introducir las coordenadas allí, resultó en otro problema adicional de falta de precisión en las coordenadas que recibía el emulador respecto de las introducidas. Esto redundaba en que las coordenadas introducidas no se recibieran correctamente (como sí sucede desde Eclipse), y por tanto que las localizaciones fallasen, ya que en coordenadas geográficas una pequeña diferencia del tercer decimal se puede traducir en kilómetros de distancia.

Otros problemas encontrados durante el desarrollo han sido no poder conectar el depurador a dos procesos simultáneos del mismo paquete, así como la limitación en las configuración de ejecución, que solo permiten arrancar Activities. En particular, cuando se lanza una Activity el *plugin* reinstala en el dispositivo el paquete y para ello mata sus procesos previamente, lo que impide lanzar simultáneamente y en distintos procesos dos, algo sí permitido por Android [73] y que barajamos -y descartamos por este motivo- utilizar para separar el servicio persistente del editor de reglas.

5.1.2. Control de versiones - Bazaar

En cuanto al control de versiones, se utilizó la herramienta de línea de comandos Bazaar [74], que implementa control de versiones distribuido, junto con Olive [75] para la revisión visual y más usable de cambios y Meld [76], integrado con Olive, para la resolución de conflictos gráficamente. También se usaron puntualmente el *plugin* de Eclipse para soportar Bazaar [77], así como el plugin de salida XML de Bazaar [78], requerido por éste.

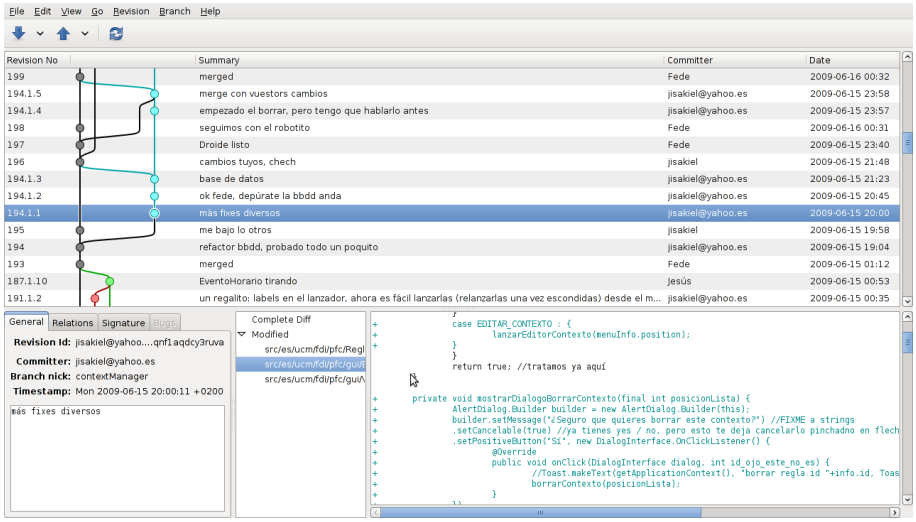

FIGURA 5.3: Olive

La infraestructura para centralizar el desarrollo fue un servidor dedicado personal, sobre una máquina virtual con *Gentoo Linux Hardened* [80], a la que se accedía mediante *ssh* [81] y que albergaba el repositorio central de Bazaar así como otras ramas puntuales del proyecto. También se usó al principio del desarrollo un *hosting* de otro de los desarolladores, sobre el que se desplegó un *wiki* para la edición colaborativa y centralizar la información basado en *MediaWiki* [82].

5.1.3. Memoria del proyecto

La memoria del proyecto ha sido desarrollada por su parte sobre Google Docs [83] para facilitar la edición colaborativa, y posteriormente mediante LaTeX [84]. Los dia-

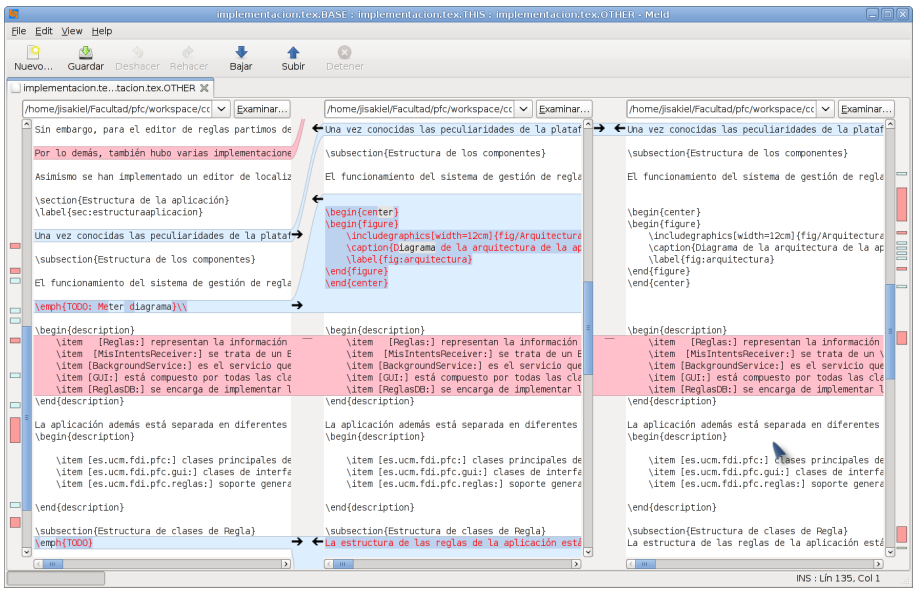

FIGURA 5.4: Meld

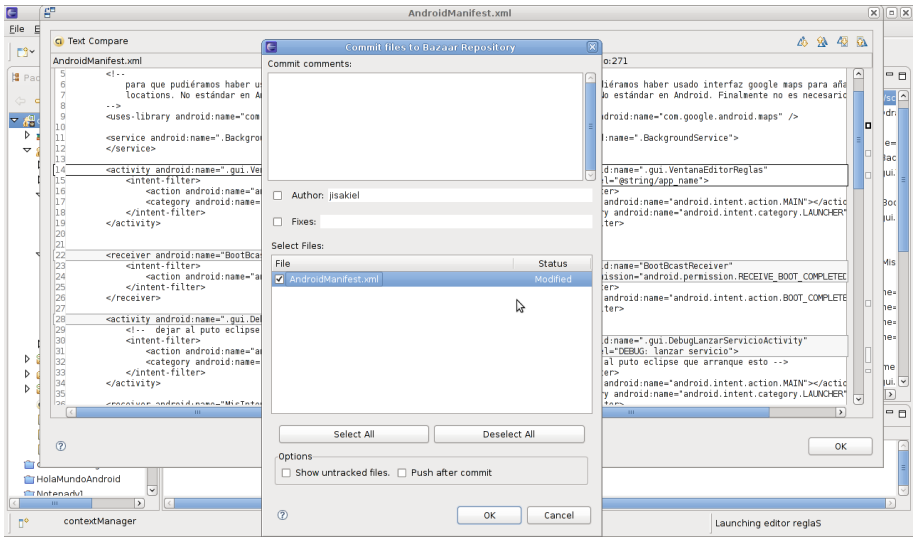

FIGURA 5.5: Plugin Bazaar para Eclipse

gramas de clases están hechos con Netbeans [85] y su *plugin* de modelado UML [86], y el resto de gráficos están hechas mediante Openoffice.org Impress [87].

5.2. Desarrollo sobre la plataforma Android

Elegir Android como plataforma sobre la que desarrollar nos evitó tener que implementar grandes bloques de funcionalidad bastante complejos, principalmente la involucrada en la recogida de la información del entorno del teléfono móvil. Por ejemplo, el propio API implementa la gestión de avisos al entrar en localizaciones con la llamada *addProximityAlert* [79], que nos permite registrar acciones a realizar cuando se entra en ciertas coordenadas, e implementa directamente toda la gestión de energía y la elección del proveedor de posiciones más adecuado para la precisión requerida. Sin embargo nos obligaba a modificar el diseño de nuestro código para adaptarnos a las restricciones y exigencias de Android.

Algunos de los conceptos fundamentales en Android son el *Intent* (usado para paso de mensajes o *intenciones*), que puede ir dirigido a una clase, con un contenido que determina quién lo recibe a partir de filtros, o lanzado a todo el sistema como *BroadcastIntent*. Las aplicaciones se separan en varios componentes: *Servicios*, equivalentes a demonios en terminología Unix -procesos ejecutándose sin requerir entrada directa del usuario-, *BroadcastReceivers* o receptores de *broadcasts*, y *Activities* que conforman el interfaz de usuario. Todas ellas (incluidas las distintas Activities que forman la secuencia de pantallas de una aplicación) se lanzan mediante Intents. Además, el ciclo de vida de cada componente lo controla en general Android y no el programador. Para información más detallada, se remite al lector al anexo A.

Nuestra primera aproximación al problema fue equivocada al asumir un comportamiento a la hora de recoger la información que no se dio. De la documentación que existía sobre Broadcast Intents - que posteriormente cambió cuando actualizaron la plataforma a su versión 1.1 - deducimos incorrectamente que todos los eventos del sistema (llamadas, etc) producían su correspondiente Intent, cuando para la mayoría de ellos (llamadas, localización, hora, etc) hay que solicitarlo explícitamente al sistema y van asociados a un *PendingIntent* (o Intent pendiente de lanzarse). Asumir que se lanzaban como Broadcast suponía un problema con el ciclo de vida, ya que para desregistrar un oyente que se ha registrado dinámicamente (por ejemplo al desactivar una regla) es necesario pasar el objeto con el que se ha registrado, que habría cambiado entre ciclos del proceso. Analizamos varias posibilidades (serializar el objeto, que nos valdría para el almacenamiento a largo plazo, o dejar el oyente registrado hasta que se lanzase, momento en el que él mismo tendría la responsabilidad de desregistrarse), que finalmente no llegamos a implementar por no hacerse necesarias.

Finalmente se hizo patente que se podía solicitar a los distintos gestores o Mana-

gers del sistema que avisasen mediante un PendingIntent. Por tanto se adaptó el ciclo de vida: en principio dejaríamos el servicio en ejecución (aunque no era necesario, ya que dichos PendingIntent podían elevarlo de nuevo al producirse eventos para tratarlo), y lanzaríamos mensajes al mismo mediante Intents (que no lo vuelven a instanciar si ya está instanciado) para las distintas operaciones: desactivar o activar una regla, borrarla, añadirla, etc. Así, evitamos la complejidad del IPC. Todos los eventos están pensados para que sean idempotentes (esto es, que registrarlos varias veces no cambie su resultado), por lo que en caso de apagar el servicio basta con detectarlo sobreescribiendo los métodos adecuados. A continuación es necesario planificar mediante el gestor de alarmas volver a levantarlo, y entonces se vuelven a registrar todos los Intents, que se lanzarán de nuevo si es procedente para activar los contextos relevantes en los que se halla el usuario.

Basándonos además en esta idea, decidimos que las ediciones de reglas se harían directamente desde el GUI, que manipularía directamente la base de datos y avisaría al servicio de cambios en una regla. En caso de ediciones, la regla sencillamente se desregistra y se vuelve a instanciar una nueva en memoria, que se encargará de la lógica de su propia activación o desactivación según se producen estos eventos. El servicio, por su parte, almacena estas reglas sobre una tabla *Hash* que utiliza como clave el índice de regla, y cada evento que se produce lleva en el intent una *Uri* (Identificador Unico de Recurso) que identifica a qué regla, y qué parte dentro de la misma, hace referencia, para poder pasar el procesamiento al objeto pertinente de un modo rápido.

Por su parte, la idea de editar directamente sobre la base de datos partió sobre funcionalidades que proporciona Android para mostrar listas apoyadas sobre cursores de la base de datos, que refrescan automáticamente su contenido y generan dinámicamente las mínimas vistas necesarias para mostrar su contenido por pantalla. Esta funcionalidad también existe para arrays y listas, pero al tener las reglas almacenadas en base de datos juzgamos que sería conveniente (ya que se podría editar directamente sobre la misma). La lista de reglas, así como el editor de localizaciones, están basados en esto.

Sin embargo, para el editor de reglas partimos de una implementación apoyada en la base de datos que hubo que desechar. La idea inicial era iniciar una transacción al crear esa ventana, que quedaría abierta mientras se hicieran cambios (cambios que, por tanto, se reflejarían automáticamente en las vistas), y al cerrar la ventana se decidiría si guardar la transacción o descartarla. Esto supuso problemas, ya que SQLite no soporta accesos concurrentes de escritura, y por tanto dejaba la base de datos bloqueada, lo que interfería con el ciclo de vida (al cerrarse automáticamente la

ventana del editor de reglas que queda oculta por el editor de una regla, sus cursores interferían con la base de datos abierta, provocando errores). La reimplementación del editor de una regla fue sobre una regla en memoria creada específicamente para la edición.

Por lo demás, también hubo varias implementaciones desechadas del editor de elementos concretos, buscando abstraer funcionalidad común, ya que un *Dialog* independiente –en la primera de ellas, y a diferencia de un diálogo como clase privada– no da facilidades de ciclo de vida de bases de datos ni permite devolver resultados de un modo sencillo a la aplicación base. Finalmente se reimplementó como *Activity* genérica que muestra vistas contenidas y obliga a sobreescribir los métodos que tratan estas vistas, y se utiliza el estilo de diálogo para ella a fin de que no se note la diferencia.

Asimismo se han implementado un editor de localizaciones que utiliza los API de Google Maps (y por tanto necesita de una clave específica para funcionar, que va asociada al certificado autogenerado para el entorno sobre el que se desarrolla) para seleccionar y editar las localizaciones, y también se ha integrado la aplicación con el selector de contactos, como pruebas de concepto de la integración con la plataforma.

5.3. Estructura de la aplicación

Una vez conocidas las peculiaridades de la plataforma de desarrollo, y tras los citados cambios a los que nos llevaron las primeras implementaciones, finalmente obtuvimos un diseño que nos permitía estructurar las diferentes funcionalidades de la aplicación. En esta sección analizamos la implementación final de la parte de la aplicación referente a la gestión de reglas, y el funcionamiento de éstas.

5.3.1. Estructura de componentes

El funcionamiento del sistema de gestión de reglas de la aplicación queda resumido en el siguiente diagrama 5.6:

Reglas: representan la información completa de una regla, con sus *Contextos*, *Evento disparador* opcional y *Reacciones*. Son la base de funcionamiento de la aplicación, por lo que detallaremos sus componentes y funcionamiento más adelante.

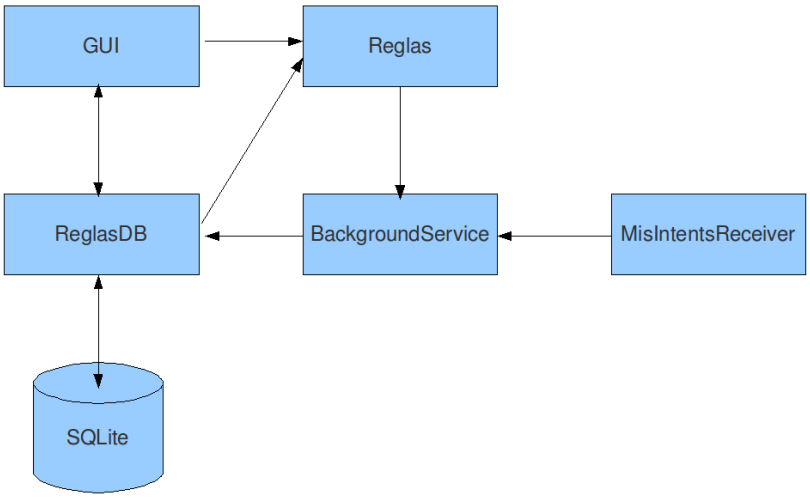

FIGURA 5.6: Diagrama de la estructura de los componentes

MisIntentsReceiver: se trata de un BroadcastReceiver encargado de registrar todos los Intents relevantes para la gestión de reglas de la aplicación, que están detallados en el archivo *AndroidManifest.xml*. Se reciben aquí a fin de que sigan funcionando aún en caso de que se detuviese el servicio, en cuyo caso lo reiniciaría. Cuando recibe un Intent, llama al método *onStart* del servicio *BackgroundService* para su tratamiento.

BackgroundService: es el servicio que hace las funciones de controlador principal de las reglas en memoria de la aplicación, es decir, las que están activas por orden del usuario. El servicio recibe eventos en forma de Intents desde la clase *MisIntentReceiver*, o Intents de la GUI cuando el usuario realiza una modificación, y en función de la acción asociada a estos, realiza la tarea correspondiente. Éstas pueden ser tratar un *Evento*, añadir o borrar una nueva regla en memoria, o activar o desactivar una ya existente.

GUI: está compuesto por todas las clases que conforman la interfaz gráfica de la aplicación. Por un lado, *VentanaEditorReglas* muestra todas las reglas existentes en el sistema, permitiendo editarlas, borrarlas o desactivarlas. El *EditorRegla*, por su parte, permite editar los componentes de una regla concreta o crear una nueva. Asimismo existen el *EditorUbicaciones* para añadir o editar nuevas localizaciones al sistema sobre las que se basen las reglas, y clases auxiliares como un *DialogoE-*

ditor genérico usado internamente o *GoogleMaps* para mostrar una localización concreta apoyándose en esta tecnología.

ReglasDB: se encarga de implementar la funcionalidad de acceso a la base de datos. Lo conforma una clase que contiene constantes para los distintos nombres de tabla y columna, así como las cadenas que crean esas tablas, y hereda de una clase superior que abstrae la funcionalidad común (actualización entre versiones, apertura y cierre, etc). Sus funciones son leer la información de la base de datos y prepararla para mostrarla en el editor de reglas de la GUI, guardar los cambios que el usuario realice a través de la misma, construir objetos regla desde la base de datos, y suministrar la información de las reglas al *BackgroundService* cuando se produce un evento.

La aplicación además está separada en diferentes paquetes:

es.ucm.fdi.pfc: clases principales de la aplicación. Ayudantes varios (de base de datos, de generación de Intents, etc), receptores de Intents de tipo Broadcast, constantes y el servicio que contiene las reglas en memoria.

es.ucm.fdi.pfc.gui: clases de interfaz de usuario

es.ucm.fdi.pfc.reglas: soporte general de las reglas y contextos, así como los subpaquetes correspondientes para eventos y reacciones.

5.3.2. Estructura de clases de Regla

La estructura de las reglas de la aplicación está definida en la figura 5.7, así como los métodos principales de que dispone cada una, omitiéndose *getters*, *setters* y métodos de menor relevancia en pos de mayor claridad.

Como se puede observar, esta estructura de clases representa la definida en la sección 4.2.1, con algunas pequeñas modificaciones realizadas para adecuarse a la plataforma Android.

Una regla contiene una lista de Contextos, otra de Reacciones y opcionalmente un Evento disparador. También mantiene una reacción privada que utilizamos para registrar la activación de la regla con la infraestructura de las reacciones, pero que no presentamos al usuario para su edición o consulta, y por tanto no es necesario almacenar en base de datos.

A su vez, los contextos están compuestos de un evento activador, y opcionalmente uno desactivador – debido a que la implementación de Android recibe el mismo

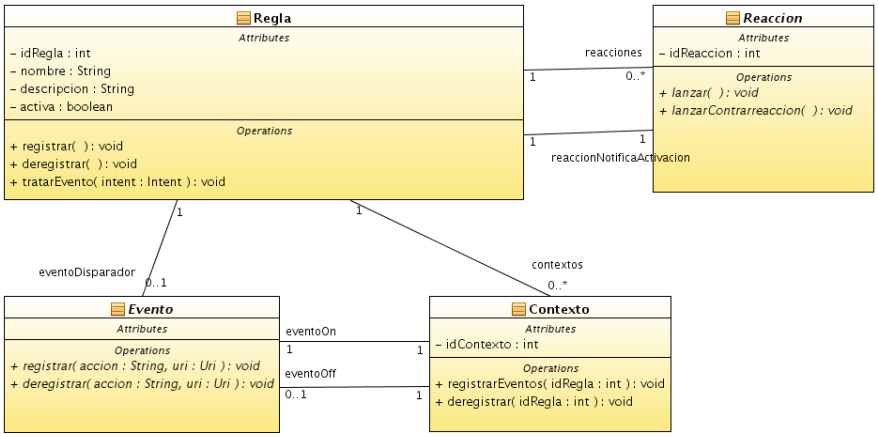

FIGURA 5.7: Diagrama de los componentes de Regla

Intent para ambos, como ese el caso de la localización, donde el propio sistema añade un campo extra al Intent para indicar si se entra o se sale de la misma, y por tanto no existe uno diferente para la desactivación.

Por tanto, los distintos tipos de Evento pueden formar parte de un Contexto y ser disparadores, y heredan de una clase abstracta Evento, al igual que las Reacciones heredan de una clase abstracta Reaccion.

A continuación exponemos la jerarquía de clases de Evento y Reacción que hemos implementado. La elección de éstos viene dada por aquellos que hemos considerado más representativos, así como por las limitaciones de la propia plataforma. La intención del proyecto no es realizar una aplicación que considere todas las reglas posibles, sino implementar la estructura necesaria para que ésta funcione, haciendo que añadir nuevos tipos de eventos y reacciones sea lo más cómodo y rápido posible. Una vez que se cuenta con esa estructura se abren múltiples caminos para el desarrollo posterior del proyecto.

5.3.2.1. Eventos

Los eventos que hemos implementado en la aplicación son Evento Horario, Evento de Localización y Evento Teléfono (llamada entrante).

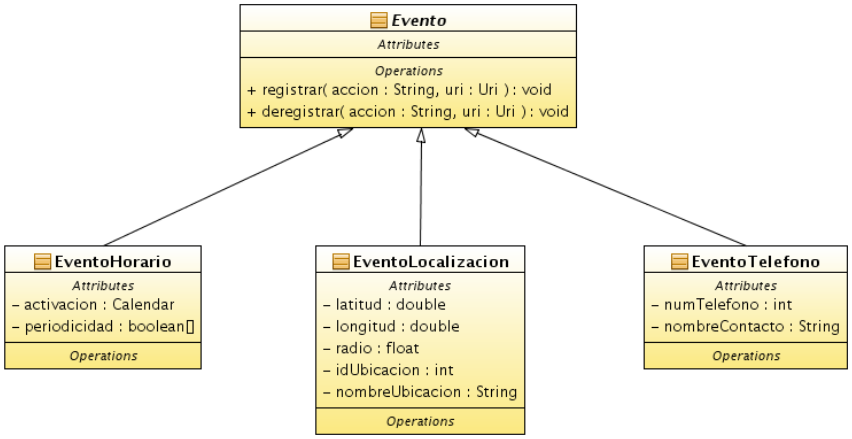

FIGURA 5.8: Diagrama de los clases Evento

Esta elección está basada en el hecho de que estos tres tipos de evento son los más representativos de cara a las acciones más realizadas por un usuario común, y que contemplan los casos de uso principales (sección 3.1) que habíamos considerado, siendo además sencillos de tratar mediante el emulador.

Algunos de los eventos que fueron contemplados en los casos de uso, y que tambien pueden ser implementados de un modo muy directo son:

- Evento Teléfono (SMS entrante).
- Evento Correo (e-mail entrante).
- Evento Batería (el nivel de batería rebasa un porcentaje de su capacidad).
- Evento Wifi (el teléfono detecta una wifi conocida).

Android ofrece también funcionalidad suficiente para implementar muy fácilmente otros tipos de eventos que no estaban contemplados en los casos de uso, como por ejemplo aquellos relacionados con la interacción con redes sociales como *Twitter* [90].

Además hay otra serie de eventos que sí estaban contemplados pero que no fue posible implementar durante el desarrollo por no estar disponibles en el API de Android o en el emulador, como es el caso de partes de la conectividad *bluetooth* o USB [91].

5.3.2.2. Reacciones

Las reacciones que hemos implementado son Reacción Modo Teléfono, Reacción Notificación, Reacción Llamada y Reacción SMS.

La Reaccion Silenciar Llamada se trata de una simulación, que únicamente muestra un mensaje por pantalla indicando que la llamada en curso debería ser silenciada, pero el API de Android no permite actualmente realizar tal operación, que sí puede hacerse pulsando el botón correspondiente en el teléfono. Decidimos incluir esta simulación ya que es una reacción asociada al Evento Teléfono, que sí está implementado, y que habíamos considerado importante en los casos de uso.

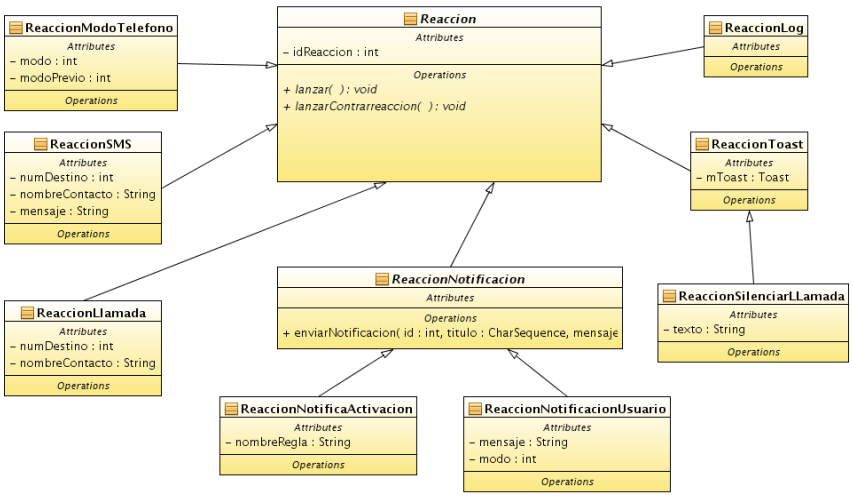

FIGURA 5.9: Diagrama de los clases Reacción

Contamos además con la Reacción Notifica Activación, que no es configurable por el usuario y que sirve para notificarle que una regla ha sido activada. Todas las reglas cuentan con una de ellas privada y por defecto, como se ha indicado anteriormente.

Para utilizar en la depuración hemos implementado además otras dos reacciones, Log y Toast, que respectivamente muestran un mensaje en el registro del sistema o con un mensaje emergente por pantalla.

Al igual que en el caso de los eventos, hay reacciones adicionales que pueden ser implementadas asociadas a los eventos que fueron descartados:

- Reacción Correo (descargar el correo automáticamente, en respuesta a conectarse a Internet).

- Reacción GPS (activar/desactivar, por ejemplo en función de la batería).
- Reacción Localización (guardar la localización actual).

Otras reacciones contempladas y que no están disponibles o no han sido estudiadas en el API son:

- Reacción Red (elegir la red por la que llamar, GSM, GPRS, VoIP, por ejemplo al conectarse a una red inalámbrica).
- Reacción Bluetooth (enviar un mensaje por bluetooth a un contacto conocido).

5.4. Funcionamiento del prototipo

En esta sección ilustramos mediante capturas de pantalla la aplicación desarrollada y el funcionamiento de su interfaz gráfica.

La ventana para editar o añadir reglas (figura 5.10) aparece cuando se lanza la aplicación desde el menú de Android. En cada fila aparece el nombre de esa regla, y opcionalmente una descripción que ayude al usuario a recordar su propósito. Pulsando en la casilla adyacente puede activar y desactivar, y haciéndolo sobre la regla se lanza el editor de ésta. Para borrarlas a su vez se utiliza el menú contextual, presente en todas las interfaces de lista de elementos que hemos desarrollado. Por último, mediante el menú se añaden reglas, y también se puede lanzar directamente el editor de localizaciones.

En la pantalla de edición de una regla (figura 5.11) podremos observar cuatro pestañas. En la primera se añade la información general de la regla, es decir, nombre y descripción. En la segunda, la información de contexto. La tercera pestaña corresponde a la edición del evento disparador, en caso de que éste exista. Finalmente, en la última pestaña podremos elegir las reacciones que deben ocurrir al cumplirse las condiciones de la regla.

A continuación mostramos los pasos seguidos al crear una regla que ponga el móvil en silencio mientras el usuario está en el lugar de trabajo. Al pulsar sobre “añadir regla” se lanza el editor de una nueva regla, permitiendo editar su nombre y descripción.

El contexto relativo sería una localización, “en el trabajo”. La añadimos en la segunda pestaña (figura 5.12), eligiendo en el desplegable el tipo de contexto deseado y pulsando el botón “+”.

FIGURA 5.10: Ventana Principal – Lista de reglas

FIGURA 5.11: Editor de una regla – información general

Al pulsar en "+", por estar seleccionado un contexto de localización nos aparece el diálogo del editor de éste tipo (figura 5.13). En él podemos seleccionar entre una de las preexistentes, o bien lanzar el editor de localizaciones para editar una existente o

FIGURA 5.12: Editor de una regla – elegir un tipo de contexto

crear nuevas.

FIGURA 5.13: Diálogo selector de localización

Lanzamos el editor de localizaciones (figura 5.14) para añadir una nueva. Al entrar

en este editor, observamos la lista de las localizaciones que hemos guardado previamente. Mediante el menú podemos añadir una nueva, y la edición de localizaciones existentes o su eliminación. El uso es modo similar al editor de reglas (pulsando sobre una fila, o dejando pulsado para el menú contextual).

FIGURA 5.14: Editor de localizaciones

El editor de localizaciones utiliza la tecnología de Google Maps (figura 5.15), permitiendo también buscar por nombre un lugar concreto.

Mediante el botón "Fijar centro" podemos fijar un círculo alrededor del área deseada, que conformará la localización (figura 5.16). Su radio es por tanto dependiente del nivel de aumento del mapa, y puede editarse a su vez pulsando sobre el mapa (sin arrastrar) en el modo de movimiento: aparecen dos botones, ausentes de la captura, para acercarse y alejarse. Éstos están ocultos durante el movimiento para permitir mayor área de visión.

Una vez que hemos guardado la localización y añadido el contexto a la regla, por último sólo queda añadir una reacción de modo de teléfono en la última pestaña. Su editor correspondiente (figura 5.17) permite elegir entre los modos del teléfono disponibles, y la reacción sería dicho cambio de modo.

FIGURA 5.15: Editor de una localización

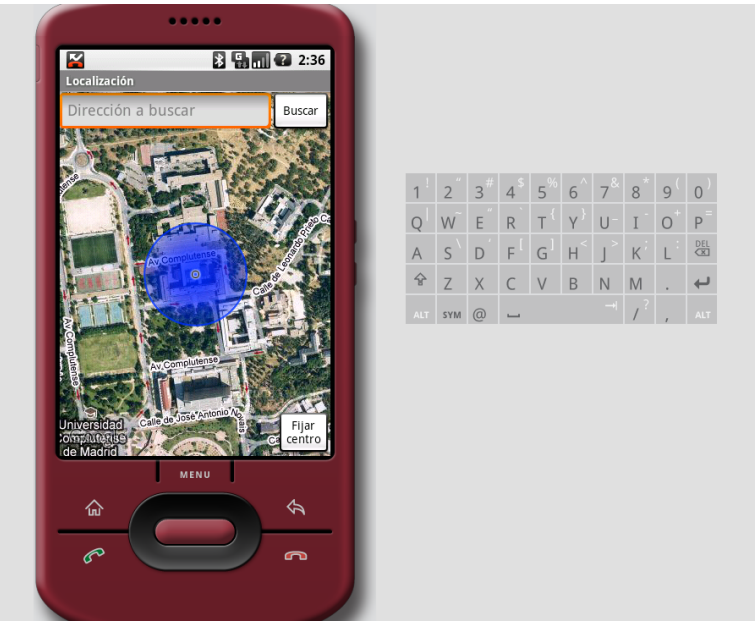

FIGURA 5.16: Editor localización – centro fijado

Finalmente podremos observar que al entrar en el centro de trabajo se activará la regla y obtendremos una notificación correspondiente (figura 5.18). Además, el teléfono pasará a estar en modo silencio.

FIGURA 5.17: Editor de una regla – elegir un tipo de reacción

FIGURA 5.18: Regla activada – notificación

Ejemplificamos también a continuación la creación de otra regla adicional: no aceptar llamadas entrantes de nuestro jefe fuera del horario laboral. Para ello, añadiremos la nueva regla, con un nuevo contexto horario (figura 5.19) formado por las

horas no incluidas en el horario de trabajo (de seis de la tarde a nueve de la mañana, por ejemplo).

FIGURA 5.19: Diálogo editor de contexto horario

Esta regla cuenta con un evento disparador: que la llamada entrante sea de nuestro jefe. Para añadirlo, accedemos a la tercera de las pestañas del editor de una regla (figura 5.20). Al marcar la casilla (parcialmente oscurecida en la captura) de "Pasa algo más" se habilita el selector de tipo de evento que aparece mostrado.

Al seleccionar el tipo de evento "llamada entrante", aparecerá el selector de contactos de la agenda de Android (figura 5.21). Una vez seleccionado el evento, sólo faltaría añadir la reacción correspondiente como hemos visto en el anterior ejemplo, eligiendo en este caso "silenciar llamada entrante".

Finalmente, para completar el último ejemplo de las reglas que veíamos en la primera pantalla de edición de reglas, falta por explicar cómo se añade la regla de haber llegado a una ubicación (al pueblo en el ejemplo). Se trata de una regla únicamente de evento, por lo que no es necesario añadir ningún contexto. El evento disparador es en este caso de localización, por lo que elegimos la correspondiente. En la última pestaña, de selección de reacciones, elegimos finalmente "Realizar una llamada" y aparecerá el selector de contactos como se ha visto previamente. Una vez guardada la regla, podemos comprobar cómo al llegar a dicha localización aparece en pantalla

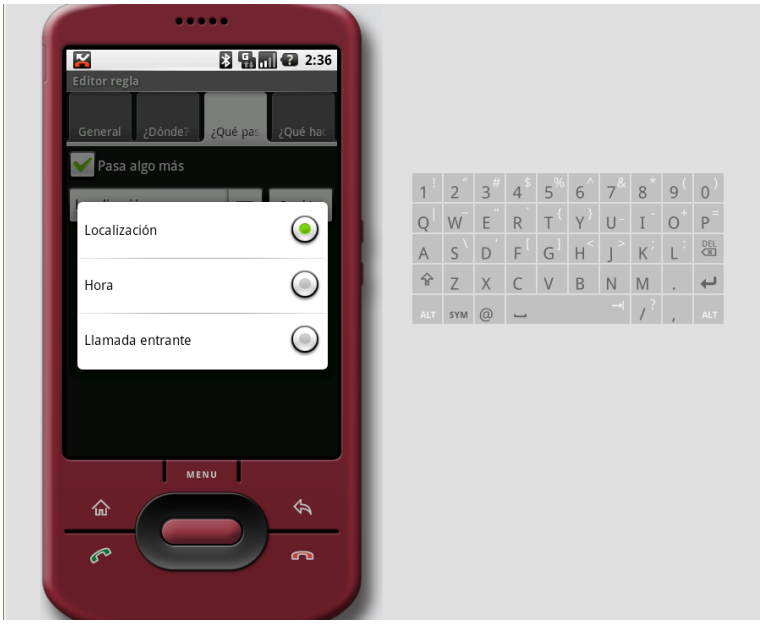

FIGURA 5.20: Editor de una regla – Selector de tipo de evento disparador

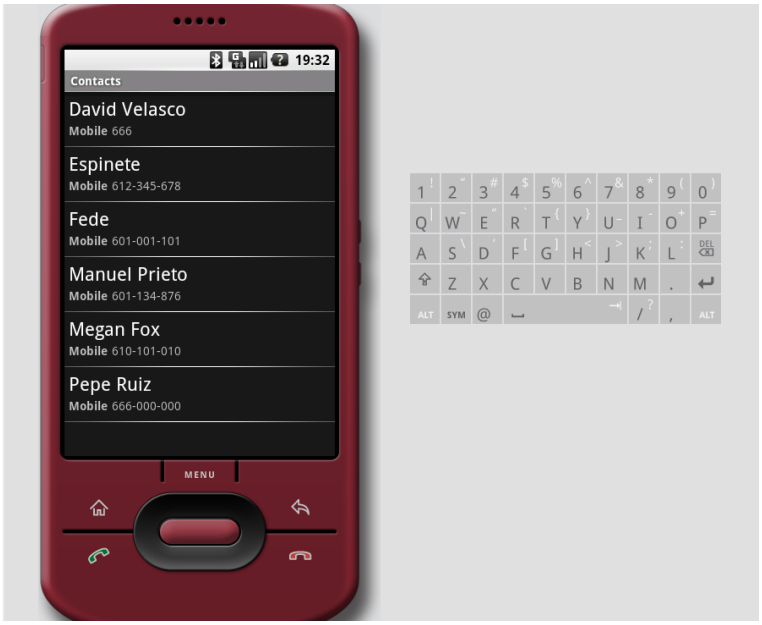

FIGURA 5.21: Diálogo de llamada entrante – selector de contactos

el número de teléfono marcado (figura 5.22), sólo a falta de pulsar el botón de marcar (ya que no tiene sentido que el teléfono llamase por sí mismo si el usuario no está preparado para hablar). En la parte superior podemos observar la notificación de que

la regla se ha activado.

FIGURA 5.22: Regla activada – realizar una llamada

Capítulo 6

Conclusiones

6.1. Alcance

La idea inicial del proyecto resultó ser bastante ambiciosa debido a tener que implementar desde cero toda la infraestructura de código necesaria para soportar las reglas, antes de poder entrar en los aspectos relativos a la Inteligencia Artificial. En esa infraestructura son imprescindibles no sólo la lógica del comportamiento de las reglas, sino también una interfaz de usuario para poder editarlas y visualizarlas, y un sistema de almacenamiento para guardar los cambios hechos por el usuario. Además, en el análisis previo de las diferentes técnicas de Inteligencia Artificial que podían ser de utilidad pudimos comprobar que ese desarrollo tendría una complejidad considerable y mayor de la prevista, más aun tratándose de un terminal móvil con menor capacidad de proceso y memoria que un PC. Por tanto, optamos por considerarlo un proyecto a largo plazo, y se ha concluido únicamente el desarrollo de la funcionalidad esencial del tratamiento de reglas, siempre teniendo presentes los objetivos finales e intentando facilitar su futuro desarrollo.

Esta infraestructura consta por tanto del soporte de reglas en base de datos SQLite para su almacenamiento mientras el teléfono está apagado, así como del servicio que las construye, evalúa y activa según es necesario, y de un editor gráfico para las reglas y sus componentes. Por otra parte, también integramos la recogida de acciones del usuario y la información de contexto asociada a éstas en una base de datos separada. Sobre dicha base de datos (y teniendo en cuenta, si fuesen necesarias, las reglas definidas) procedería en un futuro desarrollo implementar los métodos de inteligencia artificial para sugerir nuevas reglas.

La estructura de código del proyecto, si bien sólo implementa los eventos y reacciones que consideramos esenciales para probar nuestra infraestructura, está también

diseñada para ser ampliada con más tipos de eventos o reacciones. La licencia libre de nuestro código también favorece sus futuros avances.

Por su parte, cada uno de los componentes de una regla son fáciles de integrar con el resto de elementos de Android para utilizar otras partes del sistema operativo, como de hecho se hace para eventos de llamada entrante o para elegir localizaciones. Además hemos respetado el diseño de la plataforma orientado a componentes independientes y reutilizables, por lo que los nuestros se podrían usar también en terceros proyectos (por ejemplo cualquiera de nuestros editores de componentes, o el editor de localizaciones, sin depender del resto de infraestructura).

6.2. Líneas futuras de trabajo

Dado que el proyecto está planteado a varios años han quedado abiertas muchas posibilidades para continuar desarrollándolo. Los principales caminos a seguir son la mejora y ampliación de la parte del software que se encarga de la gestión de reglas, tanto en la estructura como en la lógica disponible y la usabilidad del interfaz; y, por otro lado, la implementación de técnicas de inteligencia artificial para complementar la aplicación y alcanzar los objetivos iniciales. En esta sección detallamos estas líneas de trabajo:

6.2.1. Mejoras en la estructura del código

A fin de facilitar el futuro desarrollo consideramos que hay una serie de cambios que se podrían hacer, como por ejemplo:

- Cambiar los selectores o *switch* sobre tipos de Eventos o Reacciones por técnicas de *reflection* (análisis en tiempo de ejecución de la estructura de código y clases definidas) sobre los tipos posibles de éstas, para no tener que definirlas de antemano estáticamente.
- Con el fin de hacer el código más robusto, integrar soluciones para evitar la limitación de Java de no poder definir una clase *abstract static* ni métodos estáticos en *interfaces* (esto es, para poder obligar a una clase hija a implementar un método estático). Es previsible que ese problema desapareciese a medio plazo, al estar propuesto como una mejora para la futura versión 7 de Java [88], [89]. Esto haría el código menos propenso a errores de implementación.
- Revisar la implementación de los contextos como dos eventos separados de activación y desactivación, lo que ha dado problemas durante el desarrollo. Es conveniente que un evento conozca las condiciones de desactivación, en caso de existir, lo que simplificaría la lógica.

- Compartimentalizar mejor la base de datos. Por comodidad, el tratamiento de la base de datos es relativamente monolítico, salvo aquellos métodos relativos a clases que pueden ser ampliadas (Evento o Reaccion). Sin embargo, las distintas tablas de apoyo que necesiten esas subclases se definen estáticamente en la clase ReglasDB. Utilizando *reflection*, como en el primer punto, se podría evitar el problema de tener que conocer de antemano estas clases.

6.2.2. Mejoras en la gestión de reglas

Las diferentes formas de ampliar la parte de la aplicación que gestiona la lógica de las reglas que consideramos interesantes de reseñar son:

- Implementar la lógica que condicione los contextos, eventos y reacciones disponibles en función de los ya presentes en una regla. Por ejemplo, no tiene sentido la reacción "silenciar llamada" si no hay un evento "llamada entrante".
- Completar y mejorar la lógica de las reglas, implementando lógica binaria entre los contextos y eventos, por ejemplo lógica disyuntiva (*or*) y negativa (*not*). La dificultad añadida de esta mejora es conseguir mantener la usabilidad a la vez que se introduce más complejidad en la definición de las reglas.
- Ampliar los diferentes tipos de contextos, eventos y reacciones disponibles, con algunos de los ejemplos expuestos en la sección **??** u otros nuevos que completen casos de uso adicionales, y mejorar los ya existentes.
- Diversas mejoras de eficiencia son posibles pero no se han implementado por haber priorizado terminar el desarrollo a optimizar. Algunas de ellas son:
  - Registrar las reglas en hilos separados del principal para mejorar el tiempo de respuesta.
  - Minimizar el número de Intents que registrar para reducir la sobrecarga del sistema. Podría hacerse de varios modos: por ejemplo, reestructurando el código para que distintas reglas pudieran compartir contextos (y así contextos repetidos no obligan a registrar por repetido), o cambiando la gestión de alarmas para tener todos los eventos horarios ordenados por tiempo de activación (y así solo tener el siguiente registrado, de modo similar a como implementan en su código el código para las alarmas del teléfono).
  - Relacionado con el punto anterior, mejorar el consumo de energía. Por ejemplo, usar distintos proveedores de localización como GSM (posicionamiento a partir de triangulación sobre torres de ubicación fija conocida) hasta que sea necesaria la precisión que da el GPS, evitando encenderlo lo más posible.

- Sería interesante también evaluar la diferencia de rendimiento respecto de soportar las reglas directamente sobre la base de datos, eliminándolas de memoria. Podrían añadirse columnas para almacenar qué contextos están activos en cada momento. Ello consolidaría la información en un único punto y facilitaría todas las mejoras arriba citadas (la lógica se implementaría sobre clases ayudante), a cambio de posible lentitud en los accesos por la sobrecarga de las conexiones y consultas. También haría necesario solventar las dificultades (ver sección 5.2) ya encontradas para editar una regla directamente sobre la base de datos, permitiendo deshacer los cambios, y sin usar para ello transacciones.

6.2.3. Otras mejoras y pruebas

- Mejorar el GUI, por ejemplo aprovechando pruebas de usabilidad con usuario real y en escenarios reales.
- Pruebas en un terminal real.
- Exponer las reglas mediante un *ContentProvider* para facilitar aún más el uso de nuestra aplicación a otras.
- Modificar los eventos para que se registren dinámicamente como extensiones (*plugins*) a la aplicación. Así, la aplicación base podría llevar un conjunto de eventos predefinido, y el resto se podrían instalar, incluso manteniendo un registro de plugins conocidos que se pudieran descargar directamente.
- Permitir desactivar tipos de eventos según los permisos de seguridad que requieran. Así, combinado con el punto anterior, el usuario podría decidir qué permisos le otorga a la aplicación y en función de ello se registrarían unos plugins u otros, y no sería necesario concederle permisos completos monolíticamente (para capturar llamadas, para capturar mensajes, para enviar mensajes, para cambiar modo, para acceder a Internet...), que puede preocupar al usuario sin motivo y prevenirle de que instale la aplicación.
- Refactorizar las cadenas para permitir internacionalización, ya que la plataforma lo permite.
- Traducir el código y comentarios al inglés para aumentar sus posibilidades de ser ampliado por terceras personas.

6.2.4. Técnicas de Inteligencia Artificial

Como hemos comentado en la sección 4.3, la aplicación cuenta con un sistema de registro de las acciones del usuario, así como de la información de contexto asociada

a éstas, con el fin de implementar técnicas de Inteligencia Artificial para analizar los patrones de comportamiento del usuario y poder ofrecer recrear automáticamente las acciones cotidianas que éste realiza. Partiendo de esa información, disponemos de dos técnicas distintas para utilizar, descritas en su apartado previo correspondiente: inferencia de reglas y algoritmos bayesianos.

En el primer caso, consideraríamos esta nueva funcionalidad como una ampliación de la ya existente en la aplicación. Las reglas inferidas por el sistema serían mostradas al usuario para su aprobación, y aquellas que fuera aceptadas se incorporarían a la base de datos de reglas como si se tratara de una creada por el usuario. Adicionalmente podría eliminarla o editarla como cualquier otra.

En el caso de los algoritmos bayesianos, contamos con una funcionalidad diferente. El clasificador calcula un porcentaje de probabilidad para cada posible "estado" en función de las condiciones actuales, por lo que no podríamos obtener reglas fijas sino la respuesta a la pregunta *¿qué debería hacer el teléfono en este momento según las costumbres del usuario?* De esta forma consideramos que la implementación de este tipo de algoritmos conllevaría una nueva funcionalidad de la aplicación o un *"modo inteligente"*, en el que, al activarlo, el usuario deja en manos del algoritmo las decisiones que deben tomarse en cada momento. Pensamos que esta nueva opción puede resultar interesante como complemento a las reglas fijas, aunque la probabilidad de obtener una situación anómala es mayor que en una regla aceptada por el usuario, lo que es un inconveniente y un posible problema de usabilidad. Otra opción consistiría en preguntar al usuario antes de realizar los cambios de estado, pero esto atenta contra el principio de nuestro proyecto de automatizar al máximo las acciones cotidianas del usuario, y podría llegar a convertirse en algo molesto para el mismo.

Para ambas opciones, la implementación estaría siempre limitada por el hardware sobre el que se ejecuta (un dispositivo embebido de capacidades limitadas, mientras que la mayoría de las técnicas habituales están más destinadas a grandes capacidades de cómputo y de tiempo). Para evitar afectar al consumo energético se podría limitar dicho postproceso de los eventos al momento en el que el teléfono móvil esté cargando su batería, por ejemplo, o realizar un preprocesado mínimo aprovechando los tiempos en que se despierte la CPU de sus modos de menor consumo (por ejemplo al recibir un evento del usuario, o incluso al desbloquear la pantalla).

Como mejora a lo anteriormente citado, y de no ser posible hacerlo en el propio teléfono, se podría externalizar el procesamiento de sugerencia de reglas a un ordenador personal cuando se conectase a él el teléfono, lo cual requeriría de toda una infraestructura adicional de código pero permitiría implementar algoritmos mucho

más complejos, así como abriría toda una gama de posibilidades de sincronizar con el PC.

Apéndice A

Plataforma Android

Android [94] es una plataforma de desarrollo orientada a teléfonos móviles, implementada sobre un kernel de linux modificado y una librería de C (*libc*) propia – por motivos de eficiencia y licencias. Se ejecuta sobre el conjunto de instrucciones (*ISA*) *ARMv5TE*, si bien existen versiones en *ARMv4* y *x86*. Expone funcionalidad al programador mediante un lenguaje basado en Java pero con su propio API y biblioteca de clases, que incorpora la mayor parte de las llamadas y funcionalidad de la edición estándar de Java (*J2SE* [92]), pero modificando fuertemente los API gráficos y el modelo de procesos para adaptarse mejor a una plataforma embebida, y ofreciendo mucha funcionalidad integrada (incluso un componente para navegación por Internet basado en Webkit [93]). Ello lo hace incompatible con otros sabores de java como *J2ME* [18], a cambio de ofrecer a Google la máxima flexibilidad en la definición y desarrollo de la plataforma.

La máquina virtual, llamada *Dalvik*, está diseñada por Google y ejecuta su propio formato de bytecode de nombre *DEX*, sobre una máquina de registros (a diferencia de la estándar Java, que se basa en una arquitectura a pila). Está especialmente optimizada para minimizar consumo de memoria, con características como conversión automática de XML a un formato binario más optimizado, y otras como cambio automático, al cargar, del ordenamiento de bytes en función de la arquitectura destino.

Los procesos están controlados por el *runtime* o ejecutable de Android, parte del sistema operativo, sin exponer la funcionalidad de creación (*fork*, *exec*, etc.) al desarrollador. Cada proceso ejecuta una máquina virtual independiente que le contiene, evitando con ello problemas de seguridad. Además incluye diversas optimizaciones para reducir la sobrecarga en memoria (en esencia las máquinas virtuales comparten la mayor parte de su memoria entre ellas), así como el tiempo de creación de procesos

mediante instanciado bajo demanda de procesos contenedores.

El desarrollador no tiene control directo sobre el ciclo de vida [95] –salvo algunas funciones concretas–, sino que es Android quien decide cuándo se instancian los componentes y cuándo salen del ciclo de vida en función de la presión de memoria, y siempre priorizando la actividad con la que interactúa el usuario y aquellos servicios a los que estuviera unida y que por tanto necesite.

El paso de mensajes entre componentes se realiza mediante el *Intent* [96], o "intención", que es un mensaje entre dos componentes independientes, conteniendo opcionalmente características como una cadena con su acción, una URI que determina el destino, una categoría, o campos extras (en general sencillos, o en su defecto serializables) en un formato llamado *Bundle* o paquete. También existe el concepto de *PendingIntent* o "intención pendiente", mecanismo por el cual se cede permisos (incluso a otra aplicación) para lanzar un Intent en un futuro.

Un componente puede exponer a qué intents responde dinámicamente o estáticamente mediante filtros para las características del mismo. Al registrarlo es el propio sistema quien, cuando recibe un Intent que no tiene destino explícito, lo encamina hacia el componente que se ajuste a sus características.

Con ello, el concepto de aplicación unificada muta por uno de federación de componentes: las distintas pantallas que componen una aplicación se instancian independientemente, y se pueden registrar en el manifiesto del paquete (*AndroidManifest.xml* [97]) proveedores de contenido o *ContentProviders* para exponer datos de la aplicación a las demás de un modo unificado. Así, por ejemplo, para escoger un contacto de la agenda una aplicación simplemente necesita lanzar un Intent con la URI *People.CONTENT_URI*, que se traduce a *"contact://people/"*, y con la acción *Intent.ACTION_PICK* (escoger), y se mostrará la aplicación del sistema para elegir un contacto. Las aplicaciones desarrolladas, por su parte, también pueden exponer sus datos de este modo, e incluso sobreescribir a las propias del sistema (y que se mostrase otra aplicación en su lugar).

Por su parte, existen varios tipos de componentes principales:

- Por un lado, la presentación gráfica al usuario se hace mediante *Activities* (actividades [98]), cada una de las cuales instanciadas por el sistema mediante un Intent, que conforman entre ellas una tarea o *Task* que representa la pila de ventanas que se ha ido abriendo. De este modo, si se ven expulsadas de memoria porque el usuario cambie de aplicación, entre una llamada, cambie la orienta-

ción del teléfono, etc. tienen oportunidad de guardar su contenido en *Bundles* para cuando vuelven a estar en pantalla, momento en que se restaura la pila de ventanas completa (pero solo se instancia la más externa, que es la que se muestra al usuario). Existen también otras funcionalidades como diálogos (*Dialog*) o popups de información (*Toast*), así como notificaciones persistentes mediante *NotificationManager*. Además, la definición de los layout o presentación de elementos gráficos se puede construir tanto en código como mediante XML.

- Por otra parte, para procesos corriendo constantemente aparece el servicio o *Service* [99], similar al concepto de demonio de Unix. Para conectarse al servicio se puede invocar mediante Intents, o conectarse (*bind*) al mismo mediante un mecanismo de comunicación entre procesos (IPC) basado en el lenguaje de descripción de interfaces *IDL*, llamado *AIDL* (Android IDL [100]).

- Se pueden recibir y enviar notificaciones en broadcast (a todos los procesos registrados para recibirlas) aunque el proceso no esté ejecutándose, y tanto los filtros registrados en el manifiesto como los receptores registrados en tiempo de ejecución determinan qué BroadcastReceiver recibe estas notificaciones. Estos receptores están limitados por Android a 10 segundos de procesamiento desde que se instancian (para evitar que el sistema se ralentice o cuelgue), por lo que para ejecutar tareas más largas se deben delegar a un servicio.

- Android también ofrece facilidades para el almacenamiento de datos además de la citada de los *ContentProviders*, con almacenamiento de pares clave-valor para preferencias, acceso a sistema de archivos y una base de datos SQLite [101] integrada en el sistema.

Con respecto a la seguridad de la plataforma [102], la mayor parte de la misma la provee el kernel de Linux, ya que cada aplicación corre en un proceso independiente (por lo tanto con su espacio propio de memoria), bajo un UID o identificador de usuario único por aplicación y autogenerado, si bien existen modos de compartir datos entre aplicaciones. Existen también permisos de grano más fino que imponen restricciones sobre las operaciones que puede realizar un proceso, siendo la filosofía de diseño que los permisos por defecto no puedan hacer nada que afecte negativamente a otras aplicaciones, al sistema operativo, o al usuario. Así, por ejemplo, para que una aplicación pueda utilizar datos de posicionamiento, conectarse a Internet o mandar mensajes, debe ir especificado que requiere permisos adicionales en su manifiesto [103] –en caso contrario falla en tiempo de ejecución–, y se pedirá confirmación al usuario a la hora de instalarla.

Asimismo, los paquetes que contienen la aplicación (de extensión *.apk*) deben ir firmados criptográficamente mediante un certificado, que puede ser autogenerado

o distribuido por Google al registrarse como desarrollador. De este modo, las aplicaciones disponibles en el *Android Market* (la tienda de aplicaciones presente en los teléfonos) van asociadas a un responsable, y además pueden establecer relaciones de confianza –a fin de compartir datos, por ejemplo– entre ellas basándose en dicha firma, pero al mismo tiempo un desarrollador independiente puede distribuir su aplicación fuera de estos canales (a diferencia de en otras plataformas alternativas).

Por último, alguna de la funcionalidad extra que provee Android es soporte para Logging o registro de mensajes de error o informativos, accesibles mediante la herramienta de depuración *logcat*, soporte para unidades de pruebas (testunit), soporte de Threading como en J2SE, etc.

Además, recientemente ha sido anunciado un entorno de desarrollo de scripts o guiones [104] soportando varios lenguajes: Lua [105], BeanShell [107] y Python [106], con otros pensados a añadir como Ruby [108] y JavaScript [109]. Mediante el mismo se expone la funcionalidad de buena parte del API [110] (tratamiento de Intents o Activities, hacer llamadas o mandar mensajes, obtener datos de posicionamiento, etc) a estos lenguajes, que permiten un desarrollo muy rápido, y además es posible ejecutar scripts desarrollados en el propio teléfono, o en modo interactivo desde la consola [111].

También muy recientemente ha sido anunciada la disponibilidad del NDK (Native Development Kit) [38], soportado a partir de la versión de *firmware* 1.5, que permite desarrollar en código nativo contra las cabeceras de la plataforma (C ó C++ en principio, si bien la disponibilidad de cabeceras permitiría desarrollar en cualquier lenguaje que compilase a *ARMv5TE*). Esto posibilita obtener mayor rendimiento del hardware, si bien impide la portabilidad inmediata del software, que sí permite, en cambio, la opción de desarrollar íntegramente en Java, y tampoco permite utilizar el API. Las bibliotecas facilitadas contra las que enlazar son *libc* (la librería de C estándar), *libm* (librería de matemáticas), *libz* (compresión *Zlib*), *liblog* (logging o recogida de errores de Android), *JNI* (*Java Native Interface*, para enlazar el resto de la aplicación escrita en Java, que sí podría usar el API o contener un interfaz de usuario, con el código nativo), y otras librerías para soportar aplicaciones escritas en C++. No obstante, está planificado [112] soportar en el futuro librerías de *OpenGL ES* [113] y de audio, para el desarrollo de juegos.

El entorno de desarrollo en Java [114], por su parte, permite trabajar tanto mediante un plugin para Eclipse, como directamente sobre el sistema de archivos, generando archivos de *Ant* [115] sobre plantillas de estructura de directorios para construir los paquetes. En su versión 1.5 permite elegir el objetivo contra el que se

construyen los paquetes, así como desplegar varias versiones del *firmware* (1.1 con añadidos de Google como *Maps* o *Gmail*, estos opcionales según decida el fabricante del teléfono pero presentes en los primeros terminales; 1.5 con añadidos de Google; o 1.5 sin ellos), así como personalizar el hardware disponible en el emulador, o cargar *firmwares* compilados por el propio usuario a partir del código fuente de Android, disponible bajo licencia Apache 2.0 [116] en `http://source.android.com/`. El emulador, por su parte, está basado en *qemu* [117].

Bibliografía

[1] Cinco Días. La telefonía móvil supera en España el número de habitantes. http://www.cincodias.com/articulo/empresas/telefonia-movil-supera-Espana-numero-habitantes/20060605cdscdiemp_5/cdsemp/, Abril 2006.

[2] Tomas Klockar, David A. Carr, Anna Hedman, Tomas Johansson, and Fredrik Bengtsson. Usability of Mobile Phones. Diciembre 2003.

[3] El País. El futuro que imagina Intel en movilidad y salud. http://www.elpais.com/articulo/semana/futuro/imagina/Intel/movilidad/salud/elpeputeccib/20080619elpciblse_2/Tes, Junio 2008.

[4] The H open Heise Media U.K. Intel and Nokia partner on open source mobile computing. http://www.h-online.com/open/Intel-and-Nokia-partner-on-open-source-mobile-computing--/news/113599, Junio 2009.

[5] Wikipedia. The History of Smartphones. http://en.wikipedia.org/wiki/Smartphone#History, Junio 2009.

[6] Anandtech. The Palm Pre Review. http://www.anandtech.com/gadgets/showdoc.aspx?i=3586&p=11, Junio 2009.

[7] Movistar. Tarifas de acceso a Internet. http://www.movistar.es/fwk/cda/controller/comun/0,2188,8887_154348879_154348885_0_0,00.html, Junio 2009.

[8] Vodafone. Tarifa Plana Internet Móvil. http://www.live.vodafone.es/TarifaPlana/navegacion/, Junio 2009.

[9] IDG. Diez tendencias en teléfonos móviles vistas en el Mobile World Congress. http://www.idg.es/cio/mostrarNoticia.asp?id=64852&seccion=wireless, Febrero 2008.

[10] HTC. T-Mobile G1 Specification. http://www.htc.com/www/product/g1/specification.html, Junio 2009.

[11] HTC. HTC Magic Specification. http://www.htc.com/www/product/magic/specification.html, Junio 2009.

[12] Microsoft. Windows Mobile. http://www.microsoft.com/windowsmobile/es-es/default.mspx, Junio 2009.

[13] RIM. Blackberry. http://es.blackberry.com/, Junio 2009.

[14] Symbian Foundation. Symbian. http://www.symbian.org/, Junio 2009.

[15] IDG. Symbian se mantiene fuerte en el mercado de los smartphones. http://www.idg.es/comunicaciones/noticia.asp?id=63174, Diciembre 2007.

[16] Gartner. Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 3.7 Per Cent Increase in Fourth Quarter of 2008. http://www.gartner.com/it/page.jsp?id=910112, Marzo 2009.

[17] Forbes. Nokia Opens Up Symbian. http://www.forbes.com/2008/06/24/nokia-symbian-software-markets-equity-cx_ll_0624markets09.html, Junio 2008.

[18] Sun Microsystems. Java ME. http://java.sun.com/javame/index.jsp, Junio 2009.

[19] Maemo. Maemo. http://maemo.org/, Junio 2009.

[20] Maemo. Intro: Software Platform. http://maemo.org/intro/platform/, Junio 2009.

[21] Apple. iPhone: Teléfono móvil, iPod y dispositivo de acceso a Internet. http://www.apple.com/es/iphone/, Junio 2009.

[22] Google. Android Developers. http://developer.android.com/, Junio 2009.

[23] Open Handset Alliance. Open Handset Alliance. http://www.openhandsetalliance.com/, Junio 2009.

[24] Open Handset Alliance. Open Handset Alliance Members. http://www.openhandsetalliance.com/oha_members.html, Junio 2009.

[25] Android x86. Android x86. http://www.androidx86.org/, Junio 2009.

[26] Digital Trends. Acer Brings Timeline Notebooks to U.S., Plans Android Netbooks. http://news.digitaltrends.com/news-article/20085/acer-brings-timeline-notebooks-to-u-s-plans-android-netbooks, Junio 2009.

[27] Engadget. ASUS experimenting with Android-based netbook. http://www.engadget.com/2009/02/20/asus-experimenting-with-android-based-netbook/, Febrero 2009.

[28] Trolltech. Qt Extended Hardware. http://qtopia.net/modules/devices/, Junio 2009.

[29] Trolltech. Qt Software discontinues Qt Extended. http://www.qtsoftware.com/about/news/qt-software-discontinues-qt-extended, Mayo 2009.

[30] Openmoko Inc. Openmoko. http://www.openmoko.com/, Mayo 2009.

[31] Michael Shiloh. Freerunner CAD Files - IGES ans STEP formats availalbe - Who wants to test them? http://lists.openmoko.org/pipermail/community/2008-July/022202.html, Julio 2008.

[32] Jan Wedekind. Openmoko discontinued? http://www.wedesoft.demon.co.uk/openmoko-discontinued.html, Abril 2009.

[33] Aspicore. GSM Tracker Help. http://www.aspicore.com/gsmtrackerhelp/v110/default.htm, Junio 2009.

[34] Epocware. Handy Profiles para Nokia 5800 Xpress Music. http://nokia-5800-xpress-music-software.epocware.com/es/Handy_Profiles.html, Junio 2009.

[35] Two Forty Four AM. Locale for Android. http://www.twofortyfouram.com/, Junio 2009.

[36] Versatile Monkey. An Experiment in Blackberry Development. http://www.versatilemonkey.com/story.html, Junio 2009.

[37] Apple. iPhone Development Guide Introduction. http://developer.apple.com/iphone/library/documentation/Xcode/Conceptual/iphone_development/000-Introduction/introduction.html, Junio 2009.

[38] Google. Android 1.5 NDK, release 1. http://developer.android.com/sdk/ndk/1.5_r1/index.html, Junio 2009.

[39] ARM. ARM Processor Instruction Set Architecture. http://www.arm.com/products/CPUs/architecture.html, Junio 2009.

[40] The Linux Information Project. The dangers of tivoization and the GPLv3. http://www.linfo.org/tivoization.html, Enero 2007.

[41] Openmoko Wiki. Neo FreeRunner Hardware. http://wiki.openmoko.org/wiki/Neo_FreeRunner_GTA02_Hardware, Junio 2009.

[42] Openmoko Wiki. Distributions. http://wiki.openmoko.org/wiki/Distributions, Junio 2009.

[43] Openmoko Wiki. OpenEmbedded. http://wiki.openembedded.net/index.php/OpenMoko, Junio 2009.

[44] Debian Wiki. Debian on FreeRunner. http://wiki.debian.org/DebianOnFreeRunner, Junio 2009.

[45] Openmoko Wiki. Debug Board v3. http://wiki.openmoko.org/wiki/Neo1973_Debug_Board_v3, Junio 2009.

[46] Openmoko Wiki. GPS Problems. http://wiki.openmoko.org/wiki/GPS_Problems, Junio 2009.

[47] Werner Almesberger. The GTA02 Charger Issue. https://people.openmoko.org/werner/gta02-chg/, Julio 2008.

[48] Jeremy Chang. Status report about suspend/resume issues. http://www.mail-archive.com/devel@lists.openmoko.org/msg02597.html, Octubre 2008.

[49] Michael Stather. The glamo chip and its future. http://lists.openmoko.org/pipermail/community/2008-June/020040.html, Junio 2008.

[50] Brian Fuller. Openmoko Neo FreeRunner GTA02 versions A5 and A6 Audio Buzz - Quality Enhancement Service. http://lists.openmoko.org/pipermail/community/2009-June/049102.html, Junio 2009.

[51] Trolltech. Qt Toolkit - Collection Classes. http://www.handhelds.org/~zecke/apidocs/qt/collection.html, Diciembre 2001.

[52] Trolltech. Signals and Slots. http://doc.trolltech.com/4.1/signalsandslots.html, Diciembre 2006.

[53] Jochen Hipp, Ulrich Güntzer, Gholamreza Nakhaeizadeh, and Daimlerchrysler Ag. Algorithms for association rule mining - a general survey and comparison. 2000.

[54] Mohammed J. Zaki and Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):462–478, 2005.

[55] Lotfi Lakhal and Gerd Stumme. Efficient mining of association rules based on formal concept analysis. 3626:180–195, 2005. http://www.springerlink.com/content/31aa2gtve59x3jh1/.

[56] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. pages 398–416, 1999.

[57] Wikipedia. Association rule learning. `http://en.wikipedia.org/wiki/Association_rule_learning`, Junio 2009.

[58] Burleson Consulting. All about beer and diapers. `http://www.dba-oracle.com/oracle_tips_beer_diapers_data_warehouse.htm`, Marzo 2008.

[59] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. pages 487–499, 1994.

[60] Mohammed J. Zaki. Scalable algorithms for association mining. pages 372–390, Mayo / Junio 2000.

[61] R. J. Bayardo. Efficiently mining long patterns from databases. page 85–93, Junio 1998.

[62] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[63] Stuart J. Russell and Peter Norvig. Artificial intelligence: A modern approach (international edition). page 597, November 2002.

[64] . , Junio 2009.

[65] The Eclipse Foundation. Eclipse.org. `http://eclipse.org/`, Junio 2009.

[66] Software in the Public Interest. Debian – The Universal Operating System. `http://debian.org/`, Junio 2009.

[67] Canonical. Ubuntu Home Page. `http://www.ubuntu.com/`, Junio 2009.

[68] Brett Chabot. GPS emulation issues in emulator. `http://code.google.com/p/android/issues/detail?id=2545`, Junio 2009.

[69] Debian. Exact hits: Package ia32-sun-java6-bin. `http://packages.debian.org/search?keywords=ia32-sun-java6-bin`, Junio 2009.

[70] Ubuntu. Exact hits: Package ia32-sun-java6-bin. http://packages.ubuntu.com/search?keywords=ia32-sun-java6-bin&searchon=names&suite=all&section=all, Junio 2009.

[71] Henrik V. Eriksson. Location Controls in eclipse DDMS view not send to emulator. http://code.google.com/p/android/issues/detail?id=915, Junio 2009.

[72] Google. Android Emulator. http://developer.android.com/guide/developing/tools/emulator.html, Junio 2009.

[73] Google. <activity>- android:process. http://developer.android.com/guide/topics/manifest/activity-element.html#proc, Junio 2009.

[74] Canonical. Bazaar Version Control. http://bazaar-vcs.org/, Junio 2009.

[75] Canonical. Olive - Bazaar Version Control. http://bazaar-vcs.org/Olive, Junio 2009.

[76] Meld Developers. Meld: Diff and merge tool. http://meld.sourceforge.net/, Junio 2009.

[77] Guillermo Gonzalez. Bazaar XML output plugin. http://bazaar-vcs.org/BzrEclipse, Junio 2009.

[78] Guillermo Gonzalez. Bazaar XML output plugin. https://launchpad.net/bzr-xmloutput, Junio 2009.

[79] Google. Android API - LocationManager: addProximityAlert . http://developer.android.com/reference/android/location/LocationManager.html#addProximityAlert, Junio 2009.

[80] Gentoo. Gentoo Hardened. http://www.gentoo.org/proj/en/hardened/, Junio 2009.

[81] OpenBSD. OpenSSH. http://www.openssh.com/, Junio 2009.

[82] MediaWiki Developers. MediaWiki . http://www.mediawiki.org, Junio 2009.

[83] Google. Google Docs. http://docs.google.com, Junio 2009.

[84] LaTeX developers. LaTeX. http://www.latex-project.org/, Junio 2009.

[85] Sun Microsystems. Netbeans. http://www.netbeans.org/, Junio 2009.

[86] Sun Microsystems. Netbeans UML Project. http://uml.netbeans.org/, Junio 2009.

[87] Sun Microsystems. Openoffice.org Impress. http://www.openoffice.org/product/impress.html, Junio 2009.

[88] Sun Microsystems. Extension of 'Interface' definition to include class (static) methods. http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4093687, Noviembre 1997.

[89] Roel Spilker Reinier Zwitserloot. Static Methods in Interfaces. http://docs.google.com/Doc?docid=dfkwr6vq_30dtg2z9d8&hl=en, Junio 2009.

[90] Twitter. Twitter . http://www.twitter.com, Junio 2009.

[91] Google. Emulator Limitations. http://developer.android.com/guide/developing/tools/emulator.html#limitations, Junio 2009.

[92] Sun Mycrosystems. J2SE 5.0. http://java.sun.com/j2se/1.5.0/, Junio 2009.

[93] WebKit. The WebKit Open Source Project. http://webkit.org/, Junio 2009.

[94] Ryan Paul. Dream(sheep++): A developer's introduction to Google Android. http://arstechnica.com/open-source/reviews/2009/02/an-introduction-to-google-android-for-developers.ars, Junio 2009.

[95] Google. Application Fundamentals. http://developer.android.com/guide/topics/fundamentals.html, Junio 2009.

[96] Google. Intents and Intent Filters. http://developer.android.com/guide/topics/intents/intents-filters.html, Junio 2009.

[97] Google Inc. The AndroidManifest.xml File. http://developer.android.com/guide/topics/manifest/manifest-intro.html, Junio 2009.

[98] Google Inc. Activity Reference. http://developer.android.com/reference/android/app/Activity.html, Junio 2009.

[99] Google Inc. Service Reference. http://developer.android.com/reference/android/app/Service.html, Junio 2009.

[100] Google Inc. Designing a Remote Interface Using AIDL . http://developer.android.com/guide/developing/tools/aidl.html, Junio 2009.

[101] SQLite Developers. SQLite. http://www.sqlite.org/, Junio 2006.

[102] Google Inc. Security and Permissions. http://developer.android.com/guide/topics/security/security.html, Junio 2009.

[103] Google Inc. Manifest.permission Reference. http://developer.android.com/reference/android/Manifest.permission.html, Junio 2009.

[104] Damon Kohler. android-scripting. http://code.google.com/p/android-scripting/, Junio 2009.

[105] Lua Developers. The Programming Language Lua. http://www.lua.org/, Junio 2009.

[106] Python Software Foundation. Python Programming Language. http://www.python.org/, Junio 2009.

[107] Beanshell. Beanshell: Lightweight Scripting for Java. http://www.beanshell.org/, Junio 2009.

[108] The Ruby community. Ruby Programming Language. http://www.ruby-lang.org/en/, Junio 2009.

[109] The Mozilla Foundation. JavaScript. https://developer.mozilla.org/en/JavaScript, Junio 2009.

[110] Damon Kohler. Android Scripting Reference. http://android-scripting.googlecode.com/svn/trunk/android/AndroidScriptingEnvironment/doc/index.html, Junio 2009.

[111] Google Inc. Introducing Android Scripting Environment. http://google-opensource.blogspot.com/2009/06/introducing-android-scripting.html, Junio 2009.

[112] Android-NDK Google Group. Android-NDK. http://groups.google.com/group/android-ndk, Junio 2009.

[113] Khronos Group. OpenGL ES Overview. http://www.khronos.org/opengles/, Junio 2009.

[114] Google Inc. Android 1.5 SDK, Release 2. http://developer.android.com/sdk, Junio 2009.

[115] The Apache Foundation. The Apache Ant Project. http://ant.apache.org/, Mayo 2009.

[116] Google Inc. Licenses (Android Open Source Project). http://source.android.com/license, Junio 2009.

[117] Fabrice Bellard. QEMU: Open Source Processor Emulator. http://www.qemu.org/, Julio 2008.

Índice de figuras

4.1. Diagrama de la arquitectura de la aplicación 22

5.1. Entorno de desarrollo Eclipse – Manifest.xml 30
5.2. Entorno de desarrollo Eclipse – Plugin DDMS 31
5.3. Olive 32
5.4. Meld 33
5.5. Plugin Bazaar para Eclipse 33
5.6. Diagrama de la estructura de los componentes 37
5.7. Diagrama de los componentes de Regla 39
5.8. Diagrama de los clases Evento 40
5.9. Diagrama de los clases Reacción 41
5.10. Ventana Principal – Lista de reglas 43
5.11. Editor de una regla – información general 43
5.12. Editor de una regla – elegir un tipo de contexto 44
5.13. Diálogo selector de localización 45
5.14. Editor de localizaciones 45
5.15. Editor de una localización 46
5.16. Editor localización – centro fijado 46
5.17. Editor de una regla – elegir un tipo de reacción 47
5.18. Regla activada – notificación 47
5.19. Diálogo editor de contexto horario 48
5.20. Editor de una regla – Selector de tipo de evento disparador 49
5.21. Diálogo de llamada entrante – selector de contactos 49
5.22. Regla activada – realizar una llamada 50

Índice de tablas

2.1. Modelos HTC con Android 8