Universidad Nacional de la Patagonia San Juan Bosco

Licenciatura en Informática

TESINA

"Desarrollo de Aplicaciones para dispositivos Móviles sobre la plataforma Android de Google"

Alumnos: Ana, Hugo Andrés.
Gader, Ioana Noel.
Tutor: Ing. Bianchi, Gloria.
Año: 2011

Dedicado a todas las personas que nos acompañaron todos estos años en los buenos y malos momentos.

Agradecimientos

En primer lugar, agradecemos a nuestros padres, que nos han apoyado todo este tiempo para que consiguiéramos terminar esta carrera.

A nuestros sobrinos, por hacernos tan felices.

Y finalmente, agradecer a nuestro tutor de proyecto, que ha estado disponible siempre que nos han surgido dudas.

Resumen

Una aplicación móvil es software desarrollado para ejecutarse sobre un dispositivo móvil.

La investigación enfoca el análisis de las características de las aplicaciones móviles y los diferentes paradigmas de desarrollo de las mismas.

Se realiza una breve introducción a los sistemas operativos existentes y tecnologías utilizadas en los dispositivos móviles actuales, haciendo hincapié en el sistema Android de Google, plataforma para la cual se implementa una aplicación que hace uso de las características que se consideran más relevantes.

Índice

Índice de Figuras

Índice de Segmentos de Código Fuente

CAPÍTULO 1 Introducción

1.1 Organización

Esta investigación pretende brindar conocimientos básicos en el desarrollo de aplicaciones móviles, más específicamente en aplicaciones para dispositivos móviles basados en la plataforma Android de Google.

A continuación se detalla brevemente el contenido de cada uno de los Capítulos que componen esta investigación.

Capítulo 1. Introducción: se mencionan los diferentes factores que motivaron la elección de "Aplicaciones Móviles" como tema de la presente tesina. Brevemente se describe el estado del arte en el cual se encuentra el tema seleccionado y se especifican los objetivos que se pretenden cubrir en el transcurso de la investigación.

Capítulo 2. Dispositivos Móviles: define el significado de dispositivos móviles y evaluando la funcionalidad realiza una clasificación de los mismos. Debido al auge que ha experimentado la tecnología de comunicación inalámbrica y el intenso uso que hacen los dispositivos móviles de la misma se lleva a cabo una descripción y clasificación de esta tecnología.

Capítulo 3. Sistemas Operativos para Móviles: detalla las características y funcionalidades principales de los sistemas operativos para móviles y realiza una breve descripción de los más populares existentes a la fecha.

Capítulo 4. Aplicaciones Móviles: define a las aplicaciones móviles y las clasifica de acuerdo a varios criterios tales como su arquitectura, tipo de solución y tipo de código ejecutable. Referencia diferentes mecanismos de compartición de información y por último detalla las principales plataformas de desarrollo y emuladores existentes a la fecha.

Capítulo 5. Diseño de Aplicaciones Móviles: explaya diferentes consideraciones que se deben tener en cuenta en el momento de diseñar una aplicación para un dispositivo móvil.

Capítulo 6. Patrones de Diseño: realiza una definición de patrones de diseño y presenta un conjunto de ejemplos de los mismos.

Capítulo 7. Patrones Arquitectónicos: define el concepto de patrón arquitectónico y de catálogo de patrones exponiendo un ejemplo de los mismos.

Capítulo 8. Introducción a Android: describe la plataforma Android de Google especificando sus principales características y entorno de desarrollo.

Capitulo 9: Desarrollo de la aplicación “Mis Contactos”: describe el desarrollo de una aplicación en Android que utiliza las funcionalidades de la plataforma que se consideran más importantes.

Capítulo 10: Conclusiones y Trabajos Futuros: expone las conclusiones que surgieron de la investigación realizada y se sugieren diferentes áreas de investigación que pueden ser abordadas para complementar la presente tesina.

1.2 Motivación

La tecnología móvil en los últimos años ha experimentado un gran crecimiento, lo que provocó un notable incremento en su uso. Los dispositivos móviles permitieron que el acceso a la información en cualquier lugar y momento se convierta en una tarea cotidiana, creando la necesidad de desarrollar aplicaciones móviles que satisfagan los requisitos de información de los usuarios, por lo cual se deben investigar las metodologías para maximizar la eficiencia del proceso de desarrollo de software móvil.

El desarrollo de aplicaciones móviles es una disciplina que ha quedado fuera del alcance de las asignaturas del plan de estudio 2001 de la carrera Licenciatura en Informática de la Universidad Nacional de la Patagonia San Juan Bosco.

Actualmente el escenario de desarrollo de aplicaciones móviles se ha expandido considerablemente, haciendo que el mismo se imponga como tendencia en el área de desarrollo de sistemas.

Google, en el año 2007, presentó el sistema operativo para móviles Android, basado en una versión modificada de Linux y todas sus aplicaciones se desarrollan en Java. Además de cumplir su función de sistema operativo, Android brinda una gran cantidad de software adicional tal como gestores de base de datos, frameworks de aplicaciones, drivers y aplicaciones de usuarios que aprovechan al máximo las capacidades de los dispositivos móviles.

Lo expuesto anteriormente motivó a realizar la presente investigación sobre el desarrollo de aplicaciones móviles y la plataforma Android.

1.3 Campo de estudio

Se estima, según lo informado por la Unión Internacional de Telecomunicaciones (ITU), que para finales del 2010 la cifra mundial de celulares ascendería a 5000 millones, lo que significa más de dos celulares cada tres habitantes en el mundo.

Este incremento se debe a que la telefonía móvil se ha transformado en un artículo de primera necesidad en lugar de un simple bien de consumo.

Las operadoras y fabricantes de dispositivos móviles se han mantenido en un proceso de innovación constante con el fin de lograr éxito en el mercado, lo que ha dado como resultado el surgimiento de los smartphones, por ejemplo, concebidos con el objetivo de expandir los límites de la experiencia móvil.

Debido a este avance constante que han experimentado las telecomunicaciones y el desarrollo de dispositivos móviles, y el consecuente uso masivo, surge la necesidad de nuevos desarrollos y mejores aplicaciones de uso cotidiano.

El desarrollo de aplicaciones móviles es el proceso por el cual se crea software para dispositivos móviles de mano tales como asistentes digitales personales (PDA: Personal Digital Assistant), teléfonos móviles, entre otros. Estas aplicaciones pueden ser preinstaladas en los dispositivos durante su fabricación o descargadas e instaladas por los usuarios de los mismos desde diferentes centros de distribución de aplicaciones móviles.

Existe una gran variedad de plataformas de desarrollo de aplicaciones móviles. Estas plataformas normalmente son incompatibles entre sí y cada dispositivo soporta sólo una plataforma en particular. Por estas razones el diseñador de aplicaciones debe tener en cuenta diferentes factores en el momento de escoger la plataforma sobre la cual se realizará la aplicación, para maximizar su alcance y los ingresos generados por la misma. Entre los principales factores se encuentran el dispositivo objetivo sobre el cual se desea que corra la aplicación y la variedad de dispositivos que soportan la plataforma. Algunas plataformas soportan dispositivos de diversos fabricantes tales como

Java ME (Java Mobile Edition), Android, Symbian Plataform, .NET Compact Framework, BREW, entre otros, y existen otras plataformas que soportan sólo dispositivos de un único fabricante, tales como BlackBerry e iPhone OS.

Además de una adecuada selección de la plataforma el desarrollador necesita aplicar un conjunto de técnicas y patrones de diseño de aplicaciones móviles que le permitan lograr un proceso óptimo de desarrollo mejorando la calidad del software y su mantenibilidad.

En mayo del 2010 Google lanzó la versión 2.2 (Froyo) de Android, la cual posee en la actualidad una gran comunidad de desarrolladores que publican aplicaciones gratuitas y pagas. Desde el Market de aplicaciones de Android en un dispositivo móvil se pueden elegir y descargar gran variedad de aplicaciones.

1.4 Objetivos

1.4.1 Generales

- Investigar paradigmas y tecnologías de desarrollo de aplicaciones móviles.
- Desarrollar una aplicación móvil utilizando la Plataforma Android de Google.

1.4.2 Particulares

- Analizar las características de dispositivos móviles, capacidades y limitaciones de los mismos.
- Analizar las características generales de las aplicaciones móviles.
- Investigar la tecnología de arquitectura wireless para aplicaciones móviles.
- Investigar patrones de diseño para aplicaciones móviles.
- Investigar la plataforma Android para el desarrollo de aplicaciones móviles y aplicar los conocimientos adquiridos en el desarrollo de la aplicación final.

CAPÍTULO 2 Dispositivos Móviles

En la actualidad se utiliza el término dispositivo móvil para referirse a ciertos teléfonos móviles con determinadas prestaciones, sin embargo este término no está restringido únicamente al ámbito de la telefonía ya que dispositivos tales como cámaras fotográficas y GPS (Global Positioning System), entre otros, se encuentran dentro de esta categoría de dispositivos.

Determinar “qué es y qué no es” un dispositivo móvil pude ser una tarea confusa y no existe un consenso sobre el tema. Sin embargo podemos denominar como dispositivo móvil a los aparatos electrónicos que cumplen con ciertas características, tales como un tamaño que facilite su transporte, capacidad de conectarse a una red de manera permanente o intermitente, disponer de una limitada cantidad de memoria y procesamiento, poseer mecanismos de entrada y salida que permiten interactuar con el dispositivo, y el uso de baterías como fuente de energía.

2.1 Categorías de dispositivos móviles según su funcionalidad

A continuación se detalla una clasificación para los dispositivos móviles de acuerdo con su funcionalidad:

2.1.1 Dispositivo de comunicación

Su función principal es la de ofrecer una infraestructura de comunicación, principalmente telefónica. Además pueden ofrecer otros servicios tales como el envío de mensajes SMS (Short Message Service) y MMS (Multimedia Application Protocol), o acceso WAP (Wireless Application Protocol). Ejemplos de esta categoría serían los tradicionales teléfonos móviles, las BlackBerrys y los Smartphones, que amplían las funcionalidades mediante el uso de pantalla táctil, la disponibilidad de una conexión a Internet y/o la instalación y ejecución de aplicaciones.

Figura 1. Dispositivos de comunicación.

2.1.2 Dispositivo de computación

Se caracterizan por ofrecer un poder de procesamiento de datos mayor y cuentan con una pantalla y teclado más cercanos a los de una computadora de escritorio. Los servicios que prestan este tipo de dispositivos pueden igualar o superar a los ofrecidos en las computadoras de escritorio. Ejemplos de estos dispositivos son los PDA, notebook y netbook.

Figura 2. Dispositivos de computación.

2.1.3 Reproductor multimedia

Se caracterizan por ser los de menor tamaño, su principal objetivo es el de servir como reproductor de audio, video e imágenes en diferentes formatos. Ejemplos de este tipo de dispositivos son los reproductores de MP3, DVD portátiles y eBooks.

Figura 3. Reproductores multimedia.

2.1.4 Grabador multimedia

Se caracterizan por servir la función de grabación de datos en determinado formato digital, principalmente audio y video. Ejemplos de este tipo de dispositivos son las cámaras fotográficas digitales y las cámaras de video digital.

Figura 4. Grabadores multimedia.

2.1.5 Consola portátil

Se caracterizan por prestar la función de proporcionar al usuario una plataforma de juego. Ejemplos de este tipo de dispositivos son la PSP (PlayStation Portable) de Sony o la Nintendo DS de Nintendo.

Figura 5. Consolas portátiles.

2.2 Smartphone

Los teléfonos móviles tradicionales han evolucionado para dar lugar a los smartphones o teléfonos inteligentes. Éstos, además de brindar las prestaciones de los teléfonos tradicionales, agregan nuevas características que los acercan más a un computador portátil.

Los smartphones poseen mayor capacidad de proceso y almacenamiento de datos, conexión a Internet mediante WI-FI, pantalla táctil, GPS, teclado QWERTY y aplicaciones de usuario tales como navegador Web, correo electrónico, ofimáticas, entre otras, incluyendo la capacidad de instalar otras aplicaciones.

Si bien los smartphones representan un gran avance respecto a los teléfonos móviles tradicionales, su reducido tamaño conlleva limitaciones de hardware que los diferencian de las computadoras de escritorio. Entre las limitaciones se encuentran pantallas más pequeñas, menor capacidad de procesamiento, restricciones de memoria RAM (Random Access Memory) y persistente, y la necesidad de adaptar el consumo de energía a la capacidad de una pequeña batería.

Debido a estas limitaciones, los desarrolladores deben tener presentes tales características a la hora de desarrollar software para los mismos.

2.3 Tecnología wireless

La comunicación de dispositivos móviles necesita de redes de comunicaciones inalámbricas.

La comunicación inalámbrica o wireless es aquélla en la que los extremos de la comunicación (emisor/receptor) no se encuentran unidos por un medio de propagación física, sino que se utilizan la modulación de ondas electromagnéticas a través del espacio. Los dispositivos físicos sólo están presentes en los emisores y receptores de la señal, entre los cuales encontramos: antenas, notebook, PDA, teléfonos móviles, entre otros.

La tecnología wireless posee las siguientes ventajas:

- Brinda mayor comodidad permitiendo que cualquiera que tenga acceso a la red se conecte desde diferentes ubicaciones dentro de un área determinada.
- La instalación y configuración de una infraestructura de red inalámbrica es más sencilla y económica que la de una infraestructura cableada.

La tecnología inalámbrica posee las siguientes desventajas:

- Menor velocidad de transmisión que las redes cableadas a causa de las interferencias y pérdidas de señal.
- Debido a que las trasmisiones no se realizan a través un medio físico las mismas son susceptibles a ataques en contra de su seguridad. Un atacante puede ser capaz de capturar paquetes e intentar obtener las contraseñas de red para poder tener acceso a ella.

2.3.1 Wi–Fi

Wi–Fi, también llamada WLAN (Wireless Local Area Network), adopta el estándar IEEE802.11 relacionado a redes inalámbricas de área local.

El estándar IEEE802.11 ofrece diferentes tipos de WI-FI diferenciados en las bandas, velocidades y la forma en la cual se transmiten los datos:

- Los estándares IEEE 802.11b, IEEE 802.11g e IEEE 802.11n utilizan la banda de 2.4 Ghz, poseen amplia aceptación y ofrecen una velocidad de transferencia que oscila entre 11Mbps y 54Mbps.
- El estándar IEEE 802.11a (también llamado WI-FI 5), utiliza la misma tecnología que IEEE802.11g pero opera en la banda de 5.1-5.8GHz en lugar de la de 2.4 Ghz. Brinda mayor cantidad de canales con menos interferencias. Por poseer una frecuencia mayor este estándar tiene un alcance aproximadamente un 10% menor que el IEEE802.11g.
- El estándar IEEE 802.11n trabaja a 2.4 GHz y utiliza diferentes técnicas para lograr velocidades mayores al resto de los estándares 802.11.

2.3.2 Bluetooth

Es un estándar de comunicación incluido en las WPAN (Redes Inalámbricas de Área Personal), que permite la transmisión tanto de voz como de datos, interconectando diversos tipos de dispositivos por medio de un

enlace de radiofrecuencia. Esta tecnología define un canal de comunicación de un máximo de 720 kbps con rango óptimo de 10 metros. La frecuencia de radio con la que trabaja está en el rango de 2,4 a 2,48 GHz con las siguientes características:

- Suprime cableados entre los dispositivos estacionarios e inalámbricos.
- Transmite voz y datos.
- Permite la creación de redes sin infraestructura fija.
- Sincroniza todos los dispositivos.

Los dispositivos que con mayor frecuencia utilizan esta tecnología pertenecen a sectores de las telecomunicaciones y la informática personal, tales como PDA, teléfonos móviles, computadoras portátiles y personales, impresoras o cámaras digitales.

El protocolo Bluetooth fue diseñado especialmente para dispositivos de bajo consumo, con una cobertura baja y basada en transceptores de bajo coste.

2.3.3 Infrarrojo

Las redes de luz infrarroja están limitadas por el espacio. Se utilizan generalmente para estaciones que se encuentran en el mismo cuarto o piso.

Se utilizan leds de luz infrarroja para la transmisión y recepción de datos. La transmisión de luz se codifica y decodifica en el envío y recepción en un protocolo de red existente. Esta tecnología se ha mejorado utilizando un transreceptor que difunde el haz en todo el cuarto y es captado mediante otros transreceptores.

2.3.4 GSM

El Sistema Global para las Comunicaciones Móviles (GSM: Global System for Mobile) es un sistema estándar completamente definido para la comunicación mediante teléfonos móviles que incorporan tecnología digital. Por ser digital cualquier cliente de GSM puede conectarse a través de su teléfono con su computador y puede enviar y recibir mensajes por e-mail o fax, navegar por Internet, tener acceso seguro a la red informática de una compañía (LAN/Intranet), así como utilizar otras funciones digitales de transmisión de datos, incluyendo el Servicio de Mensajes Cortos (SMS) o mensajes de texto.

GSM se considera, por su velocidad de transmisión y otras características, un estándar de segunda generación (2G).

GSM presenta varias características importantes tales como llamadas en espera, identificación de la llamada, retención de llamada, llamadas en conferencia y velocidades de datos de 2400 hasta 9600 bps.

2.3.5 Tecnología GPRS

GPRS es la sigla de General Packet Radio Services (Servicios Generales de Paquetes por Radio). A menudo se describe como "2,5 G", es decir, una tecnología entre la segunda (2G) y la tercera (3G) generación de tecnología móvil digital. Se transmite a través de redes de telefonía móvil y envía datos a una velocidad de hasta 114 Kbps. El usuario puede utilizar el teléfono móvil para navegar por Internet, enviar y recibir correo, y descargar datos. Permite realizar videoconferencias y utilizar mensajes instantáneos para comunicarse con sus contactos.

2.3.6 Tecnología 3G

Al igual que GPRS, la tecnología 3G (tecnología inalámbrica de tercera generación) es un servicio de comunicaciones inalámbricas que permite estar conectado permanentemente a Internet a través del teléfono móvil, el PDA, el Tablet PC o una computadora portátil. La tecnología 3G promete una mejor calidad y fiabilidad, una mayor velocidad de transmisión de datos y un ancho de banda superior, lo cual permite la ejecución de gran variedad de aplicaciones, incluidas las aplicaciones multimedia. Con velocidades de datos de hasta 384 Kbps, es casi siete veces más rápida que una conexión telefónica estándar.

2.3.7 Tecnología WAP

WAP (Wireless Application Protocol) es un protocolo de comunicación y un medio para la ejecución de aplicaciones que le permiten a un dispositivo móvil acceder a la Web. Este protocolo está diseñado para trabajar en ambientes heterogéneos de redes inalámbricas de transmisión de datos, diversos dispositivos móviles y diferentes sistemas operativos.

Operacionalmente, WAP optimiza el uso de recursos en el dispositivo móvil y delega las tareas más demandantes a otros servidores dentro de la red.

Uno de los servidores de este tipo más importantes es el conocido como WAP Gateway. La tarea principal de un Gateway es realizar los procesos de traducción entre contenido WAP y contenido HTTP (HyperText Transport Protocol). De esta manera un dispositivo móvil puede usar servidores Web para recuperar información.

CAPÍTULO 3 Sistemas Operativos para Móviles

Un Sistema Operativo (SO) es un software que actúa como interfaz entre los dispositivos de hardware y los programas usados por el usuario para manejar un computador. Es responsable de gestionar y coordinar las actividades, llevar a cabo el intercambio de recursos y actuar como base para las aplicaciones que se ejecutan en la máquina.

Un SO móvil es un sistema operativo que controla un dispositivo móvil al igual que las computadoras utilizan Windows o Linux, entre otros. Sin embargo, los SO móviles son bastante más simples y están más orientados a la conectividad inalámbrica, los formatos multimedia para móviles y las diferentes maneras de introducir información en ellos.

Un SO móvil necesita ser fiable y tener una gran estabilidad ya que incidencias habituales y toleradas en computadoras personales, tales como reinicios o caídas, no tienen cabida en un dispositivo móvil. Además, debe adaptarse adecuadamente a las limitaciones de memoria y procesamiento de datos, proporcionando una ejecución exacta y rápida al usuario.

Es posible incluso que un dispositivo móvil deba estar funcionando ininterrumpidamente durante semanas e incluso meses antes de ser apagado y reiniciado, a diferencia de lo que ocurre con una computadora personal. Por esto, los sistemas deben estar perfectamente testeados y libres de errores antes de incorporarse definitivamente a la línea de producción, debido a que las actualizaciones e incluso reinstalar mejores versiones del sistema para cubrir fallos o deficiencias, son más limitadas en un dispositivo móvil.

Otro aspecto delicado que se debe tener en cuenta en el momento del desarrollo de un SO es el consumo de energía. Es necesario que el SO haga un uso lo más racional y provechoso posible de la batería, ya que ésta es limitada y el usuario siempre exige una mayor autonomía.

3.1 Symbian

Para competir con los SO móviles de Palm y Windows, se creó una alianza entre las empresas Nokia, Motorota y Ericsson, entre otras, para crear

un nuevo SO compartido que estuviera adaptado a los teléfonos móviles del momento. Como resultado de esta alianza surge el SO Symbian en 1998.

Fue diseñado para residir en un espacio muy pequeño, hacer un uso dinámico de escasos recursos de memoria, administrar eficientemente la energía y soportar en tiempo real los protocolos de comunicación y telefonía, además de ser más “gentil” con el usuario y tolerante a fallas, en comparación con un SO de PC.

Este SO incluye un microkernel, controladores, middleware y una considerable pila de protocolos de comunicación e interfaces de usuario muy básicas.

El microkernel se encarga de la ejecución de los procesos, manejar sus prioridades y acceso a recursos. Sólo una mínima porción del sistema tiene privilegios de kernel, el resto se ejecuta en modo usuario.

Los desarrolladores que obtienen la licencia correspondiente para trabajar con Symbian implementan sus propias interfaces de usuario y conjuntos de aplicaciones según las necesidades de sus propios dispositivos. Esto permitió a Symbian posicionarse como un SO muy flexible, que tenía en cuenta los requisitos de la mayoría de los dispositivos fabricados y, al mismo tiempo, permitía un alto grado de diferenciación.

Técnicamente, Symbian es una colección compacta de código ejecutable y varios archivos, la mayoría de ellos bibliotecas vinculadas dinámicamente (DLL: Dynamic-link library) y otros datos requeridos, incluyendo archivos de configuración, de imágenes y de tipografía, entre otros recursos residentes. Symbian se almacena, generalmente, en un circuito flash dentro del dispositivo móvil.

Gracias a este tipo de tecnología, se puede conservar información aún si el sistema no posee carga eléctrica en la batería, además de que le es factible reprogramarse.

Symbian posee cinco tipos de ediciones o series, según las características del dispositivo móvil. La principal diferencia entre ediciones no radica tanto en el núcleo del SO sino en la interfaz gráfica utilizada:

- **Serie60**. Es la más popular de todas. Los dispositivos con Serie60 tienen una pantalla pequeña y un teclado del tipo 0-9#. Serie60 es el

núcleo de casi todos los modelos de smartphones de Nokia. También lo utilizan fabricantes como Siemens, Samsung y Panasonic.

- **Serie80**. Orientada a dispositivos con pantalla táctil. Permite multitarea pudiendo tener varias aplicaciones abiertas simultáneamente.
- **Serie90**. Muy similar a la edición Serie80, sólo que estos dispositivos tienen una pantalla más grande y llevan incorporados sensores táctiles más eficientes. Utilizan teclados virtuales, reconocimiento de trazos o teclados acoplables mediante, por ejemplo, Bluetooth.
- **UIQ**. La interfaz de esta edición de Symbian se encuentra muy influenciada por Palm OS. Implementa una especie de multitarea virtual, dando al usuario la falsa sensación de poder realizar varias acciones simultáneas; suele tener un alto coste computacional e influye negativamente en el tiempo de respuesta apreciado por el usuario. Es utilizado en algunos modelos de Sony Ericcson y Motorola.
- **MOAP**. Esta edición se da únicamente en Japón, principalmente en el fabricante FOMA.

Las aplicaciones para Symbian se pueden desarrollar en los lenguajes habituales tales como Java, C++, Visual Basic, entre otros. Esto ha permitido que actualmente existan cientos de miles de aplicaciones disponibles para Symbian.

La compañía Nokia, en el 2008, adquirió todas las acciones de Symbian formando la Fundación Symbian y anunciando su intención de liberar el SO como software libre, en un intento por competir con sistemas libres como Android de Google.

3.2 BlackBerry OS

El surgimiento de este SO se remonta a la aparición de los primeros handheld, en 1999. Estos dispositivos permiten el acceso a correo electrónico, navegación Web y sincronización con programas tales como Microsoft Exchange o Lotus Notes, aparte de brindar las funciones usuales de un teléfono móvil.

Este SO es desarrollado actualmente por RIM (Research in Motion) que también ofrece servicios de correo electrónico a otros dispositivos a través de BlackBerry Connect.

En un SO multitarea enfocado a un uso profesional y empresarial como gestor de correo electrónico y agenda. La mayoría de estos dispositivos cuentan con teclado QWERTY completo.

A partir de la cuarta versión, se puede sincronizar el dispositivo con el correo electrónico, el calendario, tareas, notas y contactos de Microsoft Exchange Server, además de ser compatible también con Lotus Notes y Novell GroupWise.

BlackBerry Enterprise Server (BES) proporciona el acceso y organización del email a grandes compañías identificando a cada usuario con un único BlackBerry PIN. Los usuarios más pequeños cuentan con el software BlackBerry Internet Service, programa más sencillo que proporciona acceso a Internet y a correo POP3 / IMAP / Outlook Web Access sin tener que usar BES.

Al igual que en el SO Symbian, desarrolladores independientes también pueden crear programas para BlackBerry en Java, pero en el caso de querer tener acceso a ciertas funcionalidades restringidas necesitan firma digital para ser asociados a una cuenta de desarrollador de RIM.

Este SO es software propietario, pero carece de virus o troyanos, aspecto muy importante para el tipo de clientes para el cual están pensadas las BlackBerrys.

BlackBerry OS está quedando rápidamente obsoleto frente a otros SO tales como iPhone OS, Android, WebOS y MeeGo.

3.3 Windows Phone

Anteriormente llamado Windows Mobile, es un SO móvil compacto desarrollado por Microsoft, y diseñado para su uso en teléfonos inteligentes (smartphones) y otros dispositivos móviles.

Microsoft apuntó a que el SO fuera lo suficientemente flexible y adaptable para poder ser utilizado en un amplio abanico de dispositivos, cuyas únicas características comunes son su reducido tamaño y recursos limitados.

Sus principales características son:

- Es un sistema modular, por lo cual cada fabricante pueda seleccionar aquellas partes que le benefician más para su dispositivo. Algunos de estos módulos se ofrecen a los desarrolladores o fabricantes a través del propio código fuente por lo cual éstos pueden adaptarlos a sus necesidades.
- Da soporte a una considerable gama de recursos hardware: teclado, cámara, pantalla táctil, etc.
- Tiene un tamaño en memoria relativamente pequeño y bajo coste computacional.
- Es capaz de trabajar con distintas familias de procesadores de 32 bits.
- Permite interactuar con otros dispositivos móviles.

Fue diseñado para que su apariencia fuera similar a las versiones de Windows de escritorio. Surgió a partir de una ramificación del SO Pocket PC el cual está basado en Windows CE, cuenta con un conjunto de aplicaciones básicas utilizando las API (Application Programming Interface) de Microsoft Windows y fue pensado para equipos móviles con capacidades limitadas.

La mayoría de los equipos Windows Mobile vienen con un estilete digital, que se utiliza para introducir comandos pulsando en la pantalla (navegación tipo stylus).

La última versión de este SO fue anunciada en febrero del 2010 bajo el nombre de Windows Phone 7.

Existe una gran oferta de software de terceros disponible para Windows Mobile, además este SO trae incorporadas aplicaciones como ActiveSync, que tiene la capacidad de convertir archivos de versiones de escritorio a archivos compatibles con Pocket PC y adaptaciones de las aplicaciones de Microsoft más utilizadas para escritorio, tales como Pocket Word, Pocket Excel, Pocket Power Point, Windows Media Player for Mobile y Outlook Mobile.

3.4 iOS

Este SO anteriormente se denominaba iPhone OS y fue desarrollado por Apple Inc. para los dispositivos móviles iPod touch, iPhone e iPad. Es una variante del Mach kernel de Mac OS X, que está basado en Unix.

iOS tiene cuatro capas de abstracción: la capa del núcleo del SO, la capa de "Servicios Principales", la capa de "Medios de comunicación" y la capa de "Cocoa Touch". Cocoa Touch es un framework de desarrollo de aplicaciones para iPhone o iPod Touch.

Incluye varias aplicaciones software de serie, identificadas en la interfaz por su función: Calculadora, Calendario, Cámara, Fotos, iPod, iTunes, Mail, Mapas (Google Maps), Nike + iPod (exclusivo de iPhone 3GS e iPod touch 3ª generación), SMS, Reloj, Safari, Teléfono (exclusivo de los iPhones), Tiempo, Youtube.

Sobre este SO no se pueden instalar aplicaciones sin la supervisión de Apple; sin embargo, existen aplicaciones hechas por terceros que pueden instalarse extraoficialmente mediante jailbreak.

Para el desarrollo de aplicaciones se utiliza un IDE (Integrated Development Environment) muy completo llamado Xcode en el que se puede editar código fuente, acceder a un vasto volumen de documentación, y hasta a un depurador gráfico. El lenguaje en el cual se programa es un dialecto especial de C llamado Objective-C. Existe una herramienta, Aspen, mediante la cual se pueden realizar emulaciones de las aplicaciones desarrolladas.

El SDK (Software Development Kit) puede descargarse gratis, pero para publicar el software es necesario registrarse en el Programa de Desarrollo del iPhone, previo pago y aprobación por parte de Apple. Durante el proceso, se entregan al desarrollador claves firmadas que permiten subir una aplicación a la tienda de aplicaciones de Apple.

3.5 Palm OS

Palm OS es un SO desarrollado por PalmSource, Inc. para computadores de mano (PDA).

Con el paso del tiempo se han desarrollado diferentes versiones de este SO, cada una de ellas agrega nuevas capacidades tales como el apoyo para color, puertos de expansión múltiples, nuevos procesadores, acceso a memorias SD (Secure Digital), bibliotecas de telefonía, seguridad y otras prestaciones.

La versión 5 fue la primera que soportó los dispositivos ARM (Advanced RISC Machine). Las aplicaciones Palm se ejecutan en un entorno emulado denominado el Entorno de Compatibilidad de Aplicaciones Palm, disminuyendo la velocidad pero permitiendo gran compatibilidad con programas antiguos. El nuevo software puede aprovechar los procesadores de ARM con ARMlets, pequeñas unidades de código ARM.

La versión 6 completaría la migración a aparatos con ARM, y permitiría las aplicaciones nativas ARM junto con apoyo multimedia mejorado. Actualmente no existen equipos que usen Palm OS 6. No está muy claro el futuro de esta versión de Palm OS, derivado de la compra de PalmSource por la compañía japonesa Access Co, la cual pretende la creación de nuevos dispositivos PDA con un kernel basado en Linux.

Existen muchas aplicaciones interesantes para el SO de Palm OS que se pueden añadir, incluyendo software libre, tales como el lector de documentos Plucker o la base de datos Pilot-DB.

3.6 Android

Android es un SO orientado a dispositivos móviles desarrollado por Android Inc., compañía que fue adquirida por Google.

Esta plataforma permite el desarrollo de aplicaciones de terceros a través del SDK, proporcionado por el mismo Google, y mediante el lenguaje de programación Java. Una alternativa es el uso del NDK (Native Development Kit) de Google para emplear el lenguaje de programación C.

El código fuente de Android está disponible bajo diversas licencias de software libre y código abierto, destacándose la versión 2 de la licencia Apache.

Este SO se expondrá con mayor detalle en el Capítulo 8 dedicado a al mismo.

CAPÍTULO 4 Aplicaciones Móviles

El avance que han experimentado los dispositivos móviles y las telecomunicaciones ha hecho posible el desarrollo de dispositivos capaces de reconocer y transmitir sonidos e imágenes instantáneamente, entre otras funciones de avanzada, convirtiéndolos en artículos de uso masivo.

El incremento en el uso de dispositivos móviles trae aparejada la necesidad de aplicaciones de uso cotidiano de la sociedad actual, de esta manera los usuarios finales de los móviles podrán sacar mejor provecho de la tecnología que tienen en sus manos.

Las aplicaciones móviles deben estar a la altura de los nuevos equipos y permitir que los usuarios puedan interactuar fácilmente con el dispositivo móvil.

4.1 Clasificación de arquitecturas móviles

Las aplicaciones móviles y sus arquitecturas pueden ser clasificadas de diversas maneras. Entre las más importantes se encuentran su aplicabilidad, arquitectura, funcionalidad y rango.

4.1.1 Aplicabilidad

Una aplicación móvil puede ser dependiente o independiente de una plataforma determinada. En el primer caso la aplicabilidad se denomina específica y en el segundo caso genérica. En otras palabras, una aplicación móvil puede ser soportada por sólo un dispositivo móvil específico o puede ser independiente del tipo de plataforma móvil.

Dependiendo del grupo de usuarios, es posible diseñarla para un único dispositivo específico, en el caso de tratarse de un grupo de usuarios que poseen el mismo dispositivo móvil. Para el caso de un grupo más amplio y heterogéneo de dispositivos es necesario diseñar una aplicación capaz de funcionar en múltiples plataformas.

En el desarrollo de una aplicación dependiente de la plataforma, los desarrolladores podrán sacar el máximo provecho del dispositivo haciendo uso de su API, con la desventaja de que todos los usuarios que no disponen de dicha plataforma no podrán correr la misma.

Si se desea desarrollar una aplicación que será comercializada para un público general, es necesario que la aplicación sea soportada por la mayor cantidad de plataformas posibles, ampliando de esta forma el mercado de la misma. En este caso es deseable que la aplicación corra en todas las plataformas, o al menos en las más importantes. Otra posibilidad es la de desarrollar múltiples versiones de la misma aplicación, cada una de las cuales esté destinada a un tipo específico de dispositivos, por lo cual el usuario deberá decidir qué versión debe instalar. Es posible que la mejor solución sea el desarrollo de una aplicación genérica en lugar de una óptima, sin embargo es cuestionable que una aplicación genérica pueda tener todas las características de una aplicación dependiente de la plataforma.

Las aplicaciones J2ME y Windows CE pueden hacer uso tanto de funciones genéricas como utilizar funciones específicas de las API de un dispositivo particular, con el objetivo de mejorar la funcionalidad.

4.1.2 Arquitectura

Otro criterio de clasificación de las aplicaciones móviles está dado por su arquitectura. Se entiende por arquitectura de un sistema a su estructura en términos de componentes. Un modelo arquitectónico de un sistema trata sobre las partes que lo componen y la manera en que ellas interactúan para lograr realizar la tarea requerida. El objetivo de los modelos arquitectónicos es garantizar que la estructura desarrollada satisfaga los requerimientos actuales y futuros.

La primer arquitectura de sistemas distribuidos que surgió fue la llamada cliente/servidor, posteriormente surgieron arquitecturas tales como Web based y peer to peer, entre otros modelos.

4.1.2.1 Cliente/Servidor

Esta arquitectura es una de las más utilizadas, está determinada por la forma en la que se comunican las aplicaciones. En general el cliente hace solicitudes de servicios o información al servidor, éste realiza las tareas necesarias para satisfacer el requerimiento y responde al cliente con la información solicitada.

Ejemplos de esta arquitectura son servicios de transferencia de archivos mediante un cliente y servidor de FTP (File Transfer Protocol), consultas Web desde un browser a un servidor Web mediante el uso de algún protocolo como puede ser HTTP (HyperText Transfer Protocol) y aplicaciones que acceden a servidores de bases de datos, entre otros.

Una desventaja de esta arquitectura es la existencia de un único servidor que ante múltiples requerimientos de clientes puede dar lugar a un cuello de botella o un punto de falla del sistema.

4.1.2.2 Peer to peer

En esta arquitectura todos los puntos de la misma desempeñan tareas semejantes, intercambiando información como iguales, sin distinción de los roles cliente o servidor.

La tecnología peer to peer permite replicar, compartir y buscar información, se adapta a nuevas fuentes de información y permite la construcción de servicios complejos basados en la combinación de servicios más simples.

La replicación de la información le brinda al sistema una mayor robustez ante la falla de algunos de sus puntos debido a que existe otro par que puede desarrollar sus tareas. El sistema aumenta su desempeño a medida que se le incorporan más puntos. Otras de sus ventajas es el mejor uso del ancho de banda, procesador, almacenamiento, memoria y otros recursos de los dispositivos.

4.1.3 Funcionalidad

La funcionalidad y la arquitectura son conceptos muy relacionados. La funcionalidad de las aplicaciones móviles se puede clasificar en tres diferentes categorías: sistemas de mensajes, aplicaciones Web y aplicaciones similares a las de escritorio.

SMS es un sistema de mensajes mediante el cual los usuarios intercambian mensajes de sólo texto, el sistema MMS permite el intercambio tanto de texto como de diversos formatos de audio, imágenes, animaciones y video.

Es posible desarrollar un sin fin de sistemas de mensajes basados en SMS, entre ellos reservas, compras y consultas de servicios.

WAP puede ser pensado como una Web simplificada. Su interfaz es simple y rudimentaria, está basada en textos, text boxes, option buttons e hipervínculos. Las páginas WAP pueden ser estáticas o pueden ser una capa de presentación de un sistema multicapas y, por lo tanto, dinámicas.

Por otro lado, J2ME y Windows Phone brindan mejores interfaces de usuario, similares a las que ofrecen las aplicaciones de escritorio. Su apariencia, capacidades y métodos de desarrollo son prácticamente idénticos a los de las aplicaciones no móviles. Las aplicaciones J2ME soportan eventos de teclado, text boxes, labels, radio buttons, lists, calendar controls, barras de progreso, imágenes, entre otros. Todos estos controles están optimizados para las pequeñas pantallas que suelen tener los móviles. J2ME soporta procesamiento multihilos, cualidad utilizada frecuentemente para el desarrollo de juegos para móviles.

Las herramientas de desarrollo como Forte de Sun, JBuilder de Borland y Eclipse representan poderosas plataformas de desarrollo para aplicaciones J2ME empresariales y personales. Algunas herramientas de desarrollo también ofrecen sus propios simuladores de dispositivos móviles.

Windows Phone también ofrece controles que brindan a sus aplicaciones apariencia similar a la de las aplicaciones de escritorio, para ello hace uso de una versión del .NET Framework llamada .NET Compac Framework. Sus aplicaciones pueden desarrollarse en lenguajes tales como C# y Visual Basic .Net utilizando entornos de desarrollo como Visual Studio 2003, Visual Studio 2005, Visual Studio 2008.

4.1.4 Rango

SMS es la tecnología móvil presente en la mayoría de los dispositivos móviles, por lo cual una aplicación basada en un sistema de mensajes SMS estaría disponible para casi la totalidad de los dispositivos móviles. Algo similar ocurre con los estándares WAP en sus diferentes versiones, los cuales se han convertido en características estándar de los teléfonos móviles modernos. La tecnología J2ME también se encuentra ampliamente difundida en los móviles

de hoy en día, incluso los fabricantes incorporan en sus dispositivos móviles implementaciones en hardware de la máquina virtual de J2ME. La tecnología Windows Mobile no se encuentra tan difundida como J2ME, a pesar del esfuerzo de Microsoft en el desarrollo de nuevas versiones y service pack para corregir bugs y agregar nuevas funcionalidades.

4.2 Tipos de soluciones

En el momento de desarrollar una solución se debe considerar qué tipo de solución es la más conveniente para la necesidad dada. Esto dependerá de que se requiera conexión o sincronización con un servidor central, de la diversidad o no de equipos de los usuarios del sistema y de la capacidad de interacción con el equipo que se precise.

Algunos de los posibles tipos de soluciones son: soluciones Stand-alone, soluciones Online o soluciones conocidas como Smart Clients.

4.2.1 Stand-alone

Este tipo de solución se desarrolla para ser instalada y ejecutada sobre equipos específicos y funciona en forma desconectada de Internet o de un servidor central.

Para el desarrollo de una aplicación de este tipo se debe generar un paquete ejecutable que fuese compatible con el SO que posee el dispositivo móvil sobre el cual se instalará la misma.

Debido a que diferentes dispositivos utilizan distintos SO es muy probable que las aplicaciones desarrolladas para un sistema no sean capaces de ejecutarse en otro. Un ejemplo de esto sería intentar correr una aplicación Palm OS en un SO Android.

Este tipo de solución brinda a las aplicaciones las siguientes ventajas:

- Mayor velocidad de ejecución.
- Mejor rendimiento, dado que se desarrollan para un determinado dispositivo haciendo mejor uso de las características del mismo, lates como manejo de memoria, herramientas y controles.
- Soporte para la sincronización con dispositivos de escritorio.

- Ejecución sin necesidad de disponer de una conexión, ya que la aplicación se encuentra instalada en el dispositivo y no es un requisito que esté conectada a otro.

Dentro de las desventajas de este tipo de aplicaciones encontramos las siguientes:

- Debido a la incompatibilidad entre los diferentes SO, es necesario el desarrollo de diferentes versiones de una misma aplicación destinada a cada SO sobre el cual se desea instalar.
- Las aplicaciones Stand-alone deben ser instaladas manualmente en cada dispositivo sobre el cual deba ejecutarse la aplicación.
- Las capacidades de almacenamiento se encuentran limitadas por el almacenamiento brindado por el dispositivo móvil.
- Como consecuencia de la falta de conexión de las aplicaciones, estas no pueden acceder a bases de datos u otras fuentes de información remota.

4.2.2 Soluciones Online

Una aplicación conectada u Online, es una aplicación que se vale de páginas Web o WAP como interfaz a través de Internet para lograr su función. La mayor funcionalidad de estas aplicaciones es ejecutada por un servidor remoto.

Estas aplicaciones logran compatibilidad con un mayor número de dispositivos y SO que las soluciones Stand-alone, ya que se basan en lenguajes comprendidos por varios tipos de dispositivos, como por ejemplo el lenguaje HTML (HyperText Markup Language). Otros lenguajes difundidos en dispositivos móviles son xHTML (eXtensible Hypertext Markup Language), WML (Wireless Markup Language), cHTML (Compact HTML).

Ventajas de las aplicaciones Online:

- Mayor rango, debido al uso de lenguajes como HTML que le permiten a este tipo de aplicaciones tener una compatibilidad mayor con diferentes modelos de dispositivos y SO.

- A diferencia de las aplicaciones Stand-alone, estas no necesitan ser instaladas en los dispositivos y pueden ser ejecutadas en SO en los cuales el fabricante impide la instalación de aplicaciones.
- Se puede acceder a la potencia de cómputo de los servidores, permitiendo la ejecución de procesamientos complejos desde dispositivos con capacidad de cómputo reducidas.
- Las capacidades de almacenamiento en los servidores son mayores que la de los dispositivos móviles.

Las desventajas son:

- Al no ser diseñadas para un dispositivo específico, estas aplicaciones no pueden sacar provecho de características propias del mismo.
- Los controles que se utilizan en la interfaz están restringidos a aquéllos que brinda el lenguaje utilizado.
- Utilizan conexión permanente para poder ejecutarse, restringiendo el uso de la aplicación a momentos y lugares en los cuales haya disponibilidad de una conexión.
- La información no se encuentra almacenada en el dispositivo, por lo cual acceder a ella implica una comunicación con el servidor, lo cual hace más lento el acceso a la misma.

4.2.3 Soluciones Smart Client

Una aplicación Smart Client (cliente inteligente) combina las ventajas de las soluciones Stand-alone y Online. Estas aplicaciones se distribuyen e instalan en los equipos, como las aplicaciones Stand-alone, y utilizan una conexión para comunicarse e interactuar con un servidor, como las soluciones Online.

Este tipo de aplicaciones tiene la capacidad de funcionar aún cuando el equipo pierde conexión con el servidor, guardando la información en buffers de información que luego serán sincronizados cuando se reestablezca la conexión.

Entre las ventajas de las soluciones Smart Client, se destacan:

- Reúnen lo mejor del mundo conectado y del desconectado.

- Permiten consultar grandes volúmenes de información almacenada en los servidores y aprovechar su potencia de procesamiento.
- Utilizan las funciones de bajo nivel de los equipos.
- Permiten seguir trabajando cuando el equipo se desconecta.

Algunas desventajas:

- El desarrollo de aplicaciones que permitan trabajar tanto de manera Online como Offline, sin que esta diferencia sea percibida por el usuario, es una tarea compleja.
- Se debe diseñar el cliente de manera específica para el dispositivo y SO sobre el cual se ejecutará, lo cual puede provocar que la aplicación posea diferentes versiones, una para cada tipo de dispositivo o versión del SO.
- Se debe distribuir e instalar el cliente en todos los equipos.

4.3 Tipo de código ejecutable

Dependiendo de la manera en la que está generado el código, es posible clasificar las aplicaciones que se ejecutan directamente en un dispositivo en dos clases: en código nativo y en código manejado.

4.3.1 Código nativo

Una aplicación en código nativo es aquélla en la que su ejecutable se encuentra expresado en un lenguaje entendible tanto por el SO, como por el procesador del dispositivo. Este lenguaje es conocido como lenguaje Assembler o ensamblador.

El hecho de que una aplicación se encuentre expresada en lenguaje ensamblador no significa que haya sido desarrollada en dicho lenguaje, la misma puede haber sido desarrollada en algún lenguaje de alto nivel y luego traducida a ensamblador mediante un compilador.

Las ventajas de desarrollar bajo código nativo son:

- El código desarrollado a medida mediante lenguaje ensamblador, es más compacto y se ejecuta con mayor velocidad que el desarrollado en un lenguaje de alto nivel y traducido por un compilador.
- Es posible hacer uso de todas las capacidades del dispositivo.

- No se requiere la instalación de ningún agregado para que la aplicación pueda ser entendida y ejecutada por el dispositivo.

  Las desventajas son:

- Es necesario recompilar el proyecto y generar ejecutables diferentes para cada SO o hardware en los cuales se ejecutarán las aplicaciones.
- Las diferencias de hardware pueden dar lugar a que algunas funciones no existan o se comporten de manera diferente en cada dispositivo.
- La libertad de acceso directo a memoria, junto con la falta de controles, pueden ocasionar problemas.

4.3.2 Código manejado

Este tipo de código surgió para dar solución a los inconvenientes que presenta el código nativo al tener que desarrollar código ensamblador específico para cada plataforma en la cual se ejecute una aplicación.

Al compilar la aplicación se genera un código que no es entendible por el hardware o SO, sino que éste es interpretado por una aplicación intermedia, denominada Máquina Virtual, que se encarga de convertir en tiempo real dicho código al lenguaje nativo del sistema y hardware en el que se encuentra.

Las ventajas del código manejado son:

- Una aplicación puede ser portable a diversos SO y dispositivos que tengan instalada la máquina virtual, sin la necesidad de recompilar. De esta forma es necesario generar sólo un paquete de instalación para todos los equipos.
- La máquina virtual administra automáticamente los recursos del dispositivo, evitando usos indebidos, como puede ser el acceso indebido a áreas de memoria o a algún otro recurso protegido.

  Entre las desventajas se puede mencionar:

- Se accede solamente a las funcionalidades brindadas por la máquina virtual, limitándose la posibilidad de acceder a características específicas de un dispositivo.

Dentro de los lenguajes de código manejado más utilizados para móviles se encuentran J2ME, PalmOS y .NET Compact Framework para equipos con distintas versiones de Windows Phone.

4.4 Administración de la información en dispositivos móviles

El objetivo principal de la mayor parte de las aplicaciones es la de almacenar datos para permitir su acceso en el momento en que se necesite.

El manejo e intercambio de la información en dispositivos móviles se puede realizar a través de una arquitectura de documentos compartidos o a través de una de base de datos.

Normalmente la forma de actualización de los documentos se realiza mediante la copia total del archivo, sin embargo en determinadas situaciones es necesario que la aplicación conozca la estructura interna del mismo para poder realizar la administración y actualización de los datos. Una aplicación que hace uso de una base de datos puede realizar las actualizaciones con un nivel de detalle mayor, ya que la aplicación encargada de administrar la base de datos conoce la estructura interna de la misma.

4.4.1 Arquitectura de documentos compartidos

La compartición de documentos se puede llevar a cabo de las siguientes formas:

- SCDS (Single Copy Document Sharing). En este mecanismo de compartición existe una única copia del documento, la cual es accedida por todos los usuarios. Cualquier cambio en el mismo es absoluto, inclusive su destrucción, y será percibido por el resto de los usuarios. Debido a la existencia de una única copia del documento, sólo un usuario puede acceder al mismo para modificarlo, lo cual reduce la productividad del sistema y representa un punto de falla si el usuario por descuido no deja disponible el archivo luego de utilizarlo. Puede existir acceso simultáneo a un archivo sólo para lectura.
- MCDS (Multiple Copy Document Sharing). En este mecanismo de compartición de archivos existe una copia del documento por cada usuario. La existencia de múltiples copias del documento introduce una dificultad a la hora de mantener la información actualizada y consistente. Si un usuario actualiza su copia del documento, el resto de las copias quedan desactualizadas y deben ser revalidadas. Si dos usuarios

intentan modificar su copia simultáneamente, ambas copias resultarán incorrectas y deberán ser sincronizadas o reintegradas.

Un mecanismo de compartición de documentos debe permitir que la información contenida en los documentos sea compartida, se encuentre disponible bajo demanda y encargarse de mantener la consistencia de la misma ante operaciones realizadas por diferentes usuarios simultáneamente.

Compartiendo los documentos mediante una única copia (SCDS) se evita el acceso múltiple a los documentos, limitando la productividad del sistema. En cambio, haciendo uso de múltiples copias (MCDS) existe el riesgo de que las actualizaciones realizadas sobre un archivo no sean propagadas oportunamente a sus copias. Sin embargo, esta situación puede ser resuelta mediante mecanismos que permitan sincronizar las múltiples copias del documento.

Un sistema de administración de archivos debe permitir el almacenamiento de documentos originales y copias de los mismos, ser capaz de propagar los cambios a cada copia en el sistema y resolver conflictos de versiones de manera automática o mediante la intervención del usuario en los casos en los cuales no sea posible resolverlos automáticamente.

La disponibilidad de los documentos y la consistencia de los mismos se logran a través de una conexión constante entre el servidor y las diferentes copias. De existir dicha conexión, el sistema puede proveer una semántica de actualización similar a la de una sola copia.

En un entorno móvil, los recursos de hardware y espacio de almacenamiento son limitados y no se puede garantizar que la conexión entre el dispositivo y el sistema de administración de archivos sea continua, por lo cual el mecanismo de compartición de archivos debe ser modificado para poder mantener la consistencia a pesar de estas limitaciones.

4.4.1.1 Características deseables en una arquitectura para compartir documentos en ambientes móviles

Clientes que soporten múltiples plataformas

Para que la arquitectura soporte múltiples plataformas debe ser capaz de identificar el dispositivo móvil, recursos disponibles, ancho de banda,

capacidad de almacenamiento, tipo de pantalla, etc. y en función de esto modificar el contenido del documento.

Capacidad de trabajo Offline

Esta capacidad se logra almacenando los archivos a los cuales se desea acceder en cache, ya sea en forma automática como a elección por parte del usuario. Se deben copiar tantos archivos como sean necesarios, teniendo en cuenta las limitaciones de almacenamiento que poseen los dispositivos.

Posteriormente, en el uso Offline de la aplicación, se accederá a la copia local de los archivos y una vez restablecida la conexión se sincronizarán los archivos almacenados en el servidor y el resto de las copias en el sistema.

Uso optimizado de energía y performance

Se debe permitir al usuario configurar el sistema de manera que se obtenga una buena relación performance-energía. Un sistema que intente mantener la consistencia mediante un chequeo continuo de la validez de la copia local puede traer como consecuencia un desgaste de energía, reduciendo el tiempo de autonomía del dispositivo móvil.

Conectividad continua

La falta de conexión debe ser transparente para el usuario, la aplicación debe utilizar mecanismos que le permitan hacer uso rápidamente de las conexiones existentes y ser capaz de continuar funcionando, aún cuando no existan conexiones disponibles. Además, ante la presencia de múltiples conexiones, este mecanismo debe ser capaz de seleccionar aquélla que le brinde una mejor performance.

4.4.1.2 Modelo de arquitectura de documentos compartidos en un ambiente móvil

Componentes del cliente

- *Document Management Application*: es la aplicación que utiliza los documentos compartidos, la cual es capaz de comprender la estructura interna de los mismos.
- *File System Agent*: este componente se encarga de administrar las solicitudes que realiza el Document Management Application, determina si la solicitud puede ser satisfecha mediante el uso de documentos

locales o remotos, administra la cache de documentos de acuerdo a los parámetros configurados por el usuario.

- *Content Adapter*: forma parte del File System Agent y su función es la de adaptar el contenido de los documentos compartidos a las características del dispositivo móvil y la estructura del documento.
- *Reintegration Manager*: forma parte del File System Agent y se encarga de resolver conflictos de versiones producto de las actualizaciones de los documentos por parte del usuario. De ser posible intentará resolver los conflictos en forma automática, cuando no lo sea hará uso de la aplicación asociada al documento y, como último recurso, solicitará la intervención del usuario.
- *Local File System*: se encargada de mantener todos los documentos locales.
- *Remote File Cache*: es un repositorio de copias de documentos originados por sistemas de archivos remotos.
- *System Capabilities Supporting Mobility*: administra las conexiones y la energía disponibles en los dispositivos. Posee los siguientes elementos:
    - *Connection Awareness*: brinda información sobre el estado de la red y energía.
    - *Message queuing*: se utiliza para ocultar los estados de conexión de la red y asegurar confiabilidad.
    - *Security*: se encarga de proporcionar medidas de seguridad sobre los nodos móviles, protocolos inalámbricos y a las conexiones poco fiables.

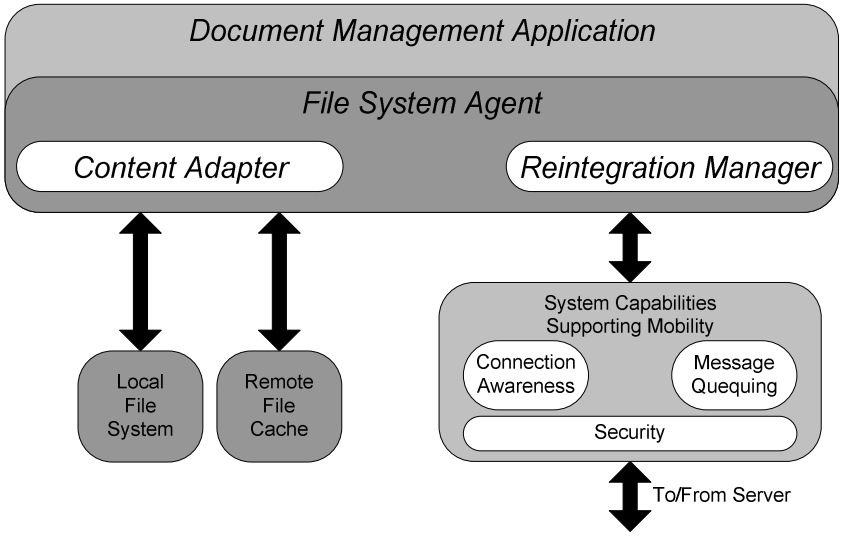

Figura 6. Arquitectura de documentos compartidos. Componentes del cliente.

Componentes del servidor

- *Document Sharing Application*: administra los documentos compartidos. Recibe y procesa las peticiones del File System Agent en cada cliente. Lleva registro sobre un log de los documentos compartidos que han sido modificados y cuáles son sus diferentes versiones. Al igual que el cliente puede poseer un Content Adapter que tiene la misma función que el Content Adapter en el cliente. La decisión de la ubicación de este componente depende de los recursos disponibles en los dispositivos.
- *Shared Files*: archivos compartidos a los cuales los usuarios pueden acceder.
- *System Capabilities Supporting Mobility*: su funcionalidad es análoga a la que tiene este componente del lado del cliente.

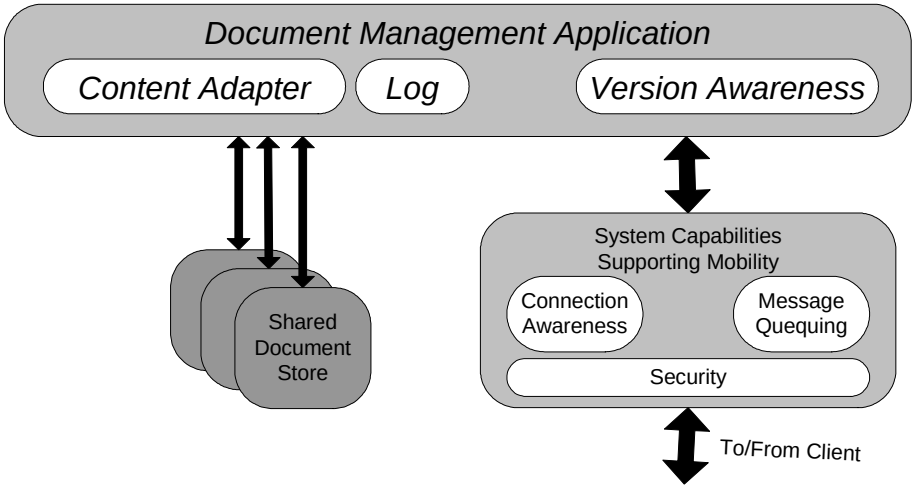

Figura 7. Arquitectura de documentos compartidos. Componentes del servidor.

4.4.1.3 Análisis del modelo de compartición de documentos en ambientes móviles

Reintegración y estructura de los documentos

Si el elemento de reintegración, Reintegration Manager, tuviera información sobre la estructura interna del archivo compartido, podría realizar una reintegración de sólo una parte del mismo, comportándose de esta manera, en forma similar a una Base de Datos en la cual se reintegran sólo los registros modificados.

Aquellos documentos que tengan una estructura rígida y conocida pueden ser reintegrados automáticamente. Por otro lado, en aquellos documentos cuya estructura no pueda ser reincorporada parcialmente, una reintegración de menor nivel del archivo resultará imposible.

Ubicación del adaptador de contenido

Este componente, Content Adapter, que puede estar ubicado tanto en el cliente como en el servidor, requiere acceso a lo siguiente:

Recursos del sistema: debido a que las versiones adaptadas de los documentos suelen ser de menor tamaño que las almacenadas en el servidor y que el servidor generalmente posee mayor capacidad de proceso que el cliente, es deseable que el adaptador de contenidos se encuentre en el servidor.

Conocimiento del formato del documento: la aplicación que utiliza los documentos es quien mejor conoce la estructura de los mismos. Un servidor

que contiene documentos no necesariamente debe conocer la estructura de los mismos. Un cliente que haya requerido un documento con seguridad dispondrá de una aplicación que pueda interpretar su estructura, por lo cual ubicar el adaptador en el cliente garantiza el acceso a la aplicación que lo gestiona y por lo tanto a la estructura del mismo.

Conocimiento de los recursos del sistema destino: para que un servidor pueda adaptar el contenido de un documento necesita disponer de información sobre los recursos de cada uno de los clientes, o el cliente debe enviarle una descripción de sus recursos ante cada solicitud que realiza al servidor.

Si la adaptación se realiza del lado del cliente, el módulo encargado de adaptar el archivo puede acceder fácilmente a la información de los recursos del mismo.

Sincronización completa o incremental del árbol de directorio

El sistema de compartición de documentos debe garantizar la coincidencia entre las versiones de los documentos del servidor y aquellos que se encuentran en la cache de los clientes. El cliente, por lo tanto, debe solicitar al servidor periódicamente una instantánea del árbol de documentos que éste mantiene la cual incluye versiones de cada archivo. Existen dos métodos de sincronización. La sincronización completa garantiza la coincidencia de versiones entre el cliente y el servidor, con el inconveniente ocasionar grandes demoras si se deben sincronizar gran cantidad de documentos. La sincronización incremental contiene sólo los cambios realizados desde la última sincronización; este método es más eficiente que la sincronización completa pero puede introducir pérdidas de coincidencias de versiones.

Si el servidor conoce qué clientes están accediendo a qué documentos y el modo de acceso, puede evitar modificaciones simultáneas de archivos que produzcan inconsistencias y notificar a los clientes cuando los documentos que poseen han sido modificados por otro usuario. Esta información puede ser almacenada en una base de datos.

En la mayoría de las situaciones una consulta periódica por parte del cliente para detectar cambios en las versiones de los archivos del servidor es eficiente y evita añadir la complejidad innecesaria al servidor para que gestione los clientes que acceden a los documentos.

4.4.2 Arquitectura de base de datos

Normalmente los usuarios acceden a un servidor de base de datos a través de una conexión física de red. El procesamiento se lleva a cabo totalmente en el servidor, lo cual impide trabajar sin una conexión entre el mismo y el cliente. El servidor puede atender solicitudes simultáneas de varios clientes, para ello hace uso de mecanismos de bloqueo que impiden que las concurrencias provoquen inconsistencias.

El escenario en un ambiente móvil es diferente al planteado anteriormente. No existe una conexión permanente de los clientes con el servidor, por lo cual los clientes deben ser capaces de almacenar copias de los datos del servidor, para poder trabajar con ellos en períodos de desconexión y posteriormente, cuando se disponga de alguna conexión, sincronizarlos con los datos almacenados en el servidor.

En el momento de diseñar una arquitectura de base de datos para un ambiente móvil se deben tener en cuenta los siguientes inconvenientes:

- Los clientes que se encuentran desconectados no serán capaces de percibir las actualizaciones realizadas por otros clientes hasta el momento en que se sincronicen, lo que puede provocar que se trabaje con datos desactualizados o no válidos.
- Los conflictos que se producen a causa del trabajo Offline, deben ser resueltos en el momento de la sincronización. En muchas ocasiones el servidor será incapaz de resolver estos conflictos. En dichas ocasiones los conflictos serán resueltos por el cliente o mediante la intervención del usuario.
- El poder de procesamiento que puede tener un cliente es inferior al que puede brindar el servidor.
- Los datos propietarios deben mantenerse seguros en las ubicaciones remotas.

Una arquitectura de base de datos para aplicaciones móviles debe permitir al cliente seleccionar los datos que tendrán disponibles durante el período de desconexión.

La cantidad de datos a almacenar estará limitada por la capacidad de almacenamiento del dispositivo móvil.

La aplicación cliente debe tener un buen desempeño, tanto en modo conectado como desconectado, y las transiciones entre estos modos deben ser transparentes para el usuario.

En un período de desconexión el usuario debe ser capaz de tomar decisiones basadas en los datos que dispone, pero debido a que los mismos pueden encontrarse desactualizados, éstos deben ser marcados para que la decisión tomada pueda ser actualizada cuando se restablezca la conexión.

Habitualmente, en una transacción se establece una conexión entre el cliente y el servidor, se transfieren datos y una vez que la misma haya terminado, se envía una notificación sobre el resultado de la misma. En un ambiente móvil la arquitectura de base de datos debe asegurar que las transacciones se ejecuten correctamente mediante el uso de algún mecanismo, como puede ser el uso de colas para almacenar transacciones que serán transmitidas posteriormente al servidor. El usuario continuará trabajando y accediendo a los datos a pesar de que los mismos no hayan sido actualizados en el servidor.

4.4.2.1 Modelo de arquitectura de base de datos en un ambiente móvil

Componentes del cliente

- *Transport Proxy*: servicio que se encarga de almacenar mensajes enviados desde fuentes externas de manera confiable. Se encuentra accesible para todas las aplicaciones, con las cuales interactúa enviando y recibiendo mensajes, para lo cual las aplicaciones deben registrase mediante el DataBase Proxy.
- *Policy Manager*: recibe mensajes del Context Manager con información sobre el contexto en el cual se encuentra el dispositivo móvil. Dentro de esta información se encuentran los tipos de redes disponibles, calidad de las mismas, tiempo de autonomía de la batería. Esta información es utilizada por el Policy Manager para tomar decisiones en función de políticas definidas por el usuario.
- *Context Manager*: servicio que se encarga de mantener actualizada la información del contexto, monitoreando continuamente los diversos dispositivos. De esta forma recaba información sobre conexiones

disponibles junto con sus características, tipo de conexión, ancho de banda y costo, el estado de la batería y estado de los dispositivos de almacenamiento, por ejemplo. Esta información es posteriormente utilizada por el Policy Manager para la toma de decisiones.

- *Database Proxy*: servicio que se encuentra entre la aplicación y la base de datos y entre ésta y la cola de mensajes del Transport Proxy. La aplicación hace requerimientos a la base de datos a través del Database Proxy, el cual mediante ODBC (Open DataBase Connectivity) o JDBC (Java DataBase Connectivity) realiza la comunicación con la base de datos. Cuando el sistema se encuentra en modo conectado se comunica con el Transport Proxy para poder almacenar datos localmente permitiendo en todo momento al Database Proxy tener conocimiento sobre los nuevos datos en la base local.

  Ya que, el Database Proxy conoce qué datos se encuentran en uso y cuáles han, sido actualizados puede comunicarse con la aplicación para informar sobre estas modificaciones. De esta manera la aplicaciones pueden llevar a cabo acciones para cumplir con las políticas definidas por el usuario. Un ejemplo de estas acciones puede ser la actualización de los datos disponibles localmente.
- *Client mobility Proxy*: servicio de comunicación entre la aplicación y los diferentes proxys. La aplicación, a través de una interfaz, notifica al Client Mobility Proxy los eventos que se han producido, como puede ser la actualización de algún dato. A partir de esta notificación el usuario podrá tomar la decisión de forzar la sincronización completa o parcial de los datos.

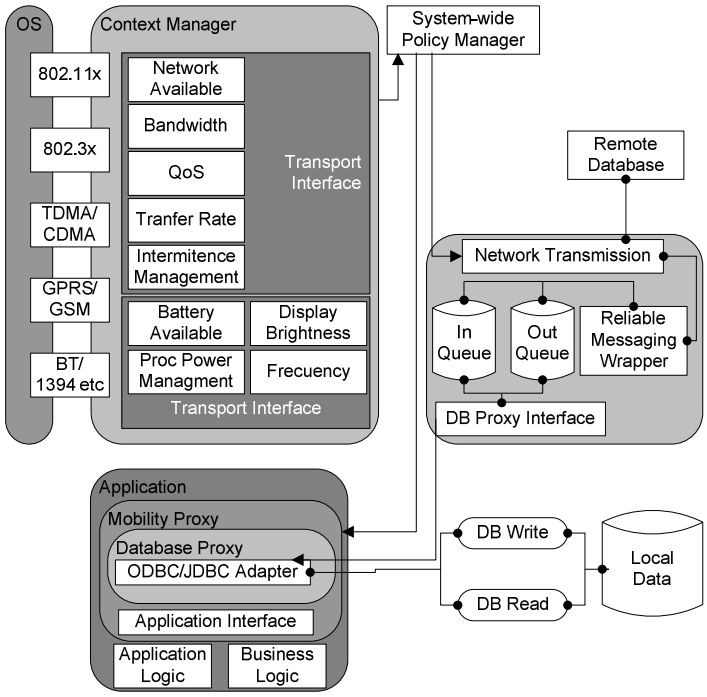

Figura 8. Arquitectura de base de datos. Componentes.

Componentes del servidor

- *Transport Proxy*: servicio que lleva un registro de las direcciones destino de los clientes. Se comunica con los servicios sign on y verifica la identidad del cliente, mantiene la relación entre el cliente y su dirección IP (Internet Protocol), de tal manera que cuando esta última cambia se detiene el envío de los paquetes que se encuentran en la cola de salida, los cuales son redirigidos a la nueva dirección del cliente.
- *Database Proxy*: componente que utiliza ODBC, JDBC o cualquier otro adaptador para gestionar las actualizaciones a la base de datos maestra que se encuentran en la cola de entrada. Estas actualizaciones se realizan de acuerdo a prioridades y políticas de usuario. Ejemplo de envío de datos: se pueden enviar mensajes al cliente para alertar sobre qué datos a los que está suscripto han sido modificados. De acuerdo a las políticas definidas por el usuario se disparará o no la actualización desde el servidor.

La arquitectura propuesta no define un mecanismo de resolución de conflictos ni de seguridad, sólo son mencionados, pero no descriptos en detalle. Tampoco describe mecanismos para resolución de conflictos de

sincronización, como pueden ser operaciones que se basan en datos desactualizados o que se encuentran en conflicto.

A diferencia de un ambiente en donde la interconexión esta garantizada y los datos se almacenan en una única base de datos central, en un ambiente móvil los datos se replican en los dispositivos cliente con el objetivo de que los mismos estén disponibles cuando el dispositivo carezca de conexión con el servidor. Este mecanismo de replicación de datos hace necesario métodos para la actualización cuando el dispositivo vuelva a tener conexión con el servidor.

4.5 Herramientas de desarrollo

4.5.1 Plataformas de desarrollo

Para el desarrollo de aplicaciones móviles existe una gran variedad de plataformas. En el momento de seleccionar la plataforma sobre la cual se desarrollará una aplicación se deben tener en cuenta varios factores, como por ejemplo, las características de los dispositivos sobre los cuales correrá la aplicación y el lenguaje con que está familiarizado el desarrollador, entre otros.

Entre las plataformas más utilizadas se encuentran:

4.5.1.1 J2ME

Es una versión reducida de Java orientada a dispositivos móviles. La baja potencia de cálculo y las limitaciones presentes en las interfaces de los dispositivos móviles hacen que el uso de versiones de Java tales como J2SE o J2EE, no fuese posible. Debido a estas restricciones fue desarrollada una versión reducida de J2SE dando origen así a Java 2 Micro Edition (J2ME).

4.5.1.2 BREW

Binary Runtime Environment For Wireless es una plataforma desarrollada por la empresa Qualcomm para ofrecer una herramienta de desarrollo de aplicaciones dinámicas y altamente gráficas en ambientes móviles. Esta tecnología está basada en los lenguajes C y C++. BREW ofrece un Market desde el cual se pueden administrar y adquirir nuevas aplicaciones,

permitiendo que los dispositivos móviles sean como pequeñas computadoras a medida.

4.5.1.3 .NET Compact Framework

Es una plataforma creada por Microsoft para el desarrollo de aplicaciones móviles. Esta plataforma está basada en los servicios Web XML (eXtensible Markup Language) y fue desarrollada teniendo en cuenta las limitaciones de los dispositivos móviles. Las aplicaciones que se desarrollan con esta herramienta son soportadas por dispositivos Pocket PC, Smartphones o equipos que utilizan Windows CE, Windows CE .NET MSDN, Microsoft .NET Compact Framework.

4.5.1.4 WML

Wireless Markup Lenguaje es un lenguaje utilizado para el desarrollo de páginas que serán visualizadas en dispositivos que utilizan la tecnología WAP. Entre sus principales características se encuentran el soporte de imágenes y texto con formato, navegación al estilo Web, uso de variables y formularios para realizar intercambios de información entre el dispositivo móvil y el servidor.

4.5.1.5 SQL Server Mobile

Es un motor de base de datos reducido diseñado para ejecutarse en dispositivos móviles, en los cuales se posee un reducido poder de procesamiento. Esta versión no requiere licencia, salvo que la versión compacta se comunique y sincronice con un servidor SQL mayor, para lo cual sí se hace necesaria.

4.5.2 Emuladores

Son herramientas que emulan el SO y las características del dispositivo para el cual se desarrolla una aplicación. De esta forma se puede probar el comportamiento de la aplicación sin necesidad de contar con un dispositivo físico.

4.5.2.1 Windows

El entorno de desarrollo Visual Studio.NET posee emuladores básicos para cada una de las plataformas que soporta. Además de contar con estos emuladores, se pueden descargar en forma gratuita desde la página de Microsoft imágenes de nuevos dispositivos o versiones del SO, mediante los cuales es posible probar los ejecutables de las aplicaciones desarrolladas o las aplicaciones ASP.NET Mobile haciendo uso del Pocket Internet Explorer.

4.5.2.2 Palm OS

Este emulador sirve para reproducir la manera en que se desempeña una aplicación móvil en un browser de Palm OS. Existen emuladores para las diferentes versiones del SO Palm OS que pueden descargarse desde la dirección Web http://palm-os-emulator.softonic.com/palm. El emulador de Palm OS realiza la emulación tanto de hardware como del SO. Además del emulador es posible descargar un simulador para aquellos equipos Palm sobre chips Intel, que emulan el SO pero no el hardware.

4.5.2.3 Symbian OS

Existen pocos emuladores de Symbian OS que tengan su navegador Web disponible para utilizar. Es posible descargar algunos emuladores desde el sitio Web oficial http://www.forum.nokia.com/Library/Tools_and_downloads/Other/Symbian_SDKs/ o alguno de los emuladores de Nokia que pueden emular equipos Series 60 con Symbian OS.

4.5.2.4 Nokia

Esta empresa posee una gran cantidad de emuladores, los cuales traen incorporados el SO completo, incluyendo el Web browser que no requiere configuraciones adicionales. Es posible descargarlo gratuitamente desde http://www.forum.nokia.com/Library/Tools_and_downloads/Other/Symbian_SDKs/.

4.5.2.5 OpenWave

OpenWave Mobile SDK puede ser descargado desde el sitio http://ptf.com/download/openwave_phone_simulator/181977/. Esta herramienta

trae incorporado un simulador de teléfono que posee un browser, el cual puede ser utilizado para probar los desarrollos. Este browser se encuentra disponible en una amplia gama de teléfonos celulares de diferentes marcas.

4.5.2.6 Opera

Este emulador se descarga gratuitamente desde www.opera.com. El navegador Web Opera, comúnmente utilizado en computadoras de escritorio, también posee una versión desarrollada para diferentes plataformas móviles. La versión completa del navegador Opera para escritorio incluye un motor móvil que permite visualizar una página Web que se desee probar en la versión móvil. Presionando las teclas SHIFT+F11 se podrá acceder al contenido de la página adaptado a la pantalla de un dispositivo móvil.

4.5.2.7 Motorola Browser ADK

Éste es un emulador de descarga gratuito desde el sitio de Motorola para desarrolladores www.motocoder.com.

Este emulador proporciona una ventana en la cual se muestra la pantalla de un teléfono virtual junto con su browser en el cual se podrá probar la página Web desarrollada y otra ventana en la que se muestra el código fuente de la aplicación.

4.5.2.8 Android

Este emulador viene incluido en el SDK de Android, se puede hacer uso del mismo ejecutándolo como una aplicación o a través de un plugin del entorno de desarrollo, como puede ser Android Development Tools (ADT) para Eclipse. ADT posee diferentes skins que permiten simular diferentes dispositivos móviles incluyendo su apariencia, pantalla, teclado, botones, así como las aplicaciones instaladas en el dispositivo.

Se puede emular, haciendo uso de ADT, la ejecución de la aplicación sobre un procesador ARM. Además, ADT permite la simulación simultánea de múltiples dispositivos interactuando entre sí. También permite la simulación de llamadas y envío y recepción de mensajes.

CAPÍTULO 5 Diseño de Aplicaciones Móviles

La forma en la que un usuario utiliza una aplicación móvil difiere ampliamente de la forma en la que lo hace con una aplicación de escritorio, debido a que éstas presentan formas de interactuar y características diferentes.

El éxito de una aplicación depende en gran medida de cómo ésta fue diseñada. Las consideraciones que se deben tomar en el momento de diseñar una aplicación de escritorio pueden no ser las más adecuadas cuando se trata del diseño de una aplicación para dispositivos móviles. Consideraciones tales como el limitado poder de procesamiento, reducido tamaño de pantalla, tiempo de autonomía del dispositivo, capacidad de almacenamiento, entre otras, hacen necesario el uso de técnicas de diseño específicas para ambientes móviles.

5.1 Consideraciones de diseño en aplicaciones móviles

A continuación se detallan los tópicos fundamentales que deberían tenerse en cuenta en el momento de diseñar aplicaciones móviles.

5.1.1 Patrones de uso

Existe una marcada diferencia en la forma en que los usuarios de computadoras de escritorio y de los dispositivos móviles hacen uso de una aplicación. Esta diferencia se debe en gran medida a las características de estos tipos de dispositivos. Un usuario de escritorio dispondrá de dispositivos tales como teclado, mouse, pantalla de gran tamaño, impresoras y gran capacidad de procesamiento y almacenamiento, con los cuales puede manipular cantidades significativas de datos durantes períodos extensos. Por otro lado, los usuarios de dispositivos móviles suelen realizar consultas o actualizaciones de manera muy frecuente y desde diferentes lugares físicos, tareas que suelen ser específicas y de corta duración.

5.1.1.1 Tiempo de inicio

Debido a que los usuarios de dispositivos móviles suelen realizar accesos a las aplicaciones con mayor frecuencia que los que realiza un usuario de aplicaciones de escritorio, y estos accesos son de corta duración, minimizar

el tiempo que insume la aplicación en estar disponible se convierte en una necesidad imperiosa. Un usuario de una PC de escritorio puede tolerar que un procesador de textos u otra aplicación tarde varios segundos en iniciar su ejecución, pues este tiempo, comparado con el tiempo con el que hará uso de dicha aplicación, no representa una demora significativa. En cambio, una demora de 5 segundos en una aplicación móvil que será usada durante 15 segundos como puede ser el envío de un mensaje de texto, resulta intolerable. Una buena práctica sería lograr que el tiempo de inicio de la aplicación fuera despreciable con respecto al tiempo que dura la sesión de uso de la misma.

Habitualmente, para disminuir el tiempo de inicio de una aplicación se puede guardar el estado de la última sesión en el momento de cerrarse para ser restaurado nuevamente cuando la aplicación sea requerida, aunque esta solución es válida sólo para algunos tipos de aplicaciones. Una aplicación tal como la agenda de contactos, por ejemplo, debe estar siempre en ejecución para que su acceso sea de manera inmediata, sin ningún tiempo de inicio de por medio.

5.1.1.2 Enfocarse en el objetivo

El desarrollador de aplicaciones móviles debe determinar un conjunto que incluya los principales objetivos o tareas que la aplicación debe llevar a cabo. Una vez hecho esto se trata de lograr que las tareas pertenecientes a este conjunto se ejecuten de manera rápida, ágil y con el menor esfuerzo posible por parte del usuario. El uso de botones especiales para acceder a determinadas tareas o aplicaciones de uso frecuente es una buena práctica para hacer foco en el objetivo.

Durante la etapa de diseño es necesario identificar cuales serán las tareas que realice comúnmente el usuario, para que las mismas sean sencillas, de rápido acceso, minimizando la cantidad de datos que el usuario deba aportar para realizar la misma. No sólo se debe mostrar o requerir la información necesaria para poder realizar la tarea solicitada, sino que se deben proveer mecanismos para que este ingreso de datos se realice de la forma más ágil posible; un ejemplo de esto puede ser el uso de un control que permita al usuario seleccionar una fecha en lugar de escribirla.

5.1.2 Interfaz de usuario

El avance en la tecnología de dispositivos móviles provocó una mejora en la forma en que los usuarios pueden interactuar con éstos. Así mismo el incremento en las prestaciones de los dispositivos móviles permite ejecutar aplicaciones de mayor complejidad, tales como el manejo de documentos, transacciones, etc.

En la actualidad los usuarios de dispositivos móviles utilizan sus aplicaciones para administrar información, analizar y seleccionar datos y realizar transacciones sin necesidad de utilizar una computadora de escritorio. Por este motivo en el diseño de la aplicación es importante crear una interfaz con la que el usuario puede interactuar de manera adecuada.

5.1.2.1 Interacción con datos

Tanto las características de la pantalla como la forma en que se ingresan los datos afectan la interacción del usuario con los mismos.

- *Características de la pantalla*: el tamaño de la pantalla afecta directamente la forma en que el usuario interactúa con la aplicación; la misma debe ser lo suficientemente grande y poseer la definición necesaria para mostrar correctamente los diferentes componentes de una aplicación, tales como los menús, cajas de texto, listas de selección, entre otros. Además del tamaño de la pantalla, para lograr una presentación de la información de manera adecuada es necesario utilizarla eficientemente; esto se logra mediante una buena selección de la información mostrada, ocultando aquella que resulte irrelevante.
- *Modo de ingreso de datos disponibles*: para facilitar la carga de datos por parte del usuario se debe disponer de métodos de ingreso que agilicen la misma. Los dispositivos móviles actuales ofrecen variados mecanismos de ingreso de datos, tales como teclado QWERTY completo, touch screen, teclados en pantalla, touch pad, etc.

La aplicación debe ser desarrollada teniendo en mente que la misma será utilizada en dispositivos de pantalla reducida, permitiendo interactuar con ésta de manera ágil y sencilla. Muchos sitios, en el momento de ser accedidos utilizando una PC de escritorio, muestran gran cantidad de información,

imágenes, contenidos multimedia, publicidad o información y funcionalidades poco importantes. Sin embargo, si se accede al mismo sitio a través de un dispositivo móvil, como puede ser un teléfono celular, éste presentará sólo la información que se considere más importante, junto con links u otros mecanismos que permitan el acceso al resto de la información.

A diferencia del método de ingreso de texto de múltiples pulsaciones, T9 (Text on 9 keys) es un ejemplo claro de una buena solución que se ajusta a las limitaciones del dispositivo. Este mecanismo se utiliza para el ingreso de textos de manera ágil en teléfonos celulares convencionales cuyos teclados cuentan con tan sólo 12 teclas. Mediante el uso de T9 se produce un ahorro de tiempo en la entrada de datos debido a que el mismo infiere la información, como puede ser una palabra, a medida que se va ingresando, mediante el uso de un diccionario. Además, este mecanismo evita la espera que se produce cuando se ingresa una palabra que posee dos letras consecutivas que se encuentran en la misma tecla. El diccionario que utiliza T9 incorpora las palabras nuevas que se ingresen, lo cual hará que las inferencias futuras sean más acertadas.

5.1.2.2 Influencia del contexto en el uso de la aplicación

El contexto en el cual un usuario hace uso de una aplicación influye en la manera en que éste puede interactuar con la misma.

El uso de una aplicación que utiliza un mecanismo de reconocimiento de voz para interactuar con el usuario será dificultoso en un ambiente público o ruidoso en el que las interferencias impidan que este mecanismo funcione de manera correcta.

Una pantalla táctil con una interfaz pequeña con varias opciones puede resultar muy útil y ágil si se la utiliza en un ambiente que no se encuentra en movimiento; sin embargo el uso de este tipo de pantallas en un entorno en el cual se pueden producir saltos o vibraciones se convertirá en una herramienta de difícil manejo.

Otro aspecto que influye en el diseño de la aplicación es la forma habitual de uso del dispositivo; por ejemplo, los teléfonos celulares fueron creados para poder ser utilizados con una sola mano, los PDA fueron desarrollados para ser sostenidos con una mano y operados con la otra, y las

notebooks están pensadas para ser utilizadas con las dos manos y apoyadas sobre alguna superficie.

Para desarrollar una buena interfaz para una aplicación se debe tener en cuenta el contexto en el cual va a ser utilizada y el paradigma de uso del dispositivo sobre el cual será ejecutada la aplicación.

5.1.3 Interconexión de dispositivos

Las formas en las cuales el dispositivo móvil es capaz de conectarse a la red o con otros dispositivos influirán directamente en el diseño de las aplicaciones. Si bien un dispositivo móvil puede conectarse a una red mediante el uso de cables, lo más habitual es el uso de redes inalámbricas debido a la amplia proliferación que ha tenido este tipo de tecnología.

Actualmente los mecanismos de interconexión mas utilizados son:

- Infrarrojo o conexión por cable entre dos dispositivos móviles o entre un dispositivo móvil y una PC.
- Conexión dial-up a una red corporativa.
- Conexión inalámbrica a una red corporativa.
- Conexión de red inalámbrica a través de un teléfono celular u otro dispositivo con capacidades inalámbricas.
- Conexión a una LAN inalámbrica (802.11).
- Redes inalámbricas de área personal basadas en el estándar Bluetooth.

5.1.3.1 Selección de método de transferencia

Como ya se mencionó, las aplicaciones móviles son utilizadas con una alta frecuencia y durante sesiones de corta duración. Debido a esta corta duración de las sesiones, cuando se necesite transferir información ésta debe realizarse en el menor tiempo posible, por lo tanto es imprescindible seleccionar la manera más eficiente para llevar a cabo dicha transferencia de información desde y hacia el dispositivo móvil.

Dependiendo de la cantidad de información a ser transferida, del tiempo de validez de la misma, su importancia y el tiempo de respuesta requerido, algunos métodos de transferencia pueden ser más apropiados que otros.

Por ejemplo, en una aplicación móvil cuya finalidad es asistir a vendedores, el cambio en el stock de un producto es una cantidad de

información pequeña, que si se la entrega a tiempo tiene un gran valor para el vendedor. Lo ideal para este caso sería lograr que el stock se actualizara inmediatamente a través de una red inalámbrica.

5.1.3.2 Accesos a datos críticos

A pesar de disponer de diversos métodos de conexión, existirán momentos y lugares en los cuales el dispositivo móvil carezca de algún mecanismo de conexión.

Una posible solución a este problema es salvaguardar todos los datos críticos, sin los cuales el usuario no podría ejecutar sus tareas con normalidad, en el dispositivo móvil. Por lo general este tipo de información está disponible, accesible en tiempo real.

En el caso de una aplicación que asiste a vendedores de campo, un ejemplo de datos críticos podría ser la lista de clientes a los cuales el vendedor debe visitar y el listado de productos con sus propiedades, entre otros. La ausencia de estos datos impediría que el vendedor desarrollara su actividad eficientemente.

5.1.3.3 Seguridad

La importancia de contar con mecanismos de seguridad no es una característica exclusiva para entornos no móviles, sino que también es un aspecto crítico en los ambientes móviles, debido a que en ellos existe un gran número de amenazas.

Las políticas de seguridad en ambientes móviles deben proteger los datos tanto en el sistema back-end (servidor), como en el dispositivo móvil y en la transferencia entre ambos.

En todos aquellos sistemas donde haya objetivos potenciales de ataques maliciosos o con fines de diversión, habrá que incluir medidas de seguridad, en especial si estos sistemas tratan con transacciones financieras u otro tipo de información cuyo secreto e integridad sean críticos.

Para asegurar que la comunicación e intercambio de la información que realizan los dispositivos móviles no se vean comprometidos durante su transmisión, se utilizan técnicas criptográficas. La encriptación es el proceso de codificación de un mensaje de forma tal que oculte su contenido. Mediante esta

técnica, si alguien interceptara este intercambio de información, sería incapaz de obtener información útil ya que la misma se encuentra encriptada.

Los mecanismos de encriptación también son utilizados para la autentificación de usuarios. Las aplicaciones deben asegurar que la identidad de los usuarios que las utilizan sea correcta. Normalmente esto se realiza mediante el uso de usuario y contraseña, o mediante el uso de certificados.

Los datos sensibles que se encuentren almacenados en un dispositivo móvil deben ser protegidos mediante el uso de password y encriptación para evitar que éstos puedan ser accedidos por un usuario malicioso.

Algunos SO, como Palm OS, poseen mecanismos integrados de seguridad que pueden ser utilizados por aplicaciones que se ejecuten sobre ellos.

Un ejemplo de estos mecanismos puede ser Secure Socket Layer (SSL), proceso de encriptación que protege las comunicaciones entre dos procesos.

5.1.4 Requerimientos de confiabilidad

Debido al uso intenso de los dispositivos móviles, se espera de ellos un alto grado de confiabilidad.

5.1.4.1 Ejecución continua

Los dispositivos y aplicaciones móviles generalmente se encuentran en ejecución de manera continua durante largos períodos de tiempo. Por ejemplo, los teléfonos celulares y PDA suelen estar activos o en algún estado de standby las 24 horas del día, para asegurar que se encuentran disponibles en el momento en que se los necesita.

Debido a estas características los dispositivos móviles se asemejan a servidores, ya que éstos se encuentran siempre listos para brindar un servicio instantáneo a sus clientes.

5.1.4.2 Manejo de fallas

Las aplicaciones móviles se ejecutan en ambientes en los cuales existe un gran número de amenazas que pueden ocasionar fallas en su ejecución. Dentro de estas amenazas se encuentran: pérdida de las conexiones, cortes en suministro de energía debido a la capacidad limitada de las baterías, entre

otros. Además, el SO del dispositivo móvil puede detener la ejecución de determinadas aplicaciones cuando los recursos disponibles disminuyen.

El dispositivo móvil también puede ser extraviado o sufrir algún otro tipo de daño. El usuario debe tener la seguridad de que la información que manipula con el mismo no se verá afectada bajo ninguna de las circunstancias anteriormente descriptas. Es por esto que las aplicaciones móviles deben asegurar que los datos importantes para el usuario soporten daños inesperados o fallas.

5.1.4.3 Uso de la memoria

Las computadoras de escritorio normalmente utilizan una técnica denominada file swapping, en la cual se utiliza un archivo de intercambio para simular una cantidad de memoria mayor a la memoria física disponible, lo que permite la ejecución de un número mayor de aplicaciones. Esta técnica normalmente no es utilizada en dispositivos móviles, por lo cual las aplicaciones deben ser desarrolladas teniendo en cuenta que la memoria es un recurso escaso y debe ser utilizado en la forma más eficiente posible.

5.1.4.4 Servicios críticos

Habitualmente, en los dispositivos móviles se ejecutan múltiples aplicaciones de manera simultánea. Sin embargo, no todas las aplicaciones poseen la misma prioridad, algunas de ellas son críticas y no deben verse afectadas por el resto de las aplicaciones. El usuario del dispositivo móvil no desea que el fallo de alguna aplicación interrumpa o provoque una disminución del rendimiento de los servicios críticos que se encuentran en ejecución. Gran cantidad de SO para móviles garantizan un funcionamiento eficiente bajo cualquier circunstancia.

5.2 Diseño de Web móvil

En los últimos años el número de usuarios que acceden a sitios Web a través de dispositivos móviles se ha incrementado considerablemente. Como consecuencia de este incremento, los diseñadores Web deben adquirir conocimientos para poder diseñar sitios Web que puedan ser accedidos fácilmente desde estos tipos de dispositivos.

5.2.1 Tendencias para el diseño Web móvil

5.2.1.1 Opciones simples

Una de las características más importantes de los sitios Web móviles es la cantidad reducida de opciones que muestran al visitante. Debido a la falta de espacio en la pantalla y a las conexiones de Internet disponibles en los dispositivos móviles que suelen ser de un ancho de banda limitado, es importante que los visitantes del sitio accedan a la información que les resulte importante de la manera más rápida y con el menor número de pasos posible.

Las portadas de muchos sitios Web móviles ofrecen al visitante un reducido número de links mediante los cuales el visitante podrá acceder a la información que busca.

La simplicidad de los sitios Web es útil para alivianar la carga de aquellos sitios que son altamente concurridos. En éstos la información que carece de importancia es recortada, dejando solamente la información relevante.

Esta simplicidad logra que los sitios Web sean más útiles y sencillos, siempre y cuando las opciones presentadas permitan llevar a cabo las acciones que el usuario desea realizar.

5.2.1.2 Espacios en blanco

Los espacios en blanco en una interfaz son un aspecto importante de todo diseño, y a veces suele ser un desafío en el diseño de sitios Web debido a que lo deseable es tener tanto espacio en blanco como sea posible.

Estos espacios se tornan más necesarios en un diseño móvil dado que los tamaños de pantalla suelen ser menores.

Un sitio Web desordenado puede presentar una interfaz poco amigable y muy difícil de utilizar en ambientes móviles.

5.2.1.3 Falta de imágenes

En los últimos años la velocidad de las conexiones a Internet se ha incrementado de manera significativa, lo cual ha permitido a los diseñadores el uso de videos, imágenes y otros objetos multimedia que hacen gran uso del ancho de banda.

El visitante promedio en una computadora de escritorio o notebook desea ver sitios visualmente atractivos, lo que ocasiona que se utilice una gran cantidad de imágenes. Por el contrario, en lo que se refiere diseño para móviles, el uso excesivo de imágenes a menudo hace más daño que bien.

Existe una gran variedad de velocidades y planes de conexiones a Internet para móviles. El uso excesivo de imágenes puede hacer lento el acceso a sitios y dificultar la lectura del contenido de la página. Por estas razones, es muy común minimizar el uso de imágenes en el diseño de sitios Web móviles.

A medida que crece el número de usuarios móviles que utilizan smartphones con pantallas más grandes y mejores conexiones a Internet, se incrementan las oportunidades para usar imágenes. Sin embargo, debido a que aún existe un gran porcentaje de usuarios que no utilizan estos dispositivos, muchos sitios Web móvil continúan evitando el uso de imágenes cuando es posible.

5.2.1.4 Uso de subdominios en lugar de .mobi independiente o dominios separados

Cuando el dominio de nivel superior .mobi estuvo disponible, muchas compañías lo utilizaron para las versiones móviles de sus sitios Web. Actualmente es más común que las compañías utilicen un subdominio o una carpeta separada en su dominio principal para alojar las versiones móviles de sus sitios Web.

Existen muchas cuestiones que una compañía debe considerar cuando toma decisiones, pero uno de los beneficios más importantes del uso de subdominios es que se mantiene todo el sitio en un único dominio, en lugar de dispersar las cosas y confundir a los visitantes.

Ejemplos del uso de subdominios pueden ser sitios tales como m.twitter.com y m.facebook.com; y un ejemplo de uso de carpetas en el dominio principal sería www.lanacion.com.ar/movil/.

5.2.1.5 Priorizar contenidos

Debido a la simplicidad de las páginas móviles y la carencia de muchas opciones, el contenido que se muestra es altamente priorizado para el visitante.

Por supuesto, todos los sitios Web deberían poner foco en el usuario, pero debido a que la mayoría de los sitios Web son diseñados para propósitos comerciales, a menudo existen elementos que no son necesariamente importantes para éste, como puede ser un banner de publicidad. Mientras que los avisos publicitarios son ampliamente aceptados en Internet, la mayoría de los sitios Web para móviles se encuentran libres de publicidad.

El contenido disponible en los sitios Web móviles normalmente es de la más alta prioridad para el visitante móvil promedio, no para la compañía, sin embargo, la compañía se beneficia teniendo un sitio Web más usable.

5.2.2 Consideraciones para el diseño Web móvil

Las siguientes consideraciones deben ser tomadas en cuenta por diseñadores de sitios Web móvil durante su proceso.

5.2.2.1 HTML limpio y claro

El uso de un lenguaje claro y limpio ayuda a establecer una base sólida para un sitio Web móvil, asegurando que éste será mostrado correctamente por el navegador y brindará a los visitantes una experiencia placentera sin dificultades innecesarias.

5.2.2.2 Separación de contenidos y presentación con CSS

Además del uso de una semántica de etiquetado limpia, se necesita disponer de una separación del contenido de la página y la presentación de la misma.

Los visitantes móviles suelen visualizar los sitios Web con imágenes y CSS (Cascading Style Sheets) deshabilitados, para ellos lo más importante es acceder al contenido y a los links del sitio, dejando su presentación en un segundo plano.

Un sitio Web que utilice un etiquetado claro con CSS, que separe la presentación de los datos, es un buen comienzo para un sitio Web para móviles.

5.2.2.3 Etiquetas Alt

Las etiquetas de HTML Alt proveen una alternativa de texto cuando elementos no textuales, generalmente imágenes, no pueden ser mostradas,

brindando una mejor usabilidad del sitio. Si bien estas etiquetas deberían ser utilizadas siempre, su uso es más crítico en sitios Web móviles.

5.2.2.4 Campos de formularios etiquetados

Etiquetas en los campos de un formulario, al igual que utilizar etiquetas alt, ayuda a que el sitio sea más fácil de usar para los visitantes móviles. Sin el uso de estas etiquetas, el usuario no podrá determinar qué información debe ser ingresada en cada campo.

5.2.2.5 Uso de encabezados

Debido a la disponibilidad limitada de estilos de texto en los navegadores móviles, el uso de encabezados se vuelve más significativo.

Los navegadores móviles suelen no ser capaces de dar al texto el estilo que se desea, sin embargo, h1, h2, h3 y otras etiquetas similares ayudan a dar importancia al texto y construir la estructura de la página.

5.2.2.6 Evitar componentes flotantes

Incluso si un navegador móvil puede mostrar un sitio Web que utiliza componentes que flotan en él, es poco posible que el sitio Web se vea correctamente en pantallas pequeñas. Habitualmente el sitio Web tendrá un aspecto más amigable si no se hace uso de elementos flotantes apilando el contenido desde la parte superior.

5.2.2.7 Reducir márgenes y relleno

Los sitios Web móviles deben hacer uso de márgenes más reducidos que aquéllos disponibles en los sitios Web no móviles. Grandes cantidades de margen pueden hacer que el sitio Web tenga una disposición más torpe.

5.2.2.8 Navegación sencilla

La mayoría de los sitios Web tienen un menú de navegación en la parte superior de la página. Este tipo de menú es útil también en sitios Web móviles, aunque generalmente éstos muestran un conjunto reducido de estas opciones de navegación. De esta forma se proveen sólo los links más importantes brindando al visitante una manera fácil de navegar en el sitio Web móvil.

5.2.2.9 Buen uso de los colores

Debido a que las pantallas de los móviles no tienen la misma apariencia que las pantallas de notebook o computadoras de escritorio, el diseñador debe asegurar que el fondo de la página y el color del texto brinden un contraste adecuado para que el contenido pueda ser leído fácilmente.

CAPÍTULO 6 Patrones de Diseño

Los patrones brindan una manera de capturar y compartir conocimientos de diseño entre los diferentes agentes que intervienen en el proceso de desarrollo. Su objetivo es comunicar de manera concisa las particularidades de un problema y su solución.

Se denomina lenguaje de patrones a un conjunto de patrones interrelacionados para un dominio específico.

6.1 Definición de patrones

En el desarrollo de software, el término patrón se refiere, según la definición que aparece en www.whatis.com, a "un documento escrito que describe una solución general a un problema de diseño que se presenta en forma recurrente en diferentes proyectos". Los patrones describen el problema de diseño, la solución propuesta y cualquier factor que pueda afectar al problema o su solución.

Una característica importante de los patrones es que pueden estar anidados: un patrón solución puede tener cualquier tamaño; las soluciones para las diferentes partes del problema serían en este caso patrones más específicos.

Los patrones de diseño incluyen los siguientes tipos de información:

- Un nombre, que describe al patrón.
- La descripción del problema que se resuelve.
- El contexto en el que ocurre el problema.
- Solución propuesta.
- Contexto de la solución.
- Ejemplos y descripciones de casos de éxito y fracaso del modelo.
- Usos que se le puede dar y patrones relacionados.
- Autor y fecha.
- Referencias y palabras clave sobre el contenido.
- Código ejemplo de alguna solución, si fuera necesario.

Además, los patrones se pueden agrupar dentro de una jerarquía haciendo uso de clases de patrones, de esta manera se logra obtener un catálogo de patrones que da al diseñador una herramienta que puede brindar información acerca de los mismos.

6.2 Ejemplos de patrones

A continuación se presentan diferentes patrones a tomar en consideración para el diseño de aplicaciones móviles.

6.2.1 Patrón de sincronización

Descripción: se posee información idéntica dispersa en diferentes dispositivos móviles. Estos dispositivos carecen de una conexión continua, la que puede ser intermitente o nula.

Diferentes usuarios manipulan esta información aplicándole cambios, y estas actualizaciones muchas veces se producen de manera simultánea.

Contexto: diferentes usuarios poseen en sus dispositivos móviles una réplica de la información almacenada en un servidor. Mientras los usuarios desarrollan su actividad, generalmente en movimiento, se conectan en forma esporádica al servidor.

Problema: cuando se almacena información idéntica en diferentes ubicaciones, se pierde el sincronismo cuando los usuarios aplican cambios a su copia local y no se encuentran conectados al servidor. Esta situación no sólo ocasiona pérdida de sincronismo, sino que también puede ocasionar conflictos e inconsistencias en la información.

Solución: se provee a cada dispositivo de un motor de sincronización. El motor de sincronización se encarga de identificar las modificaciones que se aplican a la información local, intercambiar modificaciones con otros dispositivos cada vez que se cuenta con una conexión con el servidor, detectar conflictos e implementar un mecanismo de resolución de los mismos.

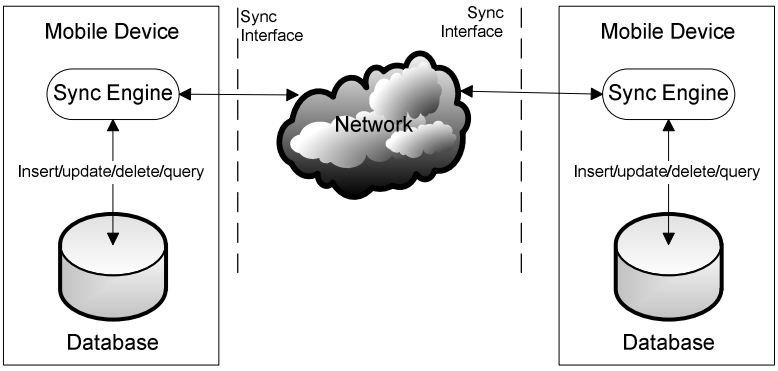

Figura 9. Patrón de sincronización.

Consecuencias: este patrón puede utilizarse en aquellas situaciones en las no existe una fuerte dependencia en la disponibilidad de datos actualizados. Los usuarios deben ser conscientes de que la información que obtienen del sistema puede ser modificada simultáneamente por otros usuarios, lo cual ocasiona conflictos en el momento de sincronizar la información con el servidor.

Para hacer efectivo el proceso de comparación de las diferentes versiones, la información debe ser estructurada en registros compuestos por campos, almacenados en tablas y registrar las modificaciones para cada uno de los campos. La información que no se encuentra estructurada en campos debe ser comparada registro a registro, lo cual incrementa la probabilidad de conflictos. Este patrón no es aplicable a información que no puede ser dividida en registros, tal como pueden ser video o audio.

Ejemplos: SyncML, framework para sincronización de datos en ambientes móviles. SyncML fue incorporado al proyecto OMA (Open Mobile Alliance), el cual es impulsado por Ericsson, IBM, Motorola y Nokia.

Palms HotSync, que permite sincronizar un dispositivo Palm con una o más PC.

6.2.2 Patrón de proxy remoto

Descripción: se dispone de un dispositivo que carece de la capacidad necesaria para llevar a cabo la tarea requerida. Este dispositivo se conecta a otro con mayor poder de cómputo, el cual se encargará de realizar la tarea que el dispositivo móvil no pueda realizar.

Contexto: un usuario que utiliza su dispositivo móvil para navegar en Internet. Como se ha dicho anteriormente, estos dispositivos móviles, poseen

un bajo poder computacional y una baja resolución de pantalla, por lo cual se hace difícil mostrar los diferentes componentes de las páginas Web.

Problema: se requiere que los dispositivos minimicen el ancho de banda utilizado, al igual que la cantidad de procesamiento requerido, y brinden una capacidad de entrada/salida acorde a las capacidades disponibles localmente.

Solución: el dispositivo móvil, en lugar de conectarse directamente a la red, identifica y se conecta con otros dispositivos o servidores que realizan el procesamiento requerido. Este tipo de servidores se denominan Proxy. La tarea del Proxy es aceptar solicitudes de servicios desde los dispositivos móviles, conectarse con el proveedor de dicho servicio para que realice la tarea requerida, procesar el resultado y enviar la respuesta al dispositivo móvil.

La aplicación se encuentra distribuida entre los dispositivos cliente y Proxy. Dentro de su estructura se identifican las siguientes entidades:

- *Cliente*: dispositivo móvil con una interfaz de usuario de la aplicación.
- *Proxy*: es un equipo, generalmente no móvil, encargado de ejecutar los procesos que requieren mayor poder de cómputo.
- *Interfaz pública*: establece la comunicación entre el Proxy y la red. Esta interfaz es idéntica a la que existe entre la red y un dispositivo no móvil.
- *Interfaz del Proxy*: no necesariamente es pública para todos los dispositivos de la red, sino que lo es sólo para aquellos dispositivos móviles que pueden hacer uso del Proxy. La comunicación entre un dispositivo móvil y un Proxy estacionario generalmente es inalámbrica.

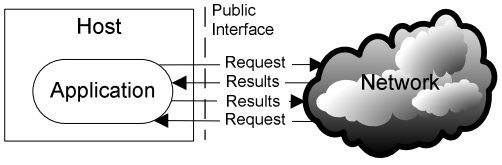

Figura 10. Comunicación entre una computadora y la red.

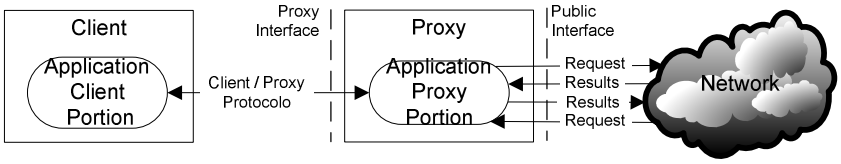

Figura 11. Patrón de Proxy remoto.

Consecuencias: permite que dispositivos con bajo poder de cómputo realicen tareas que demandan gran cantidad de procesamiento.

Este patrón posee como ventaja que mientras la interfaz pública permanezca sin cambios el resto de la red permanece sin cambios utilizando la misma infraestructura y protocolos. Como desventaja el uso de Proxy posee dos puntos de falla: el Proxy propiamente dicho y la conexión entre éste y el dispositivo, si cualquiera de estos puntos falla, la ejecución de las tareas requeridas por el usuario se ve imposibilitada.

Para que el cliente se comunique con el Proxy éste debe conocer su dirección o ser capaz de obtenerla a partir de algún servicio determinado como puede ser el servicio de nombres de dominio.

El Proxy deberá encontrarse disponible de manera continua para atender las solicitudes realizadas por los clientes.

Ejemplos: browse-it permite a los usuarios de los dispositivos móviles navegar sin las limitaciones que esta clase de dispositivos posee. Para ello el Proxy se encarga de procesar las páginas solicitadas y adaptarlas a las características del dispositivo.

6.2.3 Catálogo de patrones arquitectónicos de sitios Web para móviles

En los últimos años se ha puesto mucho esfuerzo en desarrollos de sistemas Web de información móvil (MobiWIS/MWIS Mobile Web Information System), lo cual requiere nuevas técnicas de Ingeniería de Software. Los ítems más críticos en el desarrollo de MWIS son la infraestructura de comunicación, las limitaciones de los dispositivos y la necesidad de una continua evolución.

Debido a este auge de aplicaciones Web móviles, se ha definido una serie de patrones para este tipo de aplicaciones. Estos patrones enfocan en las arquitecturas que mejor se ajusten a las aplicaciones, en lugar de hacerlo sobre cuestiones técnicas.

Para poder resolver problemas en un alto nivel de abstracción mediante la reutilización de soluciones exitosas, es necesario disponer de un lenguaje de

diseño de alto nivel que describa soluciones arquitecturales para aplicaciones móviles.

Si bien la siguiente descripción de clases no intenta ser definitiva, podría ser utilizada como un conjunto de ejes organizacionales para describir la naturaleza de los diferentes problemas arquitecturales de las aplicaciones.

Acceso Web

Aborda temáticas tales como la manera en la que el acceso a la aplicación desde un cliente móvil se ve afectado por las restricciones impuestas por los requerimientos y el dispositivo. Las limitaciones del hardware tienen un impacto importante en la manera en la que el dispositivo accede a la información. Un usuario con un teléfono móvil podrá acceder a características o sistemas a los que un usuario con un PDA Stand-alone no podrá acceder. Muchas veces el desarrollador de una MWIS debe idear una solución que funcione tanto para usuarios que disponen de gran autonomía como para aquéllos con capacidades limitadas.

Navegación Web

Esta clase trata la forma en la que el usuario móvil pretende navegar la aplicación Web. Un usuario de escritorio posee la capacidad de navegar por la totalidad de la Web de una manera sencilla, sin embargo, un usuario de un dispositivo móvil generalmente puede acceder sólo a un conjunto reducido de la Web. Además, en los casos en los que el usuario móvil fuese capaz de acceder a la totalidad de la red, es posible que sólo desee navegar un subconjunto de la misma que le es relevante según el contexto.

Por ejemplo, los usuarios de PDA disponen de un browser para acceder a la Web. Muchos de los browsers disponibles en los dispositivos móviles no son capaces de mostrar correctamente la totalidad de las páginas. Algunos sitios Web están codificados de manera tal que los mismos puedan ser visualizados sin problemas en esta clase de dispositivos.

Personalización

Esta clase trata sobre patrones arquitecturales que resuelven problemas de personalización o configuración de una aplicación móvil. Las limitaciones de los dispositivos móviles imponen restricciones en el momento de presentar la información en el mismo. Como se dijo anteriormente, es probable que un

dispositivo pueda no ser capaz de interpretar cada codificación en cada sitio Web. Sin embargo el usuario de un dispositivo móvil puede no estar interesado en una misma vista de la información que uno de escritorio.

Por ejemplo, algunos browsers de dispositivos móviles no soportan completamente la sintaxis HTML o poseen limitaciones en el tamaño y definición de pantalla, lo cual ocasiona que algunas páginas no puedan visualizarse o sólo se visualice alguna parte de ellas.

6.2.3.1 Patrón de acceso Web: acceso estático

Propósito: proveer al cliente móvil la posibilidad de acceder a una aplicación Web móvil cuando este cliente por sí mismo no es capaz de acceder a la Web.

Problema: el dispositivo móvil no posee la capacidad de acceder a la Web por sí solo, por ejemplo, una PDA sin modem o wireless, o el mismo se encuentra en un área en la cual no puede establecer una conexión.

Solución: proveer al cliente móvil el acceso al contenido Web mediante un medio de acceso no móvil, haciendo uso de otro dispositivo que pueda acceder a la Web.

Funcionamiento: la ilusión del cliente móvil de conexión a la Web se realiza mediante un mecanismo de conexión no móvil llamado conector de sincronización. En el conector de sincronización reside el Request Handler que hace de intermediario entre el cliente móvil y la interfaz Web. La información de una aplicación móvil es recuperada mediante el motor de actualización si la misma se encuentra desactualizada. Cuando un cliente móvil desea acceder a la Web el conector de sincronización recibe el requerimiento y lo encamina a través del Update Manager, que decide qué información reunir desde la Web, y por último a través del Web Connection acceder físicamente a la Web.

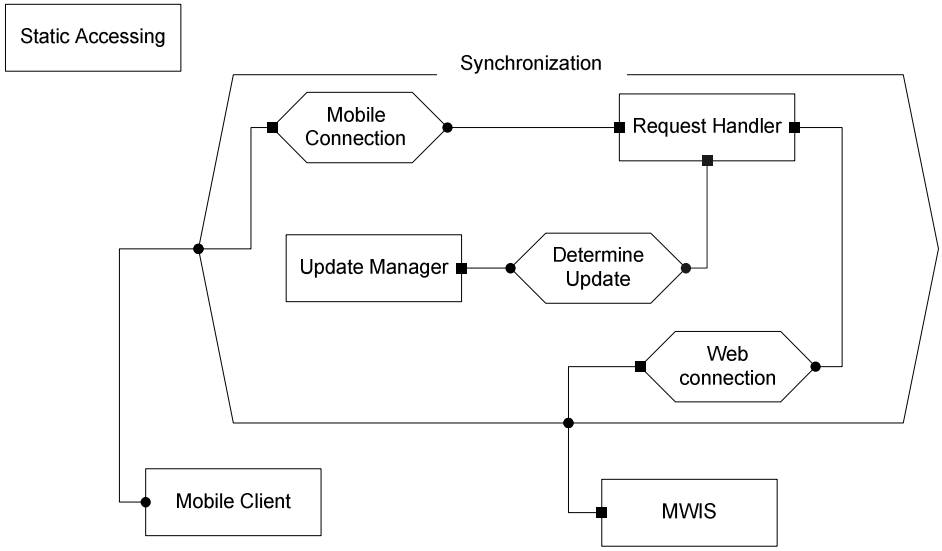

Figura 12. Patrón de acceso web: acceso estático.

Consecuencias: los dispositivos móviles que no disponen de capacidad de conexión pueden acceder a la Web si poseen un conector de sincronización adecuado, lo que puede implicar un trabajo adicional para los desarrolladores. La autonomía del cliente es reducida debido a que el proceso de acceso requiere del mecanismo de sincronización.

Usos Conocidos: el servicio AvantGo utilizado por HotSync de Palm OS Technology para transferir información desde una aplicación Web al cliente móvil a través de sincronización con una computadora de escritorio.

6.2.3.2 Patrón de navegación: corredor de canal Web

Propósito: simplificar la capacidad de direccionamiento del cliente móvil Web. Disponer de direcciones Web procesadas por un intermediario.

Problema: se debe brindar al cliente móvil la capacidad de obtener información de diferentes sitios previamente establecidos. Para que el acceso a dicha información se realice de la manera más sencilla posible, se debe simplificar la forma de direccionamiento de las páginas. En el caso de que se desee restringir al cliente el acceso a locaciones arbitrarias de la Web, se debe contar con un intermediario que asuma la responsabilidad de direccionar toda la Web y presentar al cliente móvil un universo transformado.

Solución: contar con un Broker que interactúe con toda la Web presentando al cliente móvil una vista transformada en términos de Web

Channels, donde ésta es una locación Web adecuada para el contexto del cliente móvil.

Funcionamiento: el cliente móvil se conecta al Web Channel Broker a través del Web Connection. El Web Channel Broker encamina los requerimientos del cliente móvil. El Broker se encarga de determinar que MWIS se encuentra conectado para luego encaminar el requerimiento del cliente hacia el sitio a través del componente Web Interface.

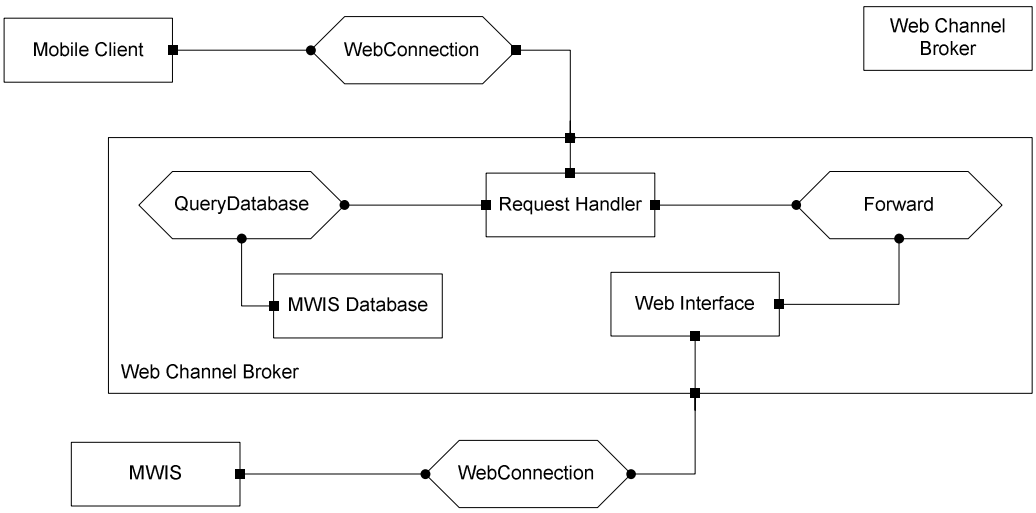

Figura 13. Patrón de navegación: corredor de canal Web.

Consecuencias: el cliente móvil obtendrá un sistema de direccionamiento simplificado mediante el uso de un Broker adecuado, lo cual implica un trabajo adicional para los desarrolladores. El Broker provee el control sobre que subgrupo de la Web que puede ser accedido por los clientes móviles.

Usos conocidos: AvantGo centraliza el acceso a un grupo de canales donde el Server AvantGo actúa como Broker entre la aplicación móvil y el cliente actual.

6.2.3.3 Patrón personalización: personalizador externo

Propósito: brindar un mecanismo que adapta el contenido Web al cliente móvil independientemente de la aplicación móvil a la cual se esté accediendo.

Problema: se debe acceder a información de una aplicación móvil desde un dispositivo cuya plataforma es incapaz de interpretar dicha información. Desarrollar versiones personalizadas de la aplicación móvil para cada dispositivo es una tarea impracticable debido a que existe una gran variedad de

dispositivos móviles y muchas veces no es de interés para el desarrollador que la aplicación pueda ejecutarse en diferentes plataformas.

Solución: crear un componente externo a la aplicación móvil que convierta la información brindada por la misma a un formato que pueda ser interpretado por el dispositivo objetivo.

Funcionamiento: el componente Transcoder es capaz de interpretar la codificación de un sitio y transformarla en una codificación que pueda ser manejada por el dispositivo móvil que intenta acceder a la información. El cliente móvil se conecta al Transcoder a través del Web Connection. El Transcoder transfiere el requerimiento desde el dispositivo móvil a la aplicación móvil haciendo uso de un conversor para adaptar estos requerimientos a un formato entendible por la aplicación móvil. Una vez convertidos los requerimientos son enviados a la aplicación móvil a través de la Web Interface. La aplicación responde a la solicitud del cliente, esta respuesta es tomada por el Transcoder quien utiliza un conversor que convierte esta respuesta en un formato interpretable por el cliente.

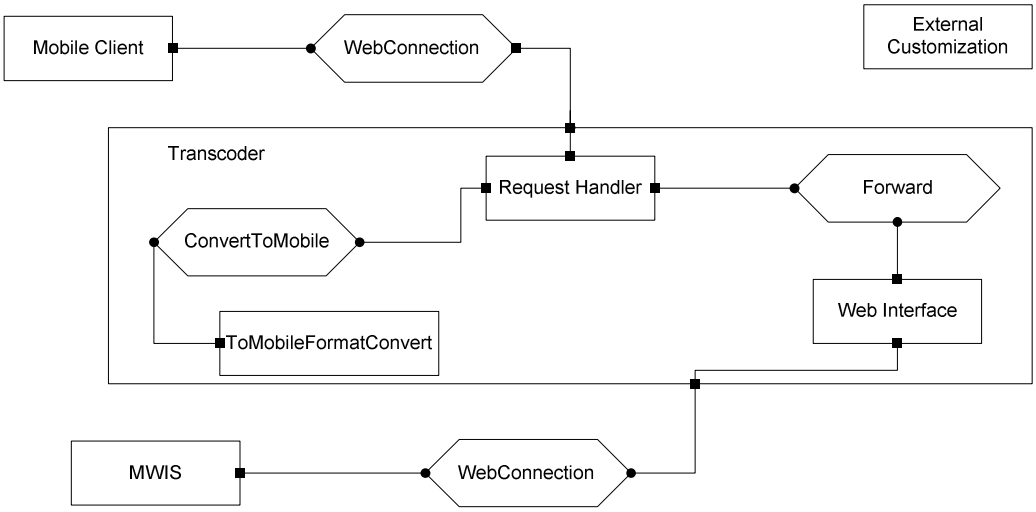

Figura 14. Patrón personalización: personalizador externo.

Consecuencias: los desarrolladores de aplicaciones móviles no deben preocuparse por personalizar la información para los diferentes clientes, debido a que el Transcoder lo realiza de manera automática y transparente. La gran variedad de clientes móviles hacen que el desarrollo de un Transcoder sea una tarea compleja.

Usos conocidos: PumaTech provee un servicio que habilita el acceso a páginas Web estándares por medio de un dispositivo PalmOS.

6.2.3.4 Patrón personalización: personalizador interno

Propósito: desarrollar una aplicación que brinde información, previamente adaptada, a los dispositivos cliente sin el uso de una personalización externa.

Problema: existen situaciones en las que es necesario adaptar la aplicación para que entregue al cliente contenidos que puedan interpretarse en forma dinámica o fija. La aplicación no sólo debe adaptar la información sino que también puede tener un comportamiento diferente dependiendo del cliente que realice el requerimiento. Debido a que las personalizaciones que se pueden realizar sobre la información están altamente relacionadas con la naturaleza de la aplicación, contar con un personalizador externo capaz de manejar todas las aplicaciones resulta casi imposible.

Solución: incorporar el personalizador a la aplicación para brindar diferentes respuestas adaptadas dependiendo del cliente.

Funcionamiento: el cliente móvil se conecta a la aplicación a través de una Web Connection, la aplicación utiliza diferentes fuentes de personalización para generar una respuesta apropiada para ser interpretada por el cliente móvil.

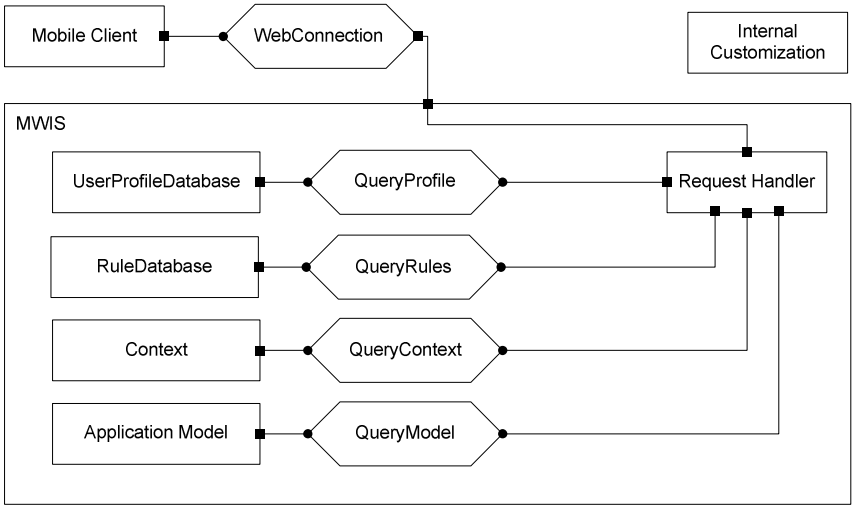

Figura 15. Patrón personalización: personalizador interno.

Consecuencias: debido a que la aplicación es diseñada teniendo en cuenta que brindará servicio a determinados clientes, disminuyen las

posibilidades de que la misma no se adapte al cliente. El desarrollador de aplicaciones debe realizar una tarea adicional diseñando aplicaciones específicas para los potenciales clientes.

Usos conocidos: en el modelo de referencia UWA (Ubiquituos Web Applications) las reglas están separadas de la aplicación y el modelo de contexto en sí está compuesto por los submodelos del usuario, el dispositivo y la red.

Patrones Relacionados: El patrón de personalización externo es similar al interno debido a que ambos poseen una forma de presentar una vista de la información adaptada al cliente móvil. Mientras que la personalización externa se enfoca en la apariencia de la información, la personalización interna lo hace en el proceso de generar la información.

6.2.3.5 Patrón de personalización: desarrollo personalizado

Propósito: diseñar un mecanismo de personalización de una aplicación móvil que no sea costoso.

Problema: una aplicación presenta información a un dispositivo móvil que no puede ser interpretada por el mismo, además no se cuenta con los recursos para desarrollar o comprar un mecanismo de personalización automática.

Solución: cada aplicación debe proveer una versión personalizada que se adecue al cliente.

Participantes: el cliente obtiene la información directamente de la aplicación a través de un Web Connection. Debido a que la aplicación fue personalizada para el cliente móvil, no se necesita ningún procesamiento adicional.

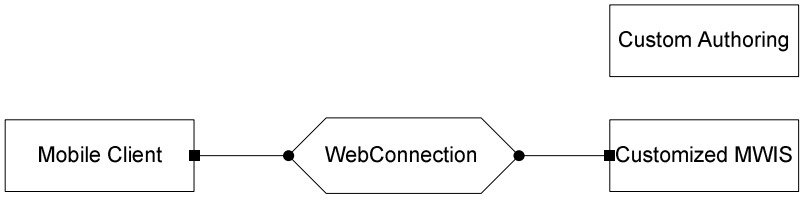

Figura 16. Patrón de personalización: desarrollo personalizado.

Consecuencias: el desarrollador de la aplicación móvil debe diseñar una versión personalizada de la misma para el cliente objetivo, de manera que no

es posible que la información que genere no se adecue perfectamente al cliente.

Usos conocidos: Las authoring guidelines de AvantGo motivan a los desarrolladores de aplicaciones móviles a que las mismas tengan versiones especialmente personalizadas a las limitaciones de los dispositivos móviles.

CAPÍTULO 7 Patrones Arquitectónicos

El objetivo de la Arquitectura de Software es describir un plan estructural de los elementos de un sistema, cómo estos elementos interactúan y cómo esas interacciones se adaptan al objetivo de la aplicación. Describe al sistema con un alto nivel de abstracción mediante el uso de vistas que incluyen los componentes computacionales del sistema y sus relaciones.

Al igual que los patrones de diseño, los patrones arquitectónicos intentan recuperar la experiencia del diseñador para que ésta pueda ser estructurada y transmitida a otros diseñadores. Permiten describir un problema y su solución de manera que ésta última pueda ser utilizada en diversas situaciones.

La mayoría de los patrones arquitectónicos poseen tres componentes: el problema, la solución y el contexto. La recopilación de patrones de un determinado dominio puede dar formar a un lenguaje que será utilizado por los desarrolladores, y les permitirá resolver problemas con un alto nivel de abstracción basándose en experiencias exitosas.

Los patrones arquitectónicos poseen un nivel de abstracción superior al de los patrones de diseño, debido a que en estos últimos sus componentes son las clases y objetos, y los mecanismos de interacción son los mensajes. En cambio, en los patrones arquitectónicos sus componentes son módulos de software y las interacciones entre ellos se realizan mediante conectores.

7.1 Clasificación de patrones

El dominio de las aplicaciones móviles, como ya se expresó, posee grandes restricciones a causa de las características técnicas de los dispositivos móviles y a las restricciones de las redes que éstos utilizan. Los temas de este dominio se pueden clasificar en los siguientes ejes fundamentales:

- *Temas relacionados con el acceso*: incluye todos los temas relacionados al acceso que realizan las aplicaciones cliente a las aplicaciones servidoras.

- *Temas relacionados con adaptación*: se refiere a cómo se adapta la información en función de las limitaciones de los dispositivos móviles en los cuales se ejecuta la aplicación.
- *Temas de personalización*: abarca temas relacionados con la forma en la que el usuario define el tipo de información que desea recibir, la red por la cual desea conectarse y otras características que el usuario puede personalizar.
- *Temas de seguridad*: se refiere a temas relacionados a la seguridad, confidencialidad e integridad de la información. Trata tanto la seguridad del dispositivo, como la seguridad de las conexiones.
- *Temas de interfaz*: se refiere a la temática de desarrollo centrado en el usuario.

Estos ejes fundamentales pueden utilizarse para realizar una clasificación de los patrones arquitectónicos. Debido a que las aplicaciones Web son un dominio relativamente nuevo, y que éste se encuentra en constante evolución, la siguiente clasificación no debe ser tomada como definitiva sino como una guía para la organización de los patrones.

Acceso

Tanto los requerimientos de las aplicaciones como las restricciones del entorno móvil afectan el acceso físico desde los dispositivos. Actualmente la mayoría de los dispositivos móviles cuentan con más de un tipo de conexión y las cuestiones relacionadas con el acceso son de gran importancia para los desarrolladores. Dentro de este tema se encuentran tópicos tales como: administración de la conexión, selección de caminos alternativos cuando se dispone de varias conexiones donde la selección se puede realizar de manera automática o manual, selección del mecanismo de compresión dependiendo del ancho de banda, seguridad, mensajería, sincronización de datos, entre otros.

Adaptación

Las cuestiones de adaptación surgen cuando un cliente móvil debe navegar por un sitio Web para acceder a información cuyo formato no es adecuado para el dispositivo móvil desde el cual se está accediendo. Por ejemplo, la adaptación de sitios Web que poseen flash o imágenes de gran

tamaño a los tamaños reducidos de las pantallas de los móviles. Las restricciones de pantalla, memoria y poder de cómputo juegan un papel preponderante en la adaptación de la información necesaria.

Personalización

Dentro de esta rama se encuentran aquellas cuestiones relacionadas con el uso de información que le permita al usuario configurar cuestiones tales como la información deseada, tareas a llevar a cabo en función del contexto, información a recibir; entre otras. En esta categoría se incluyen temas que también pertenecen a otras categorías, tales como la personalización del acceso a utilizar, la personalización de la adaptación de los sitios que se accede, entre otros.

Interfaz

Esta categoría abarca el diseño de interfaces que se adapten a las restricciones de los dispositivos móviles y la reingeniería de las interfaces de aplicaciones que serán transportadas a un entorno móvil. El éxito de una aplicación móvil depende en parte del uso de interfaces que le permitan al usuario interactuar ágilmente con la misma.

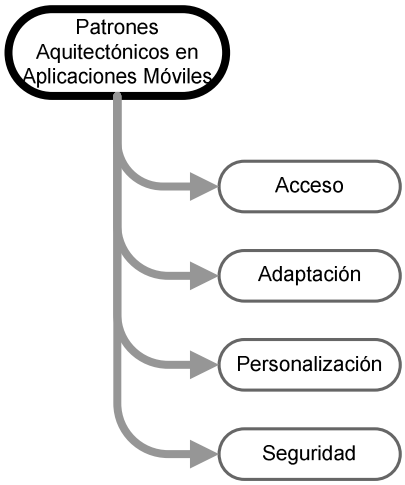

Figura 17. Clasificación de patrones en ambientes móviles.

7.2 Catálogo de patrones

A continuación se presenta un grupo de patrones clasificados según las categorías anteriormente mencionadas.

De cada patrón se detalla el nombre y clasificación, el propósito para el cual fue desarrollado, el problema que aborda, la solución que plantea, su motivación, sus consecuencias y ejemplos de usos.

Cada patrón adjunta un diagrama de la estructura del mismo, con la siguiente notación:

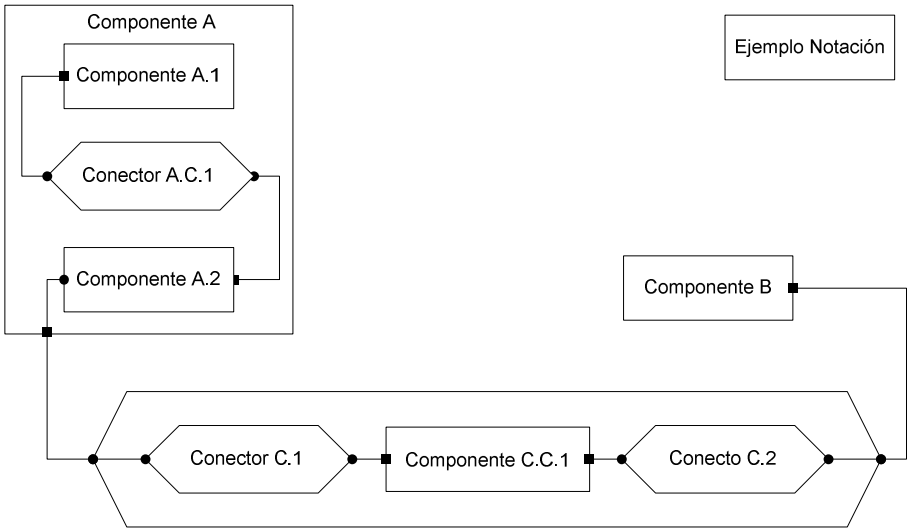

Figura 18. Ejemplo notación de patrones arquitectónicos.

En el diagrama de la Figura 18 se observa que el Componente A se conecta al Componente B por medio de un Conector C. Cada componente representa una unidad de cómputo. Las funcionalidades de los componentes pueden ser accedidas a través de puertos representados por pequeños cuadraditos negros. Los componentes interactúan con otros componentes utilizando conectores que conectan dos o más componentes a través de sus puertos. Se pueden anidar componentes y conectores.

7.2.1 Acceso: sincronización

El objetivo de este patrón es describir mecanismos que permitan mantener actualizada idéntica información en diferentes dispositivos móviles, o no, que poseen una conexión discontinua. Los usuarios consultan y modifican la información. Este patrón se describe teniendo en cuenta las siguientes áreas de aplicación: portales, archivos compartidos y accesos a bases de datos.

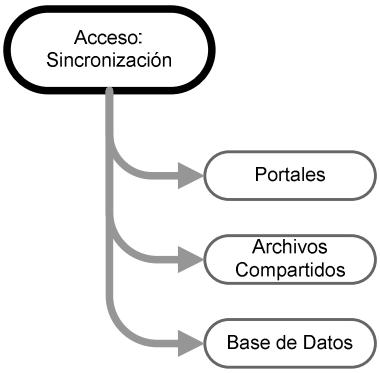

Figura 19. Patrón acceso. Sincronización.

7.2.1.1 Sincronización: portales

Propósito: dotar al cliente móvil de capacidad para navegar sitios Web cuando el dispositivo no fuera capaz de acceder a los mismos.

Problema: el usuario desea acceder a un sitio Web mediante un dispositivo que no le permite hacerlo. La incapacidad de conexión puede deberse a que el dispositivo no cuenta con esta funcionalidad, o debido a que la misma se encuentra temporalmente imposibilitada.

Solución: realizar una copia local de los sitios Web a los cuales desea acceder el cliente. Esta copia deberá ser actualizada cada vez que se pueda establecer una conexión con el sitio Web.

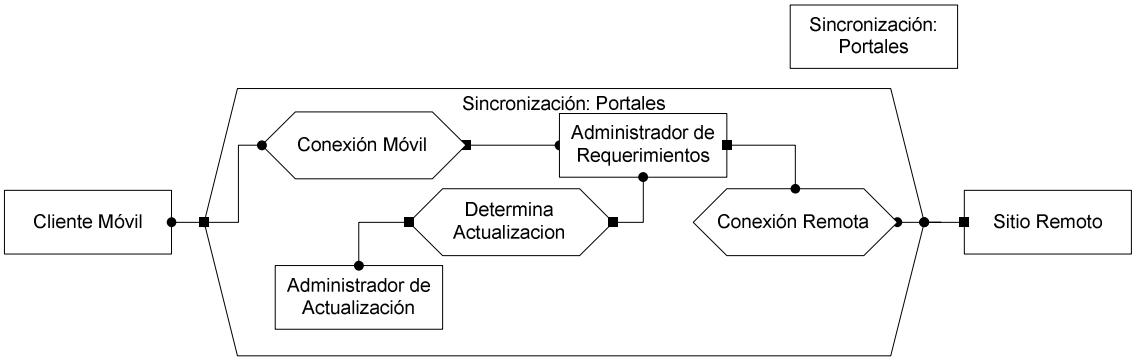

Figura 20. Patrón sincronización: portales.

Motivación: haciendo uso del conector de sincronización el cliente móvil accede al sitio Web sin importar el estado de la conexión. Cuando la conexión al sitio Web remoto se encuentra activa el Conector de Sincronización recibirá los pedidos y se los enviará al Administrador de Actualización, este último

componente será quien decida si el sitio local debe o no ser actualizado. Por último el Administrador de Requerimientos establecerá la conexión con el sitio Web remoto por intermedio del conector Conexión Remota.

En el contexto de una conexión intermitente o nula, la primera tarea que se realiza ante un requerimiento es la sincronización del sitio local con el sitio remoto, de esta forma se asegura que el cliente accederá a una copia actualizada.

Consecuencias: aquellos dispositivos que cuenten con un Conector de Sincronización tendrán la funcionalidad de acceso a un sitio Web. Debido a que la información almacenada en el sitio Web puede cambiar mientras que el usuario no disponga de conexión, puede ocurrir que el Conector de Sincronización brinde información desactualizada. Este conector tampoco es útil en los casos en los que se desea acceder a información generada dinámicamente por procesos que corren en el servidor.

La sincronización puede realizarse a nivel de código, de datos o de contenido.

Usos conocidos: Intellisync® SyncML Server permite sincronizar todo tipo de información entre un repositorio central y los clientes que se conectan a través de diferentes conexiones.

AvantGo es un servicio que brinda a los PDA y Smartphone la capacidad de acceder a sitios Web que poseen un formato determinado ya sea a través de una conexión o mediante una copia previamente descargada.

7.2.1.2 Sincronización: archivos compartidos

Propósito: permitir compartir información almacenada en documentos, incluso de manera concurrente.

Problema: el usuario móvil debe acceder a documentos que se encuentran almacenados en un repositorio central, incluso en aquellos casos en los que la conexión con dicho repositorio no pueda asegurarse.

Solución: se realiza una copia local del árbol de directorios y los documentos en los que el usuario está interesado; una vez realizada la copia, estos documentos se mantienen actualizados mediante sincronizaciones que se llevan a cabo en los momentos en los que se dispone de conexión.

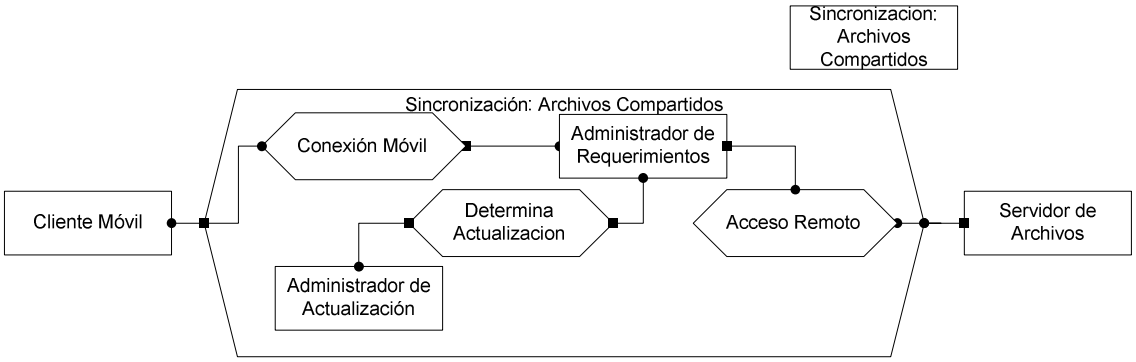

Figura 21. Patrón sincronización: archivos compartidos.

Motivación: las entidades y funciones de este patrón son similares al patrón de sincronización de portales, salvo en el caso del componente Administrador de Actualización que debe realizar la sincronización en ambos sentidos, es decir, debe actualizar el repositorio con las modificaciones realizadas por el usuario en el dispositivo local y actualizar las copias locales de aquellos documentos que hayan sido modificados por otros usuarios.

Consecuencias: el cliente móvil puede acceder a los documentos sin importar si se dispone de una conexión. Por lo general el conector se encuentra integrado al SO. La sincronización del árbol de directorio se puede llevar a cabo en forma completa o incremental, en el primer caso se le da prioridad a la exactitud de los datos almacenados en el servidor y en el cliente, mientras que en el segundo se prioriza el uso eficiente de las conexiones.

Usos conocidos: Windows 2000 Advance Server brinda a los usuarios de dispositivos móviles la capacidad de acceder a los archivos de red encontrándose desconectados de la misma. El usuario puede navegar las carpetas compartidas y unidades de red sin importar el estado de la conexión, una vez restablecida la conexión Synchronization Manager se encarga de actualizar los archivos de red con los cambios que se realizaron durante en período de desconexión.

7.2.1.3 Sincronización: bases de datos

Propósito: brindar a los clientes móviles la posibilidad de acceder a bases de datos remotas sin importar en que lugar se encuentran ni la disponibilidad de una conexión.

Problema: debido a que existen períodos en los que los clientes se encuentran desconectados de la base de datos central, los datos del cliente se desactualizarán y no será posible evitar las modificaciones concurrentes que éstos realicen.

Solución: realizar la sincronización de los datos mediante un componente. Este componente debe tener en cuenta el estado de la conexión y realizar un registro de las modificaciones que han sido aplicadas a la información local para poder llevar a cabo la sincronización.

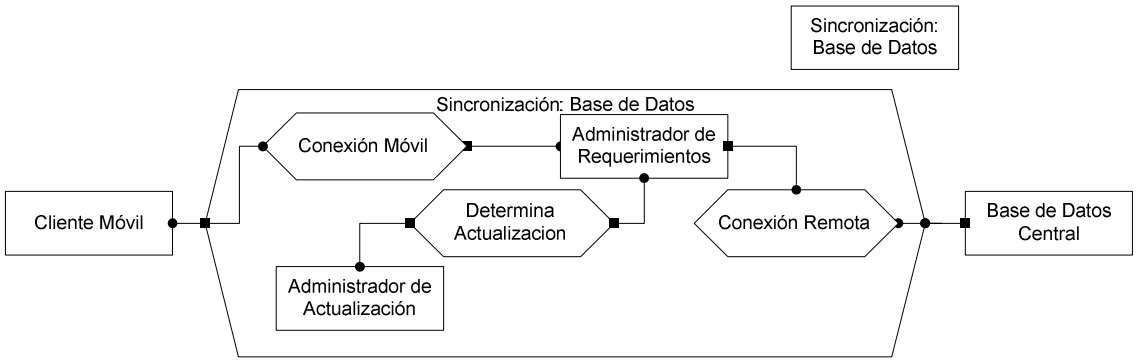

Figura 22. Patrón sincronización: bases de datos.

Motivación: los componentes de este conector son similares a los de los conectores de sincronización para portales y archivos compartidos, sin embargo el componente Administrador de Actualización posee ciertas diferencias con su equivalente en los conectores nombrados anteriormente. Como en el caso de la sincronización de archivos, las actualizaciones deben ser tanto del servidor al cliente, como del cliente al servidor. Sin embargo este conector posee una dificultad adicional ya que las actualizaciones no se tratan simplemente de reemplazar un archivo por otro en su totalidad. El Administrador de Actualización lleva un registro en el cual se incluyen todas las transacciones realizadas sobre la base de datos local que serán luego equiparadas con aquellas modificaciones realizadas en la base de datos central. El Administrador de Actualización se encarga de intercambiar modificaciones con el servidor central y resolver los conflictos que puedan existir debido a las actualizaciones simultáneas de la base de datos local y la base de datos central por parte de diferentes clientes.

Consecuencias: este patrón puede ser utilizado en aquellas situaciones en las que se prioriza la movilidad ante la disponibilidad de datos actualizados.

El usuario debe ser consciente de que puede estar tomando decisiones basándose en información desactualizada que luego será revertida cuando se realice la sincronización con el servidor central.

Usos Conocidos: Microsoft® SQLServer™ 2005 Mobile Edition 3.0 brinda acceso concurrente a una base de datos desde dispositivos móviles. El esquema multihilo de este producto permite sincronizar los datos en segundo plano, mientras el usuario continúa trabajando. Pylon Application Server es un producto que extiende aplicaciones basadas en un dominio Lotus a dispositivos móviles, cuenta con un mecanismo de sincronización que permite acceder a la información tanto en forma conectada como desconectada.

7.2.2 Acceso: emulación

El objetivo de este patrón es brindar al usuario un entorno en el cual pueda continuar accediendo a la información que le interesa sin importar cuál es el estado de la conexión. Este patrón, junto con el patrón de sincronización, permite al usuario independizarse de los problemas de acceso.

7.2.2.1 Emulación

Propósito: hacer transparente al usuario la no disponibilidad de acceso en línea.

Problema: existe una gran cantidad de motivos por los cuales un dispositivo móvil puede perder la conectividad, entre ellos se encuentran la falta de cobertura y los desperfectos técnicos.

Solución: dotar al dispositivo móvil de una copia de la información almacenada localmente, la cual será accedida por medio del emulador cuando se presenten problemas de conectividad. El emulador se encargará de encaminar las solicitudes hacia la copia local.

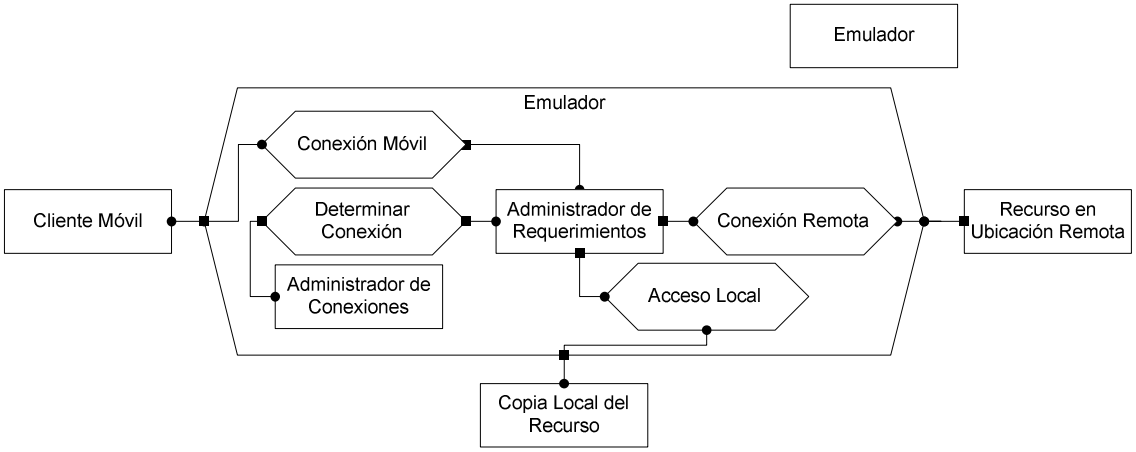

Figura 23. Patrón acceso emulación.

Motivación: el conector emulador determina si los accesos que realiza el cliente se realizarán a la copia local del recurso o al recurso remoto original. Cuando se presenta un pedido de acceso, el Administrador de Requerimientos utiliza el Administrador de Conexiones para consultar el estado de las conexiones y dependiendo de la respuesta encaminará la petición hacia la copia local o el recurso remoto.

El Administrador de Conexiones mantiene un seguimiento del estado de las conexiones.

Consecuencias: el cliente móvil no se ve afectado por las intermitencias que pueden sufrir las conexiones de red. El emulador basa sus decisiones en la información que le brinda el administrador de conexiones.

Usos Conocidos: Windows 2000 Advance Server permite utilizar archivos ubicados de la red aún cuando se está desconectado de la misma. Cuando se pierde la conexión se notifica al usuario de la situación y el mismo puede decidir continuar trabajando normalmente, pues el SO redirecciona sus pedidos a una copia local de sus archivos.

7.2.3 Adaptación: adaptador cliente

Mediante este patrón se garantiza que un servicio ofrecido por un proveedor pueda ser brindado a clientes móviles sin necesidad de ser modificado. Este servicio puede ser un sitio Web, un servidor de archivos o una base de datos, entre otros.

Propósito: adaptar la información a las restricciones del dispositivo móvil.

Problema: la información que brinda un servicio determinado no puede ser presentada o administrada por el dispositivo móvil. Se desea que quien provea el servicio no deba modificar el mismo para poder atender las solicitudes de los clientes móviles.

Solución: dotar al dispositivo móvil de mecanismos que le permitan adaptar la información brindada por un servicio determinado a sus características y capacidades.

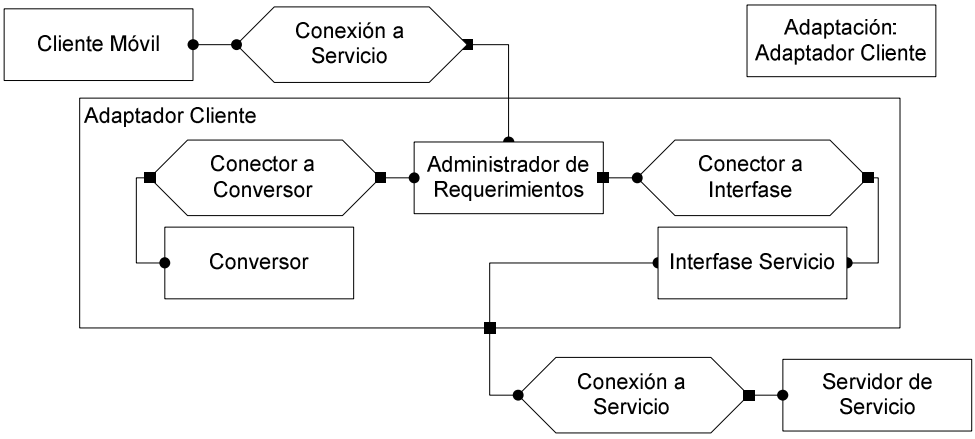

Figura 24. Patrón adaptación: adaptador cliente.

Motivación: el adaptador cliente se encarga de transformar el formato de la información brindada por un servicio a un formato que pueda ser manipulado por el cliente móvil. Cuando se recibe información mediante la interfaz de servicio, el Administrador de Requerimientos encamina dicha información al componente Conversor, el cual haciendo uso de sus conocimientos de las limitaciones del cliente, adapta la información recibida para entregársela al cliente móvil. Esto implica que el componente Conversor debe conocer también las características del proveedor de servicios.

Consecuencias: el proveedor del servicio podrá atender a los clientes móviles sin realizar ningún cambio ni conocer las características de los mismos. Se hace necesario que el dispositivo móvil posea suficiente poder de cómputo para realizar las transformaciones necesarias sobre la información.

El cliente móvil puede utilizar el mismo protocolo de comunicación que utilizan los clientes estándar del servicio, de esta manera los pedidos serán enviados por el administrador de requerimientos directamente al proveedor sin necesidad de intervención por parte del Conversor.

Los conversores deben ser diseñados específicamente para un determinado tipo de cliente móvil, por lo cual se debe contar con tantos conversores como tipos de cliente.

La utilización de componentes Conversor debería disminuir debido al aumento del poder de cómputo, las capacidades de almacenamiento y resoluciones de pantalla que han experimentado los dispositivos móviles.

Usos Conocidos: Microsoft Windows CE 5.0 – Pocket Internet Explorer permite a los dispositivos móviles navegar por la Web sin necesidad de conectarse a través de un Proxy que adapte el contenido de los sitios visitados.

El navegador Opera, en su versión 8.5, basada en la misma versión del navegador para computadoras de escritorio, posee la capacidad de reducir las páginas Web para adaptarlas a pantallas pequeñas.

7.2.4 Adaptación: adaptador servidor

Este patrón brinda la información solicitada por un cliente móvil en un formato que se adapta a las restricciones del mismo. El proveedor del servicio, a diferencia del patrón adaptador cliente, debe conocer todas las características de aquellos clientes a los cuales les brindará servicio.

Propósito: dotar al cliente móvil de la información que solicita en un formato acorde a las características del mismo.

Problema: un cliente móvil que posee diferentes tipos de restricciones, tales como el poder de cómputo, tamaño de pantalla y capacidad de almacenamiento, desea acceder al servicio que brinda un determinado servidor.

Solución: dotar al servidor de mecanismos que permitan adaptar la información que brinda, de manera que la misma sea útil para cada tipo de cliente a los cuales sirve.

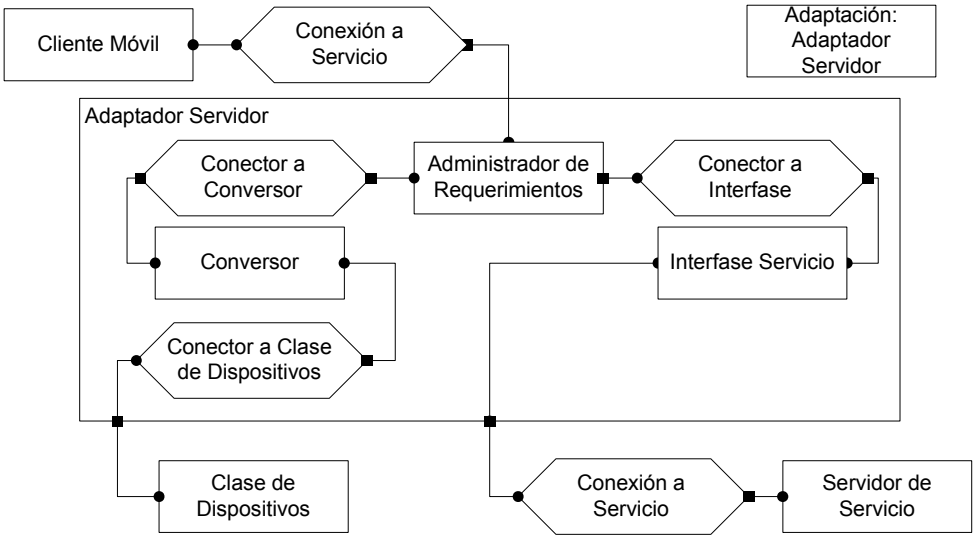

Figura 25. Patrón adaptación: adaptador servidor.

Motivación: el cliente móvil continúa accediendo al proveedor de servicio haciendo uso del conector de conexión, sin embargo, entre este conector y el proveedor del servicio se inserta el componente Adaptador. Dentro de este componente el Administrador de Requerimientos encaminará las solicitudes hacia el Conversor y luego las enviará al servidor a través de la interfaz de servicio. El Conversor obtendrá la información de los diferentes tipos de dispositivos desde un componente externo.

Consecuencias: los dispositivos móviles suelen tener un poder de cómputo restringido, por lo cual trasladar el mecanismo de adaptación desde el dispositivo móvil a un servidor con un poder de cómputo superior sería la solución más adecuada.

El servidor deberá conocer las características y restricciones de todos aquellos dispositivos a los cuales brindará su servicio. Debido a la gran cantidad de dispositivos existentes, la información almacenada puede ser excesiva y requerirá un mantenimiento continuo.

Si el servidor no puede obtener las características del dispositivo a servir se verá imposibilitado de brindar el servicio o sólo podrá brindar un servicio limitado.

Debido a la gran variedad de dispositivos, la tarea de desarrollar un Conversor puede llegar a tener un alto nivel de complejidad.

Usos Conocidos: IBM WebSphere Everyplace Mobile Portal, de la familia de WebSphere Everyplace Service Delivery, permite desarrollar servicios

independientemente de los dispositivos móviles que hagan uso del mismo. Para ello, el contenido de la información se mantiene en un formato que es independiente del dispositivo y mediante el uso de una base de datos que almacena las características de los dispositivos móviles, lleva a cabo la adaptación de la información antes de realizar el envío.

7.2.5 Personalización: entrega personalizada

Este patrón se utiliza en aquellas situaciones en las que se desea acceder a la información según las características del usuario o del contexto en el cual se encuentra.

Entrega personalizada

Propósito: brindar información adaptada a las necesidades del cliente.

Problema: se desea brindar información de acuerdo a determinadas características del cliente, las cuales pueden ser dinámicas o estáticas. Se intenta mejorar la eficiencia de una aplicación e incrementar la usabilidad, personalizando la presentación.

Solución: desarrollar un componente que fuera capaz de adaptar las respuestas del servidor en función de las características del usuario.

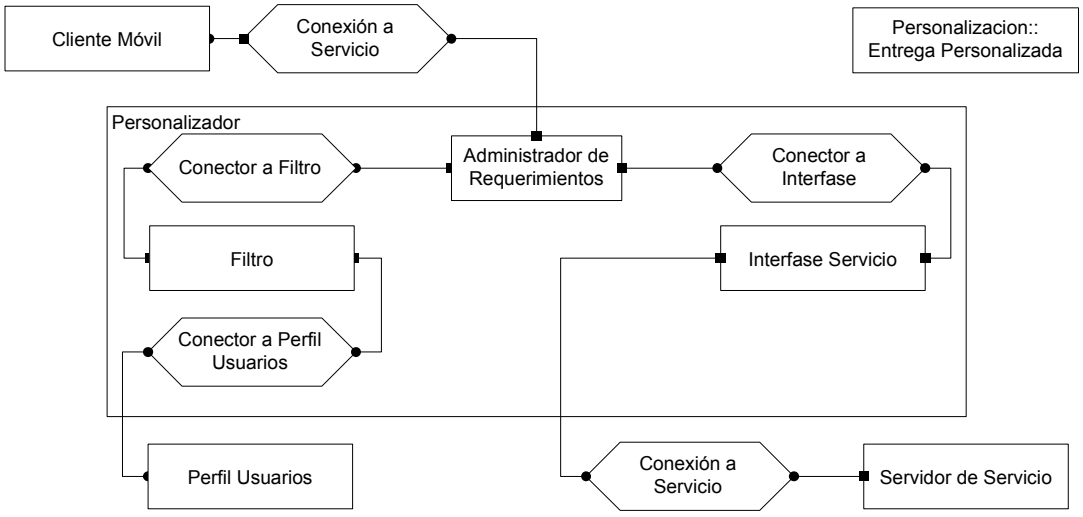

Figura 26. Patrón personalización: entrega personalizada.

Motivación: el Personalizador recibe las solicitudes del cliente a través del Conector de Servicio. Dentro del Personalizador se encuentra el Administrador de Requerimientos que traslada éstos a un componente denominado Filtro que accede a un repositorio externo de perfiles de usuario.

Una vez que el componente Filtro recuperó la información del perfil del usuario, devuelve el requerimiento adaptado de manera que pueda ser enviado al proveedor de servicio. Dentro de la información del perfil del usuario se pueden encontrar datos tales como intereses, roles, hábitos o preferencias, entre otros. Como se dijo anteriormente la información almacenada sobre los perfiles de usuario puede ser estática o dinámica.

Consecuencias: se puede controlar la información que se brinda a cada usuario en cada momento. Mantener la base de datos de los perfiles de usuario puede ser una tarea compleja, debido a la gran cantidad de perfiles que se deben mantener y qué parte de la información del mismo es dinámica. Implementar este patrón puede ser una tarea compleja y costosa. La aplicación que brinda el servicio puede ser una aplicación nueva, una modificada o una que no ha sufrido ningún cambio.

Usos Conocidos: Conference Assistant es una aplicación que provee información a los asistentes basándose en su perfil y ubicación actual. Utiliza una variante de este patrón simplificada.

7.2.6 Interfaz: entrada rápida

Este patrón aborda la problemática que presentan las interfaces de los dispositivos móviles para el ingreso de datos.

Entrada rápida:

Propósito: dotar al dispositivo móvil de un mecanismo que permita el ingreso de información de una forma ágil y eficiente.

Problema: ingresar datos haciendo uso de una interfaz reducida puede ocasionar que este ingreso se vuelva ineficiente, reduciendo de esta forma la usabilidad de la aplicación. Es necesario contar con un componente que mejore la forma en que el usuario interactúa con el dispositivo.

Solución: crear un componente que permita el ingreso de datos por parte del usuario de una forma ágil basándose en la lógica de la aplicación.

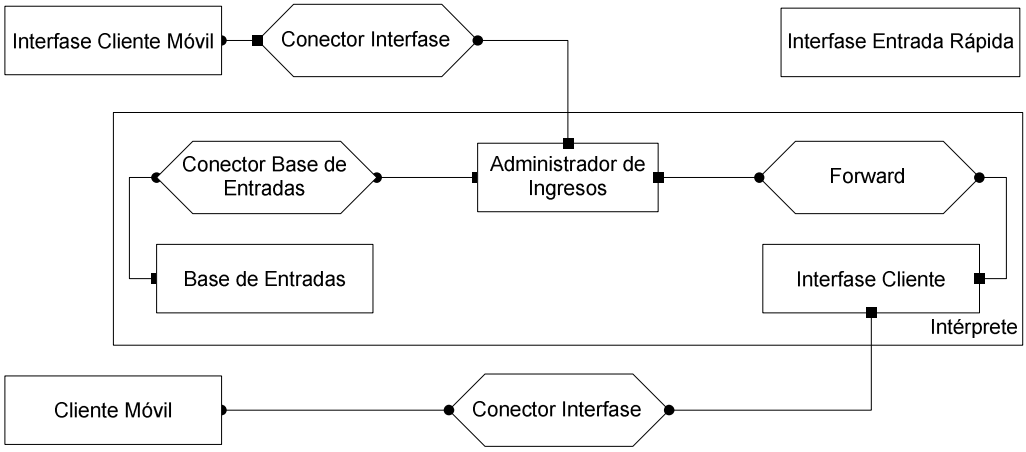

Figura 27. Patrón interfaz: entrada rápida.

Motivación: los datos ingresados por medio de la interfaz del dispositivo son manipulados previamente por el Intérprete antes de enviarlos al cliente móvil. Un ejemplo de esto sería un teléfono celular con un teclado de doce teclas a través del cual se debe ingresar la información. A medida que el usuario ingresa la información, el Administrador de Entradas, consulta una base de datos intentando predecir dicha información. Si esta información es encontrada en la base de datos y confirmada por el usuario, se enviará al cliente móvil, en el caso de no encontrarla, el usuario deberá ingresar todos los caracteres de la misma y antes de que el intérprete la envíe al cliente, registrará dicha información en la base de datos para mejorar predicciones futuras.

Consecuencias: en el caso de diseñar la aplicación mediante un modelo MVC (Model View Controller) este patrón trabajaría sobre el Controller, y si se utiliza un modelo PAC (Presentation Abstraction Control) este patrón trabajaría sobre el control. El componente intérprete es transparente para la aplicación e incrementa la usabilidad y la eficiencia en el ingreso de datos por parte del usuario. Este patrón se puede aplicar también al ingreso de símbolos o gestos a través de una pantalla táctil de un dispositivo móvil.

Usos Conocidos: muchos teléfonos celulares hacen uso del mecanismo T9 que utiliza una implementación de este patrón. TEALScript ™ es un sistema de reconocimiento de texto para PDA. Graffiti ® de Palm permite introducir datos rápidamente mediante un sistema de escritura.

.

CAPÍTULO 8 Introducción a Android

Android es un conjunto de software de código abierto que incluye el SO, middleware y una serie de librerías utilizadas para desarrollar aplicaciones móviles que se adapten a la apariencia y funciones de los dispositivos móviles.

Los dispositivos móviles modernos se han convertido en herramientas poderosas que incorporan cámaras, reproductores de multimedia, GPS y pantallas táctiles. A medida que la tecnología ha evolucionado, los dispositivos móviles han dejado de ser utilizados simplemente para hacer llamadas y es necesario disponer de una plataforma capaz de satisfacer dichas necesidades.

Hasta no hace mucho los teléfonos móviles eran entornos cerrados construidos sobre SO propietarios que requerían herramientas de desarrollo propietarias. A menudo, los teléfonos priorizaban las aplicaciones nativas sobre aquéllas desarrolladas por terceras partes introduciendo una barrera para los desarrolladores de aplicaciones móviles.

En Android tanto las aplicaciones nativas como las desarrolladas por terceros son escritas utilizando las mismas API y ejecutadas de igual manera. Estas API permiten el acceso al hardware, servicios de locación, soporte para servicios en segundo plano, actividades basadas en mapas, bases de datos relacionales, intercambio de mensajes entre dispositivos y gráficos en 2D y 3D.

8.1 Plataforma Android

Google introdujo la plataforma Android para incursionar en el mercado de los dispositivos móviles, describiéndola como la primera plataforma para móviles que es verdaderamente abierta, donde todo el software corre en el dispositivo móvil sin preocuparse de los obstáculos que provienen de la innovación móvil.

Sus aplicaciones se desarrollan en lenguaje Java y se debe contar con una máquina virtual especialmente diseñada para permitir su ejecución en esta plataforma.

Google junto con la Open Handset Alliance son los encargados de la dirección del proyecto Android. Su objetivo es el desarrollo de estándares

abiertos para la telefonía móvil con el fin de incentivar el desarrollo y mejorar la experiencia del usuario.

Android no intenta ser simplemente un SO, sino una plataforma que incorpore todos los elementos necesarios para permitir al desarrollador controlar las funcionalidades ofrecidas por los diferentes dispositivos móviles tales como llamadas, mensajes de texto, cámara, agenda de contactos, diferentes tipos de conexiones, entre otros.

Permite el desarrollo de aplicaciones portables, reutilizables y de rápido desarrollo. De esta manera, estableciendo estándares de desarrollo de las aplicaciones móviles para cualquier dispositivo, intenta terminar con la fragmentación existente hoy en día en el mercado.

Android esté integrado por diferentes componentes, entre los cuales se incluyen:

- Un sistema operativo basado en Linux, que brinda una interfaz de bajo nivel con el hardware, el manejo de memoria y control de procesos optimizado para dispositivos móviles.
- Librerías de código abierto para el desarrollo de aplicaciones, incluyendo SQLite, WebKit y OpenGL.
- La máquina de virtual de Dalvik, junto con todas las librerías, que proveen la funcionalidad de Android para ejecutar y albergar las aplicaciones.
- El entorno de ejecución, diseñado para ser pequeño y eficiente para su uso en dispositivos móviles.
- Un framework de aplicaciones que expone los servicios del sistema a la capa de aplicación, incluyendo el manejador de ventanas, proveedores de contenido, telefonía, entre otros.
- Un framework de interfaz de usuario, utilizado para almacenar y lanzar aplicaciones.
- Aplicaciones preinstaladas, como parte de la plataforma.
- Un kit de desarrollo de software, usado para crear aplicaciones que incluyen las herramientas, plugins y documentación.

Un aspecto importante de Android es su filosofía abierta, la cual asegura que cualquier deficiencia en el diseño de la interfaz de usuario o las

aplicaciones nativas puede ser solucionada escribiendo una extensión o un reemplazo. Android brinda a los desarrolladores la oportunidad de crear aplicaciones diseñadas para verse y funcionar exactamente como se desea.

8.1.1 Aplicaciones nativas de Android

Los teléfonos Android normalmente tienen incorporada una suite de aplicaciones preinstaladas, las cuales incluyen:

- Un cliente de correo electrónico compatible con Gmail, pero no necesariamente limitado a éste.
- Una aplicación de manejo de mensajes SMS.
- Una suite para manejo de información personal, tal como un calendario y listas de usuarios, ambos integrados con los servicios Online de Google.
- Las aplicaciones de Google Maps incluyendo StreetView, vistas satelitales y sistemas de navegación.
- Un Web browser.
- Un cliente de mensajería instantánea para GTalk.
- Un reproductor de música y un visualizador de imágenes.
- Una tienda de aplicaciones Android para descargar aplicaciones desarrolladas por terceros.

8.1.2 Pila de software Android

Los elementos que componen la pila de software de Android se muestran en la Figura 28 y se detallan a continuación.

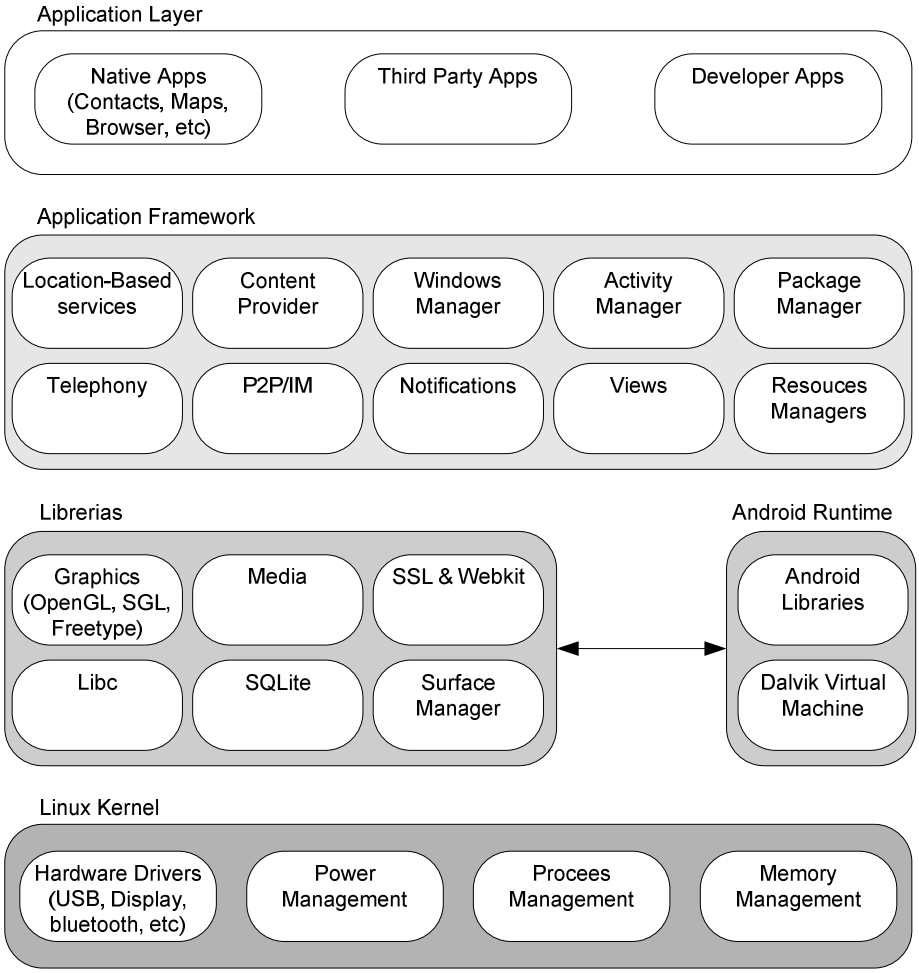

Figura 28. Pila de software de Android.

Un kernel de Linux y una colección de librerías de C/C++ son expuestas a través de un framework de aplicaciones que proveen servicios para el entorno de ejecución y las aplicaciones.

- *Kernel de Linux:* los servicios del núcleo (drivers de hardware, manejo de procesador y memoria, manejo de red, seguridad y energía) son manejados por un kernel de Linux 2.6. El Kernel también provee una capa de abstracción entre el hardware y el resto de la pila de software.
- *Librerías:* corriendo sobre el kernel, Android incluye varias librerías de núcleo escritas en C/C++, entre las que se encuentran libc, y SSL, así como también:
    - Librerías para la reproducción de audio y video.
    - Manejo de pantalla.

  - Librerías gráficas que incluyen SGL (Scalable Graphics Library) y OpenGL para gráficos 2D y 3D.
  - SQLite para soporte nativo de base de datos.
  - SSL y WebKit para navegación segura.
- *Android Run Time:* lo que realmente hace a un teléfono Android más que la implementación móvil de Linux es el entorno de ejecución Android. Incluyendo las librerías de núcleo y la máquina virtual Dalvik, el entorno de ejecución de Android es el motor que impulsa las aplicaciones y junto con las librerías forman la base para el framework de aplicaciones.
  - *Librerías de núcleo:* mientras que el desarrollo en Android se realiza en Java, Dalvik no es una máquina virtual de Java. Las librerías de núcleo de Android proveen la mayor parte de la funcionalidad disponible en las librerías de núcleo de Java, así como también en las librerías específicas de Android.
  - *Máquina virtual de Dalvik:* es una máquina virtual que ha sido optimizada para asegurar que un dispositivo puede manejar múltiples instancias de manera efectiva. Se basa en un kernel de Linux para el manejo de hilos y memoria de bajo nivel.
- *Framework de aplicaciones:* esta estructura provee las clases utilizadas para crear las aplicaciones de Android. También brinda un mecanismo de abstracción para el acceso al hardware y manejo de interfaz del usuario y recursos de aplicaciones.
- *Capa de aplicación:* todas las aplicaciones, tanto nativas como desarrolladas por terceros, son construidas en la capa de aplicación utilizando las mismas librerías de la API. La capa de aplicación corre dentro del entorno de ejecución de Android utilizando las clases y servicios disponibles desde el framework de aplicaciones.

8.1.3 Máquina virtual Dalvik

Uno de los componentes principales de Android es la máquina virtual de Dalvik. En lugar de usar una máquina virtual de Java tradicional, como puede ser Java ME, Android utiliza su propia máquina virtual personalizada, diseñada

para asegurar que múltiples instancias se ejecutan de manera eficiente en el mismo dispositivo.

Las Máquina virtual Dalvik usa el kernel Linux del dispositivo para manejar aquellas funcionalidades de bajo nivel incluyendo la seguridad, hilos, y manejo de memoria y procesos. También es posible escribir aplicaciones en C y C++ que corran directamente sobre el SO Linux, si bien en la mayoría de los casos no hay razón para hacerlo.

El acceso al hardware de Android y a los servicios del sistema es manejado utilizando Dalvik como nivel intermedio. Mediante el uso de una máquina virtual para albergar aplicaciones en ejecución, los desarrolladores tienen una capa de abstracción que les asegura que nunca tendrán que preocuparse por una implementación de hardware particular.

La máquina virtual Dalvik ejecuta archivos ejecutables Dalvik, los cuales se encuentran en un formato optimizado para asegurar un uso mínimo de memoria. Los ejecutables .dex son creados transformando las clases Java compiladas mediante herramientas suministradas por el kit de desarrollo de software.

8.1.4 Arquitectura de las aplicaciones Android

La arquitectura de Android alienta el concepto de reuso de componentes, permitiendo publicar y compartir actividades, servicios e información con otras aplicaciones. Los accesos a información de otras aplicaciones son manejados por restricciones de seguridad definidas.

El mecanismo que permite reemplazar el manejador de contactos o la interfaz de discado, también permite exponer los componentes de una aplicación para permitir que otros desarrolladores creen nuevas interfaces de usuario y extensiones de funcionalidad sobre ella.

Los siguientes servicios son fundamentales en todas las aplicaciones Android, formando la estructura que se utilizará para el software.

- *Manejador de activities:* controla el ciclo de vida de las activities, incluyendo el manejo de la pila de activities.
- *Vistas:* son utilizadas para construir las interfaces de usuario en las activities.

- *Manejador de notificaciones*: provee un mecanismo consistente y cómodo para enviar notificaciones al usuario.
- *Proveedores de contenido:* permite a las aplicaciones compartir datos con otras aplicaciones.
- *Manejador de recursos:* permite que los recursos tales como strings y gráficos sean manejados en forma externa.

8.1.5 SDK Android

El kit de desarrollo de software de Android incluye todos los componentes necesarios para desarrollar, probar y depurar las aplicaciones. Este kit incluye los siguientes componentes:

- *API de Android:* el núcleo del kit de desarrollo son las librerías API que brindan al desarrollador el acceso a la pila de software del mismo. Estas librerías son las mismas que se utilizan para desarrollar las aplicaciones nativas de Android.
- *Herramientas de desarrollo:* para transformar el código fuente en una aplicación Android ejecutable, el SDK incluye varias herramientas de desarrollo que permiten compilar y depurar las aplicaciones.
- *Emulador de Android:* emula un dispositivo Android completamente interactivo con diferentes skins. Utilizando el emulador es posible reproducir la apariencia y comportamiento de la aplicación en un dispositivo real. Todas las aplicaciones Android se ejecutan dentro de la máquina virtual Dalvik, por lo que el software emulador presenta un entorno excelente para la prueba de aplicaciones en Android y, de hecho, al ser independiente del hardware, brinda un mejor ambiente de pruebas que cualquier implementación en hardware.
- *Documentación completa:* el SDK incluye una extensa documentación que detalla de manera exacta el contenido de cada paquete y clases y cómo utilizarlos. Además incluye documentos que explican cómo comenzar a desarrollar aplicaciones en Android y explican detalladamente los fundamentos del desarrollo en esta plataforma.
- *Código ejemplo:* el SDK incluye una sección de ejemplos de aplicaciones que muestran algunos de los posibles usos de Android y

pequeños programas que señalan cómo utilizar las características de las API.

- *Soporte en línea:* a pesar de ser una tecnología reciente, Android ha generado una importante comunidad de desarrolladores.

8.1.5.1 Características del SDK de Android

Las API que provee Android son el principal atractivo de su ambiente de desarrollo.

Dentro de las características más importantes de Android se encuentran las siguientes:

- No requiere licencias, permisos de distribución o matrículas de desarrollo.
- Acceso a hardware de WI-FI.
- Mediante el uso de las redes telefónicas GSM, EDGE (Enhanced Data Rates for Global Evolution) y 3G permite hacer y recibir llamadas, enviar y recibir mensajes, o enviar y recibir información a través de móviles.
- API para servicios basados en ubicación tales como GPS.
- Acceso al hardware multimedia tales como la cámara, el micrófono y la reproducción o grabación de sonidos.
- API para uso de hardware de acelerómetro y brújula.
- Intercambio de mensajes IPC (InterProcess Comunication).
- Almacenamiento de datos compartidos.
- Web Browser open source.
- Soporte a aplicaciones que poseen Maps Controls como parte de sus interfaces.
- Soporte para transferencias peer to peer usando Google Talk.
- Soporte para gráficos 2D y 3D.
- Librerías multimedia para reproducción y grabación de una variedad de formatos de audio, video o imágenes.
- Framework de aplicación que alienta el reuso de componentes y el reemplazo de aplicaciones nativas.

8.1.5.2 Acceso al hardware incluyendo cámara, GPS y acelerómetro

Android incluye librerías para simplificar los desarrollos que involucren el hardware del dispositivo. Esto asegura que no sea necesario crear implementaciones específicas de una aplicación para diferentes dispositivos, con lo cual se pueden desarrollar aplicaciones que funcionen de la manera esperada en aquellos dispositivos que dan soporte a la pila de software de Android.

El kit de desarrollo de software de Android incluye API para hardware de ubicación tales como GPS, cámaras, conexiones de red, WI-FI, Bluetooth, acelerómetros, pantallas táctiles y manejo de energía.

8.1.5.3 Google Maps, geocodificación y servicios basados en ubicación

El soporte nativo de mapas permite crear aplicaciones basadas en mapas que aprovechan la movilidad de los dispositivos Android.

Android permite crear actividades que incluyen Google Maps como parte de su interfaz con acceso completo a mapas.

Los servicios basados en locación de Android hacen uso de tecnologías tales como GPS y tecnología de ubicación basadas en celdas GSM. Estos servicios aseguran una abstracción de la tecnología de detección de ubicación, permitiendo especificar requerimientos mínimos, en lugar de elegir una en particular. Esto también significa que las aplicaciones basadas en la ubicación funcionarán sin importar qué tecnología soporte el dispositivo.

Para combinar los mapas con las ubicaciones, Android incluye una API para geocodificación que permite encontrar las coordenadas para una dirección y la dirección para una posición en el mapa.

8.1.5.4 Servicios en segundo plano

Android brinda soporte a aplicaciones y servicios diseñados para correr de manera invisible en segundo plano. Los dispositivos móviles modernos son por naturaleza dispositivos multifunción. Sin embargo, su tamaño limitado de pantalla hace que sólo una aplicación interactiva pueda ser visible en un momento determinado. Las plataformas que no dan soporte a la ejecución en segundo plano limitan la visibilidad de las aplicaciones que no necesitan atención constante.

Los servicios en segundo plano hacen posible crear aplicaciones invisibles que realizan procesos automáticos sin intervención directa del usuario. La ejecución en segundo plano permite a las aplicaciones ser controladas por eventos y soportar actualizaciones de manera periódica.

8.1.5.5 SQLite Database para almacenamiento y recuperación de datos

El almacenamiento y recuperación de datos de manera rápida y eficiente es esencial para los dispositivos cuya capacidad de almacenamiento se encuentra limitada.

Android provee una base de datos relacional, ágil y liviana para aquellas aplicaciones que utilizan SQLite. Las aplicaciones pueden tomar ventaja del motor de base de datos relacional para almacenar datos de manera segura y eficiente.

8.1.5.6 Datos compartidos y comunicación entre aplicaciones

Android incluye tres mecanismos para transmitir información desde las aplicaciones: notificaciones, intents (abstracción de una operación a realizar) y proveedores de contenidos.

Las notificaciones son maneras estándar en las cuales un dispositivo móvil alerta al usuario. Utilizando la API se pueden disparar alertas sonoras, hacer vibrar al dispositivo y parpadear el LED (Light-Emitting Diode) del mismo, así como controlar el estado de los íconos en la barra de notificaciones.

Los intents brindan un mecanismo para el envío de mensajes entre aplicaciones o dentro de una misma aplicación. El uso de intents permite realizar un broadcast de una acción deseada a lo largo del sistema para que sea manejado por otras aplicaciones.

Los proveedores de contenidos son una manera de brindar a las aplicaciones acceso a bases de datos privadas. Los almacenamientos de datos para las aplicaciones nativas, como puede ser el administrador de contactos, son expuestos como proveedores de contenidos, por lo cual se pueden crear aplicaciones que lean o modifiquen dichos contenidos.

8.1.5.7 Servicios P2P con Google Talk

Permiten enviar mensajes estructurados desde una aplicación a cualquier otro móvil Android utilizando el servicio de comunicaciones P2P (Peer to Peer) de Android.

El servicio P2P de Android utiliza versiones especiales de XMPP (eXchange Message and Presence Protocol). Se basa en el servicio de mensajería instantánea Google Talk, crea una conexión socket persistente entre un dispositivo móvil y cualquier otro dispositivo móvil Android Online que asegura comunicaciones con baja latencia y rápidos tiempos de respuesta.

Cuando se encuentre disponible se puede hacer uso del servicio de Google Talk para el intercambio de mensajes o una interfaz para enviar datos entre instancias de una aplicación en dispositivos separados.

El servicio P2P también ofrece notificaciones de presencias las cuales son utilizadas si un contacto se encuentra en línea. Mientras que el servicio P2P es muy atractivo por sí mismo, también puede utilizarse junto con otras características de Android. Por ejemplo, un servicio en segundo plano que transmite ubicaciones entre amigos y una aplicación basada en mapas que las muestra o brinda alertas cuando un amigo se encuentra cerca.

8.1.5.8 Amplio soporte multimedia y gráficos 2D y 3D

Los dispositivos móviles se han convertido en dispositivos multimedia gracias a los mayores tamaños y resoluciones de sus pantallas.

Para hacer uso del hardware Android provee librerías gráficas para dibujos en 2D y OpenGL para gráficos 3D.

Android también ofrece librerías para el manejo de imágenes, video y archivos de audio, incluyendo los formatos MPEG4, H.264, MP3, AAC, AMR, JPG, PNG y GIF.

8.1.5.9 Administración optimizada de memoria y procesos

El manejo de memoria y procesos de Android es un tanto inusual. Al igual que Java y .Net utiliza su propio entorno de ejecución y máquina virtual para manejar la memoria de las aplicaciones. A diferencia de cualquiera de estos frameworks, el entorno de ejecución de Android también maneja el ciclo de vida de los procesos. Android asegura el funcionamiento de las aplicaciones

deteniendo y matando procesos cuando se necesite liberar recursos para aplicaciones de mayor prioridad.

En este contexto la prioridad se determina dependiendo en la aplicación con la cual el usuario está interactuando.

8.2 Componentes de las aplicaciones Android

Las aplicaciones Android están integradas por componentes débilmente acoplados, vinculados utilizando el manifiesto del proyecto que describe cada componente y cómo éstos interactúan entre sí.

Existen seis componentes que pueden ser utilizados para construir aplicaciones:

- *Activities:* cada pantalla de la aplicación será una extensión de la clase Activity. Las activities utilizan vistas para crear interfaces gráficas de usuario que muestren información y respondan a las acciones de éste. En comparación con una aplicación de escritorio, una activity es equivalente a un formulario.
- *Services:* estos componentes se ejecutan y realizan su trabajo de manera invisible, actualizando las fuentes de datos, las activities visibles y disparando notificaciones. Son utilizados para realizar procesamientos que necesitan mantenerse en ejecución incluso cuando las activities de la aplicación no están activas o visibles.
- *Content Provider:* estos componentes son utilizados para manipular y compartir datos y son la forma preferida para compartir información a lo largo de la aplicación. Esto significa que se puede configurar un content provider para permitir el acceso desde otras aplicaciones y usar los de otras aplicaciones para acceder a la información que ellas almacenan. Un ejemplo de estos componentes es aquél mediante el cual se puede acceder a la información de los contactos.
- *Intents:* utilizando estos componentes se pueden transmitir mensajes a lo largo del sistema o a una activity o service, indicando la intención de que una acción sea realizada.
- *Broadcast Receiver:* creando y registrando un broadcast receiver, las aplicaciones pueden escuchar difusiones de intents que concuerden con

un determinado filtro. Estos componentes lanzarán aplicaciones para responder a los intents recibidos, siendo muy útiles en aquellas aplicaciones que son orientadas a eventos.

- *Notifications:* las notificaciones permiten enviar indicaciones a los usuarios sin interrupción o sacando el foco de las activities. Son utilizadas para mantener la atención del usuario desde un service o broadcast receiver. Un ejemplo de notification sería una alerta mediante una luz intermitente que se produce con la llegada de un mensaje.

8.3 Archivo manifiesto de Android

Cada proyecto Android incluye un archivo manifiesto, AndroidManifest.xml, almacenado en la raíz de la jerarquía del proyecto. Este archivo permite definir la estructura y metadatos de las aplicaciones y sus componentes.

Incluye nodos para cada uno de los componentes (activies, services, content provider y broadcast receiver) que conforman la aplicación, y utilizando filtros de intenciones y permisos, determina cómo interactúan entre ellos y con otras aplicaciones.

8.3.1 Formato del manifiesto

A continuación se ejemplifica la estructura acotada del archivo:

```
<!ELEMENT manifest
(uses-permission*, permission*, instrumentation*, application?)>
<!ELEMENT application
(activity+, receiver*, service*, provider*)>
<!ELEMENT activity
(intent-filter*, metadata*)>
<!ELEMENT receiver
(intent-filter*, metadata*)>
<!ELEMENT service
(intent-filter*, metadata*)>
<!ELEMENT provider
(metadata*)>
<!ELEMENT intent-filter
(action+, category+, data*)>
```

Figura 29. DTD (Document Type Definitions) reducido de un archivo AndroidManifest.xml.

Dentro de la declaración del manifiesto se pueden encontrar los siguientes elementos:

- <*manifest*>*:* el nodo raíz, bajo el cual se declararán todos los contenidos del manifiesto.
- <*uses-permission*>*:* declara requisitos de seguridad para que el paquete pueda ser ejecutado correctamente. Estos permisos son mostrados al usuario durante la instalación para determinar si los mismos son otorgados o no.
- <*permission*>*:* define permisos utilizados para restringir a los componentes de la aplicación el acceso a ciertos servicios de Android.
- <*instrumentation*>*:* declara componentes utilizados para monitorear la funcionalidad de éste u otro paquete.
- <*application*>*:* elemento raíz que enumera los componentes básicos que constituyen la aplicación; estos atributos determinan su configuración. El archivo manifiesto puede contener sólo un nodo aplicación.
- <*activity*>*:* declara un componente activity presente en la aplicación, encargado de representar una acción concreta y generalmente asociado a una interfaz de usuario. Deberá haber tantos elementos <activity> como componentes activity haya en la aplicación. Se deben incluir el activity principal y cualquier otra pantalla o diálogo que puedan ser mostrados. Intentar lanzar una activity que no está definida dentro del manifiesto lanzará una excepción en tiempo de ejecución.
    - <*intent-filter*>*:* declara lo que se denomina un Intent Filter, encargado de describir cuándo y dónde puede ejecutarse la activity dentro de la cual está definida. Mediante este elemento se especifica qué acciones puede manejar la presente actividad, es decir, qué elemento intent puede atender. Permite además la inclusión de varios atributos que determinan su funcionamiento.
    - <*action*>***:*** acción soportada por la activity.
    - <*category*>**:** categoría de la activity.
    - <*data*>***:*** datos aceptados por la activity.
    - <*meta-data*>***:*** metadatos sobre la activity definidos por el desarrollador.
- <*receiver*>***:*** elemento que representa a un componente Broadcast Intent *Receiver*, cuya misión es lanzar una acción en respuesta a un evento.

Deberá haber tantos elementos <receiver> declarados como componentes Broadcast Intent Receiver haya en la aplicación.

- *<service>:* este elemento se corresponde con un componente service presente en la aplicación, encargado de ejecutar una acción en background. Deberán aparecer tantos elementos <service> como componentes service haya en la aplicación. Este componente también puede contener etiquetas intent-filters.
- <provider>: este elemento corresponde a un componente Content Provider, utilizado en la aplicación para almacenar y compartir datos. Debe haber tantos elementos <provider> declarados como componentes Content Provider haya en la aplicación.

8.4 Ciclo de vida de las aplicaciones de Android

A diferencia de los ambientes tradicionales, las aplicaciones Android no tienen control sobre su ciclo de vida. En su lugar, los componentes de la aplicación deben prestar atención a los cambios de estado de la aplicación y responder de acuerdo a los mismos, estando preparados para una terminación no esperada.

Cada aplicación Android es ejecutada en su propio proceso corriendo una instancia individual de la máquina Dalvik, aportando beneficios en cuestiones básicas como seguridad, gestión de memoria, o la ocupación del procesador del dispositivo móvil. El manejo de procesos y memoria de cada aplicación es realizado exclusivamente por el entorno de ejecución de Android.

Android maneja sus recursos, haciendo lo que fuese necesario para asegurar que el dispositivo mantiene la agilidad de respuesta adecuada. Esto significa que los procesos (y la aplicación que cada cual alberga) serán finalizadas abruptamente, sin advertencia alguna, si es necesario liberar recursos para aplicaciones con mayor prioridad (generalmente aquéllas con las que el usuario se encuentra interactuando).

8.4.1 Prioridad de aplicaciones y estados de los procesos

El orden en el que los procesos son finalizados para recuperar recursos queda determinado por la prioridad de la aplicación que cada cual alberga. La

prioridad de la aplicación se determina a partir del componente con mayor prioridad que la misma posee.

Cuando dos aplicaciones poseen la misma prioridad se finaliza el proceso que ha tenido menor prioridad durante mayor tiempo. La prioridad de los procesos se ve afectada por la dependencia entre éstos; si una aplicación tiene dependencias sobre un servicio o un proveedor de contenidos suministrados por otra aplicación, la aplicación secundaria tiene al menos una mayor prioridad que aquélla a la que brinda soporte.

Todas las aplicaciones en Android se mantendrán corriendo y en memoria hasta que el sistema necesite sus recursos para otras aplicaciones.

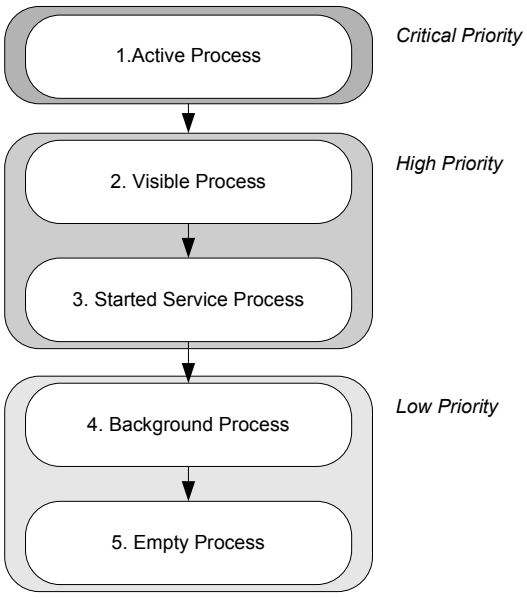

Figura 30. Ciclo de vida de los procesos en Android.

Es importante estructurar las aplicaciones correctamente para asegurar que las prioridades son apropiadas para realizar las tareas deseadas. Si no se realiza dicha planificación una aplicación puede ser finalizada en medio de un proceso importante.

Procesos activos

Son aquellas aplicaciones en las que sus componentes se encuentran actualmente interactuando con el usuario. Android intenta mantener estos componentes con el mejor nivel de respuesta manejando los recursos que el dispositivo dispone. Estos procesos generalmente son pocos y sólo pueden ser finalizados como última opción con el objetivo de obtener recursos.

Dentro de los procesos activos se pueden encontrar:

- Actividades en estado activo, es decir se encuentran visibles y respondiendo a las peticiones del usuario.
- Actividades, servicios o receptores de difusiones que se encuentran ejecutando el manejador de un evento.
- Servicios que se encuentran ejecutando los manejadores de los eventos onStart, onCreate u onDestroy.

Procesos visibles

Éstos son aquellos procesos visibles pero no activos, que albergan actividades visibles. Como su nombre lo indica, las actividades visibles se encuentran visibles, pero no se encuentran en primer plano o respondiendo a eventos del usuario. Esto sucede cuando una actividad se encuentra parcialmente oculta por una actividad transparente o que no ocupa la totalidad de la pantalla. Por lo general los procesos visibles son pocos y sólo serán finalizados en circunstancias extremas, para permitir al proceso activo continuar su ejecución.

Servicios

Son procesos que albergan servicios. Dan soporte a procesos que deberían continuar sin la necesidad de una interfaz visible. Ya que los servicios no interactúan directamente con el usuario, reciben una prioridad ligeramente menor que la de las actividades visibles. Se consideran procesos en primer plano y no serán finalizados a menos que se necesiten recursos para procesos activos o visibles.

Procesos en segundo plano

Los procesos que albergan actividades que no son visibles y no poseen ningún servicio que haya sido iniciado son considerados procesos en segundo plano. Generalmente existe una gran cantidad de procesos en segundo plano que Android puede finalizar utilizando una política determinada para obtener recursos para los procesos en primer plano.

Procesos vacíos

Para mejorar la performance del sistema, Android a menudo mantiene las aplicaciones en memoria luego de que han alcanzado el fin de su ciclo de vida. Mantiene esta cache para mejorar el tiempo necesario para iniciar las

aplicaciones cuando éstas son iniciadas nuevamente. Estos procesos son los primeros en ser finalizados cuando se necesitan recursos.

8.5 Seguridad en Android

Gran parte de las medidas de seguridad del sistema y las aplicaciones se basa en los estándares de Linux, en cuyo núcleo se basa Android.

Se brinda a cada aplicación un entorno seguro de ejecución, conocido como sandbox o caja de arena. Ninguna aplicación puede realizar operaciones que afecten negativamente el comportamiento de otras aplicaciones o del sistema en sí mismo.

Ejemplos de operaciones no permitidas son la lectura o escritura de archivos privados del usuario u otras aplicaciones o el acceso a recursos no permitidos como la red. La única forma de poder ejecutar estas operaciones restringidas es mediante una declaración explícita de un permiso que autorice a la aplicación a realizar determinada tarea normalmente prohibida.

Las aplicaciones en Android deben estar firmadas digitalmente mediante un certificado. Este cerificado puede utilizarse para establecer relaciones de confianza entre aplicaciones.

Para dotar a una aplicación de determinado permiso es necesario incluir en el archivo manifiesto uno o más elementos <uses-permission>. Para permitir que una aplicación monitorice los SMS entrantes, por ejemplo, se deben incluir en el archivo “AndroidManifest.xml” los siguientes <uses-permission>:

```
<manifest xmlns:Android="http://schemas.Android.com/apk/res/Android"
package="com.Android.app.myapp" >
      <uses-permission Android:name="Android.permission.RECEIVE_SMS" />
</manifest>
```

Código 1. Archivo Manifiesto. Permiso de envío de mensajes de texto.

Otros permisos que se pueden conceder mediante el archivo manifiesto son el uso de Wi-Fi, Bluetooth, llamadas telefónicas, cámara e Internet, entre otros.

El elemento <uses-permission> puede contener atributos que ayudan a definir de manera más detallada los permisos otorgados a una aplicación determinada, entre los cuales se encuentran:

- *Android:name:* nombre del permiso a otorgar. Estos nombres se encuentran listados en la clase Android.Manifest.permission.

- *Android:label:* nombre conocido por el usuario.
- *Android:permissionGroup*: grupos de permisos listados en la clase Android.Manifest.permission_group. Ejemplos de estos grupos son ACCOUNTS (cuentas de Google), COST_MONEY (acciones que generan un costo) o PHONE_CALLS (acciones vinculadas con llamadas), entre otros.
- *Android:protectionLevel:* establece el nivel de riesgo del permiso y cómo el sistema otorga o no el permiso a la aplicación. Puede tomar valores entre 0 y 3.
- *Android:description:* texto descriptivo del permiso.
- *Android:icon:* imagen asociada al permiso

8.6 Gestión de la información

En Android los repositorios de datos son privados y accedidos por la aplicación a la cual pertenecen. Más allá de ello, estos datos pueden ser compartidos con otras aplicaciones, para lo cual Android brinda mecanismos que posibilitan el acceso.

8.6.1 Preferencias de usuario

En Android las aplicaciones pueden tener asociados ciertos valores para cada usuario que permiten adaptar la aplicación a las preferencias del mismo. Estos valores sólo podrán ser accedidos dentro de un mismo paquete, no se permite compartir estos valores con otras aplicaciones.

Para compartir estas preferencias entre los componentes de una aplicación es necesario agruparlas y asignarle un nombre a este grupo. Luego, los componentes podrán acceder a las preferencias mediante el uso de la función Context.getSharedPreferences(). En caso de no ser necesario compartir las preferencias con el resto de los componentes, una actividad puede acceder a las preferencias llamando a la función Activity.getPreferences().

8.6.2 Archivo

Android posee los métodos Context.openFileInput() y Context.openFileOutput() para abrir, leer y escribir archivos. Los archivos deben pertenecer a una aplicación, por lo cual un archivo no puede compartirse con otras aplicaciones.

Si el archivo es estático puede ser incorporado a la aplicación como un recurso más en el momento de ser ésta compilada. Los archivos se ubican dentro de la carpeta de recursos, más precisamente en "\res\raw\nombreArchivo.extension". Los archivos incluídos en esta carpeta formarán parte del paquete y podrán ser leídos haciendo uso del método Resources.openRawResource (R.raw.nombreArchivo).

8.6.3 Bases de datos

Mediante la librería SQLite, Android permite la creación, administración y acceso a base de datos relacionales. Estas bases de datos pueden ser accedidas desde aplicaciones como si fuesen objetos pertenecientes a las mismas, mediante las clases contenidas en el paquete Android.database.sqlite.

Las bases de datos serán accesibles desde los demás componentes de la misma aplicación, pero no fuera de ésta.

8.6.4 Acceso por red

Android brinda soporte a la comunicación entre dispositivos a través de una red disponible. Este soporte se brinda haciendo uso de los paquetes java.net y Android.net.

8.6.5 Content Provider

Android posee mecanismos para acceder a información compartida entre aplicaciones, para lo cual se utilizan proveedores de contenidos, que es la única forma habilitada de acceso a información ajena a una aplicación.

Se incluyen por defecto algunos proveedores de contenidos para publicar y compartir datos entre aplicaciones tales como información de contactos, imágenes, fotografías, videos, entre otros. Estos Content Provider se encuentran en el paquete Android.provider. Los diseñadores pueden crear sus propios proveedores de contenidos.

A cada Proveedor de Contenido se le asocia una URI (Uniform Resource Identifier) que lo identifica de manera única y a través de la cual puede ser accedido. Para acceder a la información de los contactos del dispositivo se puede utilizar un proveedor de contenido identificado por la URI "content://contacts/people/".

El siguiente código ejemplifica el acceso a la información de contactos de un dispositivo utilizando la URI que identifica al proveedor de contenidos de los contactos. Entre los métodos utilizados para hacer una consulta a través de una URI se encuentra managedQuery() de la clase Activity, al cual se le debe proveer la URI que se desea consultar, los campos que se quieren obtener, y datos, como pueden ser las cláusulas WHERE u ORDER BY. Este método retorna un conjunto de filas que se pueden acceder a través del objeto Cursor de la clase Android.database.

```
// Datos a recuperar
String[] datos_a_recuperar = new String[] {
Contacts.People.NAME,
Contacts.People.NUMBER,
Contacts.People._ID
};
// URI utilizada para acceder a los contactos
Uri contactos = "content://contacts/people/";
// Lanzar consulta
Cursor cursor = managedQuery( contactos, datos_a_recuperar, null, null,
Contactos.People.NAME + " ASC");
```

Código 2. Ejemplo de uso de Content Provider.

8.7 Entorno de desarrollo Eclipse

Eclipse es un entorno de desarrollo de código abierto muy utilizado para desarrollos en Java. Posee versiones para cada una de las plataformas de desarrollo soportadas por Android (Windows, Mac OS y Linux). Estas versiones pueden ser descargadas desde el sitio Web de Eclipse (http://www.eclipse.org/downloads/). La instalación consiste simplemente en descomprimir el archivo descargado dentro de una nueva carpeta y luego correr el ejecutable. Cuando se inicia Eclipse por primera vez se crea un nuevo workspace para el desarrollo en Android.

Se ha desarrollado un plug-in especial para este entorno de desarrollo que simplifica la creación de proyectos e integra Eclipse con el emulador de Android y las herramientas de depuración.

El uso de Eclipse con el plug-in ADT (Android Developer Tools) ofrece ventajas significativas.

8.7.1 Plug – in ADT

El plug-in ADT simplifica el desarrollo en Android integrando las herramientas, incluyendo el emulador y un convertidor de .class a .dex, dentro del entorno de desarrollo. Si bien el uso de este plug-in no es obligatorio, simplifica y acelera el proceso de creación, prueba y depuración de las aplicaciones.

El plug-in ADT integra en Eclipse las siguientes características:

- Un asistente para proyectos Android que simplifica la creación de un nuevo proyecto, e incluye un template de una simple aplicación.
- Editores que ayudan a crear, editar y validar los recursos XML.
- Construcción automatizada de proyectos, conversión a los ejecutables de Android .dex, empaquetador para la creación de paquetes .apk propios de Android, e instalación de paquetes dentro de la máquina virtual Dalvik.
- El emulador de Android, incluye el control de la apariencia del emulador, configuración de conectividad y la posibilidad de simular llamadas y mensajes entrantes.
- El Servicio de Monitoreo de Debug Dalvik (DDMS) incluye port forwarding, visores de pila, montículo e hilos, detalles de procesos y facilidades para la captura de pantallas.
- Acceso al sistema de archivo del emulador o el dispositivo permitiendo navegar el árbol de carpetas y transferir archivos.
- Depuración en tiempo de ejecución que permite fijar breakpoints y ver la pila de llamadas.
- Logueo de todas las consolas de salidas de Android/Dalvik.

CAPÍTULO 9 Desarrollo de la aplicación “Mis Contactos”

En este Capítulo se presenta el diseño de una aplicación móvil en la plataforma Android, llamada “Mis Contactos”.

La aplicación es un localizador de contactos, mediante el cual los usuarios informan su ubicación geográfica a un servidor y recogen del mismo las ubicaciones de sus contactos. La información recibida se visualiza a través de mapas y se brindan determinadas funcionalidades relacionadas con los contactos, tales como el envío de mails, SMS o llamadas telefónicas.

Mis Contactos utiliza las principales funcionalidades y componentes que provee Android. Tiene una interfaz gráfica basada en activities, hace uso de una base de datos y servicios ejecutándose en segundo plano para mantener la información actualizada, interactúa mediante el uso de conexiones a Internet con el servicio Maps de Google y hace uso del GPS incorporado en el dispositivo para obtener las coordenadas del mismo e informarlas a un servidor.

El servidor es un servlet que se ejecuta en la plataforma App Engine de Google. Esta plataforma brinda al servlet, entre otras funcionalidades, la capacidad de hacer uso de objetos persistentes y mecanismos de autenticación.

En la actualidad existen aplicaciones, Latitude de Google por ejemplo, que permiten visualizar ubicaciones de una lista de contactos que estén adheridos al servicio. El objetivo de Mis Contactos es brindar un servicio similar a Latitude que posea nuevas funcionalidades no presentes en esta aplicación al día de hoy.

9.1 Funcionalidades

Tanto la aplicación Mis Contactos como el servidor cuentan con diferentes funcionalidades que se detallan a continuación:

9.1.1 Funciones de la aplicación

- *De Configuración:*
  - *Configurar usuario:* configura los datos del usuario de la aplicación.
  - *Configurar estilo de mapa:* establece la manera en que se muestra el mapa (vista satélite, calles, zoom).
  - Configurar zoom: aumenta o disminuye el zoom sobre el mapa.
- *Gestión de Contactos:*
  - *Listar contactos:* visualiza la lista de contactos.
  - *Gestionar solicitudes de amistad*: envía, recepciona, confirma o rechaza solicitudes de amistad.
  - *Bloquear o Desbloquear contactos*: oculta o muestra la ubicación del usuario a un contacto.
  - *Eliminar un contacto:* quita un contacto de la lista de contactos.
  - *Selección de un contacto:* selecciona el contacto anterior al actual, el siguiente, o uno en particular, centrando el mapa en la ubicación del mismo.
- *Funcionalidades asociadas a un contacto:*
  - *Enviar correo electrónico*: envía un email a un contacto a través de la cuenta de correo configurada en el dispositivo.
  - *Enviar SMS*: envía un mensaje de texto a un contacto.
  - *Llamar a un contacto*: realiza una llamada telefónica a un contacto.
  - *Mostrar información del contacto*: visualiza información sobre la ubicación del contacto.

9.1.2 Funciones del servidor

- *Actualizar la ubicación de un contacto:* recibe y almacena la ubicación de un usuario.
- *Responder a solicitudes de ubicación de contactos:* arma documentos XML con las ubicaciones de los contactos de un usuario.
- *Gestionar solicitudes de amistad:* agrega, confirma o rechaza solicitudes de amistad de usuarios a otros contactos.

- *Eliminar, bloquear o desbloquear contactos de un usuario:* elimina, bloquea o desbloquea contactos de un usuario.

9.2 Protocolo de intercambio de información

La aplicación Mis contactos y el servidor se comunican de manera periódica y bajo demanda mientras exista una conexión a Internet. Esta comunicación se utiliza para el intercambio de información mediante documentos XML bajo el protocolo que se detalla a continuación:

- *Intercambio de ubicaciones*: cada cierto período de tiempo la aplicación envía al servidor una petición de ubicaciones de sus contactos, la que lleva adosada la posición actual del usuario.
    - *Solicitud cliente – petición ubicaciones*

```
<solicitud_ubicacion>
   <nombre>Nombre de usuario</nombre>
   <telefono>Número de teléfono del usuario</telefono>
   <email>Correo electrónico del usuario</email>
   <latitud>Latitud del usuario</latitud>
   <longitud>Longitud del usuario</longitud>
</solicitud_ubicacion>
```

Código 3. Protocolo XML. Petición de ubicaciones.

    - *Respuesta servidor – ubicaciones contactos*

```
<contactos>
  <contacto>
    <nombre>Nombre de contacto</nombre>
    <telefono>Número de teléfono del contacto</telefono>
    <email>Correo electrónico del contacto</email>
    <fecha>Fecha de actualización de la información</fecha>
    <latitud>Latitud del contacto</latitud>
    <longitud>Longitud del contacto</longitud>
    <bloqueado>Estado del contacto</bloqueado>
  </contacto>
  <contacto>
  ……….
  </contacto>
</contactos>
```

Código 4. Protocolo XML. Respuesta a solicitud de ubicación.

- *Gestión de solicitudes de amistad*: los usuarios de la aplicación pueden realizar peticiones de solicitudes de amistad, verificar las recibidas y confirmar o rechazar una en particular.
    - *Solicitud cliente – solicitud amistad*

```
<solicitud_amistad>
```

```
   <de>Correo electrónico del solicitante</de>
   <para>Correo electrónico del contacto solicitado</para>
</solicitud_amistad>
```

Código 5. Protocolo XML. Solicitud de amistad.

- *Solicitud cliente – solicitud peticiones de amistad recibidas*

```
<solicitud_amistad_rec>
   <email>Correo electrónico del solicitante</email>
</solicitud_amistad_rec>
```

Código 6. Protocolo XML. Solicitud de peticiones de amistad.

- *Respuesta servidor - solicitudes de amistad recibidas*

```
<solicitudes_amistad_rec>
   <solicitud_amistad>
      <email>Correo electrónico del solicitante</email>
      <nombre>Nombre del solicitante</nombre>
   </solicitud_amistad>
   <solicitud_amistad>
   ……….
   </solicitud_amistad>
</solicitudes_amistad_rec>
```

Código 7. Protocolo XML. Respuesta de solicitudes de amistad.

- *Cliente solicita – confirma amistad*

```
<confirmacion_amistad>
   <de>Correo electrónico del solicitante</de>
   <para>Correo electrónico de quien confirma la solicitud</para>
</confirmacion_amistad>
```

Código 8. Protocolo XML. Confirmación de amistad.

- *Cliente solicita – rechaza amistad*

```
<rechazo_amistad>
   <de>Correo electrónico del solicitante</de>
   <para>Correo electrónico de quien rechaza la solicitud</para>
</rechazo_amistad>
```

Código 9. Protocolo XML. Rechazo de amistad.

- *Gestión de contactos*: los usuarios de la aplicación pueden eliminar, bloquear o desbloquear contactos mediante el intercambio de los siguientes mensajes.
  - *Cliente solicita – elimina contacto*

```
<eliminacion_contacto>
   <de>Correo electrónico del usuario que elimina</de>
   <para>Correo electrónico del contacto eliminado</para>
</eliminacion_contacto>
```

Código 10. Protocolo XML. Eliminar contacto.

- *Cliente Solicita – bloquea o desbloquea contacto*

```
<bloquear_contacto>
   <de>Correo electrónico del usuario que bloquea/desbloquea</de>
   <para>Correo electrónico del contacto
bloqueado/desbloqueado</para>
   <bloqueado>true/false </bloqueado>
</bloquear_contacto>
```

Código 11. Protocolo XML. Bloqueo o desbloqueo de contacto.

9.3 Almacenamiento de información

Para la persistencia de los datos se utilizan objetos persistentes en el servidor y una base de datos local en la aplicación que es consultada cuando no se dispone de una conexión a Internet.

9.3.1 Base de datos local

La aplicación Mis contactos cuenta con una base de datos SQLite que está albergada en el dispositivo móvil. La misma dispone de una única tabla denominada tContacto que almacena la información de los contactos que posee un usuario. El script que crea la tabla se muestra en el segmento de Código 12.

```
CREATE TABLE IF NOT EXISTS tContacto
   (email TEXT PRIMARY KEY,
   latitud INTEGER NOT NULL,
   nombre TEXT,
   telefono TEXT,
   longitud INTEGER NOT NULL,
   fecha INTEGER NOT NULL,
   bloqueado BOOL NOT NULL)
```

Código 12. Script de creación de tabla tContacto.

9.3.2 Objetos persistentes en el servidor

Como ya se mencionó se utiliza como hosting del servidor el servicio App Engine de Google que permite a las aplicaciones que alberga disponer de objetos persistentes. La aplicación dispone de una única clase de objetos persistentes denominada Contacto que se detalla en el segmento de Código 13.

```
package com.Android.miscontactosservidor.modelo;

import java.util.HashSet;
import java.util.Set;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
```

```
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;
/*
 * Clase de objetos persistentes, cada instancia representa un
 * contacto
 * @author Ioana y Hugo
 */
@PersistenceCapable(identityType=IdentityType.APPLICATION)
public class Contacto {
   //Campos persistentes
   @PrimaryKey
   @Persistent
   private String mEmail;
   private Long mFecha;
   private String mNombre;
   private String mTelefono;
   private Integer mLatitud;
   private Integer mLongitud;
   @Persistent
   java.util.Set<String> mContactos; //lista de contactos que posee
el usuario
   @Persistent
   java.util.Set<String> mSolicitudes; //lista de contactos que
solicitaron amistad al usuario
   @Persistent
   java.util.Set<String> mBloqueados; //lista de contactos que
bloqueados

   /*****************************************************************/
   /*          Contacto                                             */
   /*****************************************************************/
   /*
    * Contructor
    * @param pNombre - nombre del contacto
    * @param pTelefono - teléfono del contacto
    * @param pEmail - email del contacto
    * @param pFecha - fecha en la cual se actualizo la ubicación del
contacto
    * @param pLatitud - latitud donde se encuentra el contacto
    * @param pLongitud - longitud donde se encuentra el contacto
    */
   public Contacto(String pNombre, String pTelefono, String pEmail,
Long pFecha, Integer pLatitud, Integer pLongitud) {
      super();
      mNombre = pNombre;
      mTelefono= pTelefono;
      mEmail = pEmail;
      mFecha = pFecha;
      mLatitud = pLatitud;
      mLongitud = pLongitud;
      mContactos=new HashSet<String>();
      mSolicitudes=new HashSet<String>();
      mBloqueados=new HashSet<String>();
   }

}
```

Código 13. Clase de objetos persistentes.

9.4 Modelo de clases

Con el objetivo de dar una visión general de cómo se implementó la solución se presenta a continuación el modelo de clases de la aplicación y del servlet.

9.4.1 Modelo de clases de la aplicación

Las clases de la aplicación se resumen brevemente a continuación:

- *ConexionHTTP*: permite establecer una conexión con el servidor a través de la cual se envían peticiones al mismo y se reciben sus respuestas. Esta funcionalidad se encuentra implementada en el método *conectar*.
- *ConfiguracionMapa*: extiende de la clase Activity y permite configurar la manera en la cual se muestra el mapa.
- *ConfiguracionUsuario*: extiende de la clase Activity y permite configurar los datos del usuario de la aplicación.
- *Contacto*: representa un contacto; posee métodos para crearlo o modificar sus atributos.
- *Contactos*: almacena los contactos que posee el usuario de la aplicación; sus métodos permiten agregar, eliminar y modificar contactos, guardarlos en la base de datos local y obtener o fijar cuál es el actual.
- *ContactosList*: extiende de la clase ListActivity y su función es desplegar en pantalla la lista de contactos del usuario. Mediante esta lista el usuario podrá seleccionar, eliminar, bloquear o desbloquear a alguno de ellos.
- *DialogoItimizedOverlay*: permite gestionar las acciones que realiza el usuario sobre los overlays mostrados en el mapa.
- *DialogoOverlayView*: define la manera en que se muestra la información de un contacto en el mapa.
- *GestorXML*: posee métodos estáticos que arman y leen los diferentes mensajes XML que intercambia la aplicación con el servidor.
- *Global*: almacena las variables globales de la aplicación.

- *MisContactos*: extiende de Application, se encarga de inicializar las variables globales así como de la autenticación de la cuenta Google configurada en el dispositivo. Es la responsable de guardar los datos de la aplicación en la base de datos interna cuando finaliza la ejecución de la aplicación o cuando existe el riesgo de que Android lo haga por falta de recursos.
- *MisContactosBD*: gestiona la base de datos SQLite de la aplicación.
- *MisContactosItemizedOverlay*: extiende de DialogoItemizedOverlay y se encarga de gestionar los overlay que serán mostrados en el mapa.
- *MisContactosMapa*: extiende de MapActivity y es la interfaz principal de la aplicación.
- *GPSListener*: implementa la interfaz LocationListener y su método *onLocationChange* es llamado cada vez que se produce un cambio de ubicación del usuario.
- *MiReceiver*: extiende de BroadcastReceiver y se encarga de monitorear los cambios que se puedan producir en el estado de la conexión a Internet.
- *MisContactosOverlayItem*: representa un contacto en el mapa guardando la ubicación e información del mismo.
- *ProcesadorXML*: extiende de la clase DefaultHandler y se encarga de actualizar la información de los contactos procesando los documentos XML recibidos desde el servidor.
- *ServicioActualizacion*: extiende de la clase Service y se encarga de informar periódicamente la ubicación del usuario al servidor y recibir las ubicaciones actualizadas de sus contactos.
- *SolicitudContacto*: extiende de Activity y es la interfaz mediante la cual el usuario envía solicitudes de amistad a otros contactos.
- *SolicitudContactoList*: extiende de ListActivity y permite al usuario visualizar las solicitudes de amistad recibidas para poder confirmarlas o rechazarlas.

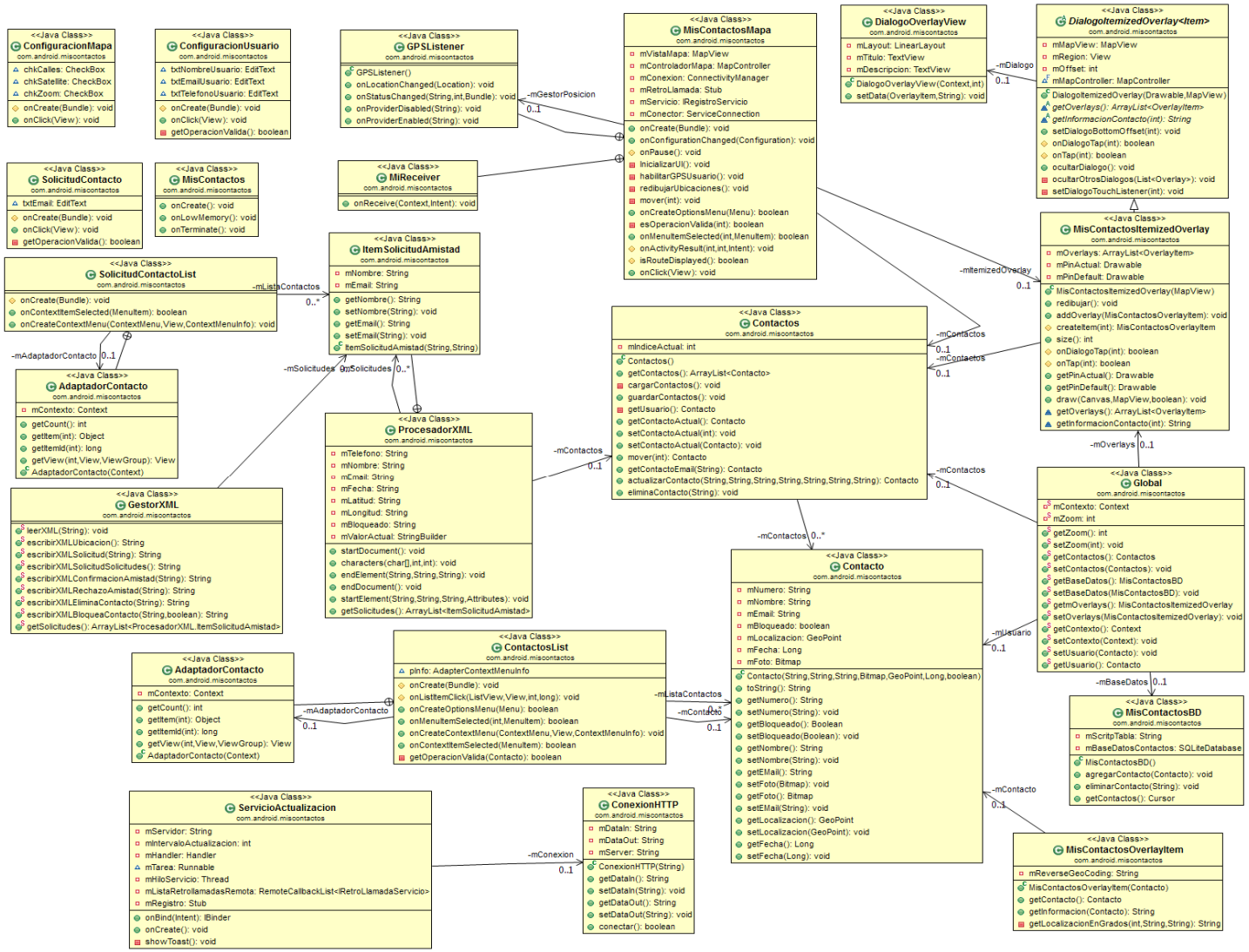

Figura 31. Modelo de clases de Mis Contactos.

9.4.2 Modelo de clases del servidor

- *GestorPersistencia*: clase singleton que contiene una instancia del la clase PersistenceManagerFactory que permite manipular objetos persistentes.
- *GestorXML*: posee métodos estáticos que arman y leen los diferentes mensajes XML que envían los usuarios al servidor.
- *MisContactosServidorServlet*: extiende de la clase HttpServlet, es la clase principal del servidor y se encarga de recibir las solicitudes de los clientes y enviarles sus respectivas respuestas.
- *ProcesadorXML*: extiende de la clase DefaultHandler y se encarga de actualizar la información de los usuarios procesando los documentos XML recibidos desde los clientes.
- *Contacto*: describe el modelo que representa a un contacto en el almacén de objetos persistentes.

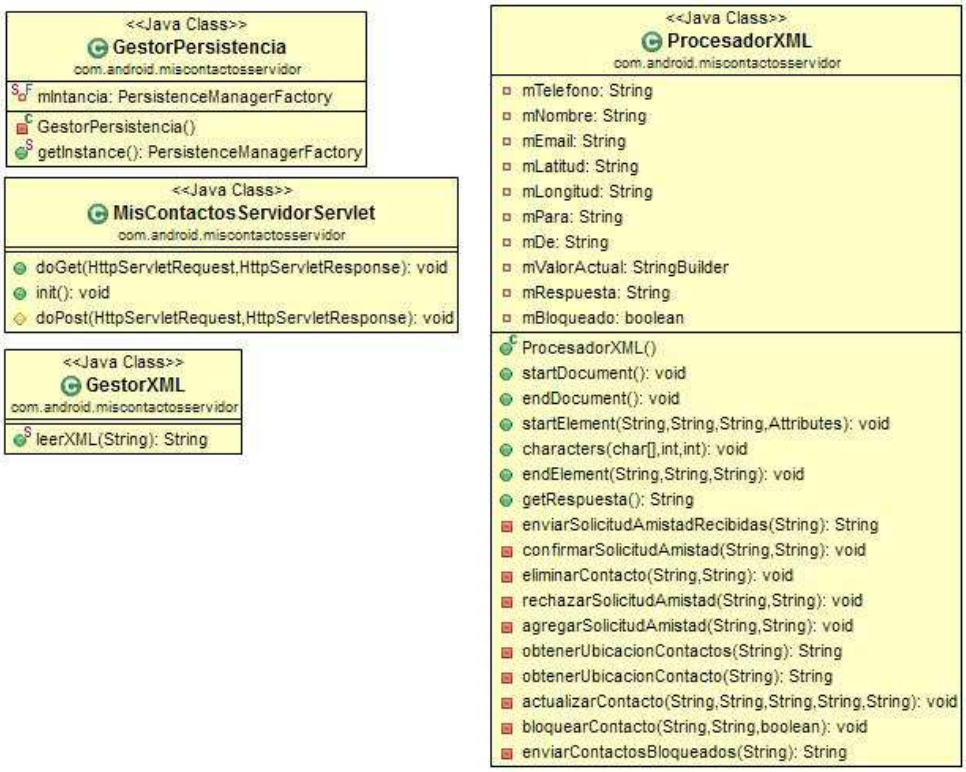

Figura 32. Modelo de clases del servidor.

9.5 Desarrollo e implementación

A continuación se describe la manera en que la aplicación fue desarrollada, explicándose con mayor detalle aquellas características de Android que fueron utilizadas para la implementación.

9.5.1 Entorno de desarrollo

Como entorno de desarrollo para la aplicación se utilizó Eclipse cuya última versión se puede bajar del sitio http://www.eclipse.org/downloads/.

Luego de la instalación de Eclipse se instaló el SKD de Android. Desde el sitio Web http://developer.Android.com/sdk/index.html se puede descargar tanto la última versión así como también sus versiones previas.

Para poder desarrollar en Android desde Eclipse fue necesario instalar el plug-in ADT de Android. Para esto desde *Software Update* del entorno de desarrollo se agregó desde *Available Software* el siguiente link http://dl-ssl.google.com/Android/eclipse/ y se prosiguió con la descarga e instalación de *Android DDMS* y *Android Developers Tools.*

Una vez instaladas todas estas herramientas se pudo comenzar con el desarrollo de la aplicación.

9.5.2 Utilización de variables globales

La aplicación utiliza variables globales que necesitan ser inicializadas sólo una vez en la aplicación y son accedidas desde diferentes clases de la aplicación.

Estas variables están declaradas en la clase Global y son inicializadas desde la clase MisContactos que extiende de Application, donde su método onCreate() comienza la ejecución de la aplicación.

El método onCreate() de las activities puede ejecutarse más de una vez, ya sea por el cambio de alguna propiedad que obliga a redibujarla o porque Android detenga la ejecución de la activity para obtener recursos. Si las variables globales se inicializarán en el método onCreate() de las activities, serían inicializadas varias veces a lo largo del ciclo de vida de la aplicación. El comportamiento descripto hace que este método no sea apropiado para la inicialización de variables globales.

En la aplicación Mis Contactos las variables globales son: el gestor de base de datos (MisContacosDB), la lista de contactos (Contactos) y el contexto de la aplicación (Context).

```
package com.Android.miscontactos;
import Android.app.Application;

public class MisContactos extends Application {
   @Override
   public void onCreate() {
      super.onCreate();
      Global.setContexto(this.getApplicationContext());
      Global.setBaseDatos(new MisContactosBD());
      Global.setContactos(new Contactos());
   }
   ..................
}
```

Código 14. Inicialización de variables globales.

9.5.3 Autenticación App Engine con una cuenta Google

Google ofrece un método de autenticación para aplicaciones que utilizan App Engine denominado Cuentas de Google, que es el sistema de acceso unificado de Google mediante cuentas de correo electrónico válidas.

Los dispositivos móviles que cuenten con SO Android, en la mayoría de los casos disponen de una cuenta Gmail registrada, por lo que Cuentas de Google fue el método de autenticación con el servidor seleccionado para la aplicación Mis Contactos.

La aplicación hace uso de una cuenta Google registrada en el dispositivo móvil de la cual se deben obtener sus credenciales para generar un token que permita la autenticación.

El método AuthenticationAppEngine() de la clase MisContactos es quien realiza la autenticación de la aplicación con el servidor.

La clase AccountManager proporciona acceso a un registro centralizado de cuentas del usuario. Para poder manipular las credenciales de una cuenta de correo electrónico se necesita una autorización por parte del usuario. El intent que solicita la aprobación de la utilización de credenciales es AccountManager.KEY_INTENT. Una vez lograda la aprobación, las credenciales son utilizadas para obtener un token mediante la sentencia authTokenBundle.getString(AccountManager.KEY_AUTHTOKEN) y éste es enviado al servidor para verificar que se corresponde con una cuenta válida.

```
private void AutenticationAppEngine() throws AccountsException,
AuthenticationException{

      AccountManager accountManager =
AccountManager.get(getApplicationContext());
      //recupera las cuentas googles registradas en el dispositivo
      Account[] accounts =
accountManager.getAccountsByType("com.google");

      if (accounts == null || accounts.length == 0)
         throw new AccountsException("no existen cuentas de tipo
'com.google' en el dispositivo");

      try {
           mAccount = accounts[0];

           AccountManagerFuture<Bundle> accountManagerFuture =
accountManager.getAuthToken(mAccount, "ah", true, null,null);
           Bundle authTokenBundle =
accountManagerFuture.getResult();

           //verifica si hay permisos para utilizar los token de
autenticaciòn

if(authTokenBundle.containsKey(AccountManager.KEY_INTENT)) {
              //no hay permiso lanza intent de solicitud de permiso
                   Intent requiredInteraction = (Intent)
authTokenBundle.get(AccountManager.KEY_INTENT);
                   startActivity(requiredInteraction);
                   return;
           }

           //recupera token de autenticación
           String authToken =
authTokenBundle.getString(AccountManager.KEY_AUTHTOKEN);
           if (authToken == null)
             throw new AuthenticationException("no se recupero el
token");


mHttpClient.getParams().setBooleanParameter(ClientPNames.HANDLE_REDI
RECTS, false);

         mCookieUrl = gaeAppLoginUrl + "?continue=" +
URLEncoder.encode(gaeAppServletUrl,"UTF-8") + "&auth=" +
URLEncoder.encode(authToken,"UTF-8");

         HttpGet http_get = new HttpGet(mCookieUrl);
         //envia una solicitud con token de autenticacion para saber
si es un token válido
         HttpResponse response = mHttpClient.execute(http_get);

         if (response.getStatusLine().getStatusCode() != 302)
             throw new AuthenticationException("La respuesta de
autenticación no fue una redirección");

         boolean vExisteCookie = false;
         for(Cookie cookie :
mHttpClient.getCookieStore().getCookies()) {
            if(cookie.getName().equals("ACSID"))
```

```
                vExisteCookie = true;
            }

            if (!vExisteCookie)
                throw new AuthenticationException("No se encontro la
cookie de autenticación en el cookie store");


            } catch (OperationCanceledException e) {
              throw new AccountsException("No se encontro el token de
autenticación en la cuenta 'com.google'", e);
            } catch (AuthenticatorException e) {
              throw new AccountsException("No se encontro el token de
autenticación en la cuenta 'com.google'", e);
            } catch (IOException e) {
              throw new AccountsException("No se encontro el token de
autenticación en la cuenta 'com.google'", e);
            }
    }
```

Código 15. Autenticación App Engine con cuenta Google.

Para poder utilizar cuentas registradas en el dispositivo y recuperar sus credenciales, es necesario habilitar los siguientes permisos en el archivo manifiesto:

```
<uses-permission Android:name="Android.permission.GET_ACCOUNTS"/>
<uses-permission Android:name="Android.permission.MANAGE_ACCOUNTS"/>
<uses-permission Android:name="Android.permission.USE_CREDENTIALS"/>
```

Código 16. Archivo Manifiesto. Permisos de utilización de cuentas y credenciales.

9.5.4 Activity, ListActivity, MapActivity

La aplicación cuenta con diferentes interfaces o pantallas de usuario con las cuales el usuario interactúa para poder acceder a las funcionalidades de la misma. Cada una de estas interfaces corresponde a una extensión de la clase Activity. Las activities utilizan vistas (View) para presentar la interfaz gráfica, que en el caso de la aplicación Mis Contactos son definidas en archivos XML ubicados en la carpeta layout del proyecto. El método setContentView() es utilizado para establecer la interfaz de la activity, indicando el documento XML correspondiente a ésta. El segmento de Código 17 muestra un documento XML que resume el diseño de la interfaz principal de la aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
   xmlns:Android="http://schemas.Android.com/apk/res/Android"
   Android:id="@+id/frame"    >
   <com.google.Android.maps.MapView Android:id="@+id/mapa"
      Android:layout_width="fill_parent"
      Android:layout_height="fill_parent"
```

```
    Android:apiKey="0wphla4-fZygnlwv_KcQs9sHHWKwPgr7Ocij_Hg" />
  <LinearLayout Android:id="@+id/controles">
    <ImageButton Android:id="@+id/btnAnterior"
      Android:src="@drawable/anterior" />
    <ImageButton Android:id="@+id/btnZoomOut"
      Android:src="@drawable/zoom_out" />
    <ImageButton Android:id="@+id/btnZoomIn"
      Android:src="@drawable/zoom_in" />
    <ImageButton Android:id="@+id/btnSiguiente"
      Android:src="@drawable/siguiente" />
  </LinearLayout>
</FrameLayout>
```

Código 17. XML interfaz de pantalla principal.

El segmento de Código 17 representa la interfaz mostrada a continuación.

Figura 33. Interfaz pantalla principal.

Existen diferentes clases de activities que son extensiones de la clase Activity. Para la interfaz principal de la aplicación se utiliza un MapActivity que encapsula toda la funcionalidad para mostrar el control MapView que representa el mapa donde se ubican los contactos a mostrar.

Para la interfaz de la lista de contactos del usuario y las solicitudes de amistad recibidas por éste se utiliza un ListActivity que permite mostrar una lista de elementos y soporta la selección de éstos.

Por último es necesario mencionar que para que las activities se puedan mostrar al usuario es necesario declararlas dentro del tag application en el archivo manifiesto.

```
  <application Android:name=".MisContactos"
Android:label="@string/app_name" >
……
        <activity Android:name=".ContactosList"></activity>
        <activity Android:name=".EditorSMS"></activity>
        <activity Android:name=".ConfiguracionMapa"></activity>
        <activity Android:name=".ConfiguracionUsuario"></activity>
        <activity Android:name=".SolicitudContacto"></activity>
        <activity Android:name=".SolicitudContactoList"></activity>
……
</application>
```

Código 18. Archivo manifiesto. Declaración de activities.

9.5.4.1 Menú principal y menú contextual

Para acceder a las funciones brindadas por las activities se utilizan diferentes tipos de menú.

La interfaz de cualquier tipo de menú es declarada en documentos XML. El segmento de Código 19 muestra un documento XML que describe un menú principal en una activity.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:Android="http://schemas.Android.com/apk/res/Android">
    <item Android:id="@+id/mnuContactos"
          Android:icon="@drawable/agenda"
          Android:title="@string/menuContactos" />
    <item Android:id="@+id/mnuConfigurar"
          Android:icon="@drawable/mapa"
          Android:title="@string/menuMapa" />
    <item Android:id="@+id/mnuUsuario"
          Android:icon="@drawable/configurar"
          Android:title="@string/menuUsuario" />
    <item Android:id="@+id/mnuLlamar"
          Android:icon="@drawable/llamar"
          Android:title="@string/menuLlamar" />
    <item Android:id="@+id/mnuSMS"
          Android:icon="@drawable/sms"
          Android:title="@string/menuSMS" />
    <item Android:id="@+id/mnuEmail"
          Android:icon="@drawable/mail"
          Android:title="@string/menuEmail" />
</menu>
```

Código 19. XML menú principal.

El segmento de Código 19 genera la interfaz mostrada en la Figura 34.

Figura 34. Menú principal.

Para incluir un menú principal a la activity se utiliza el método onCreateOptionMenu() para indicar el menú XML a desplegar.

```
public boolean onCreateOptionsMenu(Menu pMenu) {
   MenuInflater vInflater = getMenuInflater();
   vInflater.inflate(R.menu.menumapa, pMenu);
   return true;
}
```

Código 20. Ejemplo vinculación menú con activity.

Cada vez que un usuario selecciona una opción del menú principal se invoca el método onMenuItemSelected(), el cual se sobrescribe para dar la funcionalidad necesaria según la opción seleccionada.

```
public boolean onMenuItemSelected(int pFeatureId, MenuItem pItem) {
   Intent vIntent;
   switch (pItem.getItemId()) {
     // selecciono mostrar lista de contactos
     case R.id.mnuContactos:
       // lanza activity que muestra la lista de contactos
       vIntent = new Intent(this, ContactosList.class);
       this.startActivityForResult(vIntent, R.id.mnuContactos);
       return true;

     // selecciono configurar la vista del mapa
     case R.id.mnuConfigurar:
       // lanza activity que permite setear configuracion del mapa
       vIntent = new Intent(this, ConfiguracionMapa.class);
       this.startActivityForResult(vIntent, R.id.mnuConfigurar);
       return true;
```

```
        }
      }
      return super.onMenuItemSelected(pFeatureId, pItem);
   }
```

Código 21. Ejemplo evento selección opción de menú.

Una vez seleccionada una opción del menú, el método onMenuItemSelected() puede hacer una llamada a otra activity para que se ejecute la opción indicada, estableciendo mediante la sentencia starActivityForResult() que se espera un resultado en la actual activity a su regreso. El método onActivityResult() es invocado luego de ejecutarse la funcionalidad de la opción seleccionada desde otra activity para implementar los cambios necesarios en la activity actual.

```
protected void onActivityResult(int pRequestCode, int pResultCode,
        Intent pData) {
    super.onActivityResult(pRequestCode, pResultCode, pData);
    // se cancelo la operacion desde el activity lanzado
   if (pResultCode == RESULT_CANCELED)
      return;

   switch (pRequestCode) {
      case R.id.mnuContactos:
         ……
         break;
      case R.id.mnuConfigurar:
         ……
         break;
      ……
   }
}
```

Código 22. Ejemplo regreso de opción de menú.

Los menús contextuales también se definen en documentos XML, pero a diferencia de los menús principales se sobrescribe el método onCreateContextMenu() para su creación.

```
public void onCreateContextMenu(ContextMenu pMenu, View pView,
ContextMenuInfo pMenuInfo) {
    MenuInflater vInflater = getMenuInflater();
   // especifica interfaz xml del menu contextual
   vInflater.inflate(R.menu.menusolicitudcontactolist, pMenu);

}
```

Código 23. Ejemplo vinculación menú contextual con activity.

Cada vez que el usuario selecciona una opción del menú contextual se ejecuta el método onContextItemSelected(), el cual se sobrescribe para poder dar la funcionalidad que se desea ejecutar.

```
public boolean onContextItemSelected(MenuItem pItem) {
   switch (pItem.getItemId()) {
      // presiona confirmar solicitud de amistad
      case R.id.mnuAceptarSolicitud:
        // crea mensaje de confirmar solicitud que se envia al
servidor
        ……
        break;
      // seleccionar rechazar solicitud de amistad
      case R.id.mnuEliminarSolicitud:
        // genera mensaje xml de rechazo de amistad
        ……
        break;
      }
      return true;
   }
```

Código 24. Ejemplo selección de opción en menú contextual.

9.5.5 Servicio Google Maps

El servicio Google Maps permite a través del componente MapView visualizar un mapa en el cual se ubican los contactos de la aplicación. Este componente posee un controlador denominado MapController que permite controlar el comportamiento del mapa, ya sea centrando el mismo en una ubicación particular, cambiando el zoom o realizando una animación desde una ubicación a otra. El control MapView se contiene dentro de un MapActivity para poder ser visualizado en la interfaz de la aplicación. La clase MapActivity además de contener todos los métodos necesarios para visualizar una activity contiene métodos para el manejo de mapas.

Para que un control MapView se comunique con el servicio de Google y muestre un mapa, se provee al mismo de una clave de acceso a la API. Para obtener dicha clave se realizó un registro mediante una cuenta de correo electrónico de Google en el sitio http://code.google.com/intl/es-ES/apis/maps/signup.html.

Para la utilización de mapas se debe declarar en el archivo manifiesto de la aplicación la siguiente línea que figura en el segmento de Código 25, para indicar que se hace uso de la librería com.google.Android.maps para manipular mapas.

```
<uses-library Android:name="com.google.Android.maps" />
```

Código 25. Archivo manifiesto. Uso de librería Map de Google.

Para descargar mapas desde el servicio de Google se dota a la aplicación de acceso a Internet, esto se logra incluyendo el siguiente permiso en el manifiesto que figura en el segmento de Código 26.

```
<uses-permission Android:name="Android.permission.INTERNET" />
```

Código 26. Archivo manifiesto. Permiso uso de Internet.

9.5.6 Acceso al GPS del dispositivo

Para acceder a los datos de localización del dispositivo móvil se utiliza el servicio GPS que dispone el mismo. Para este acceso, se utiliza el objeto LocationManager al cual se le solicitan las actualizaciones de ubicación. Junto con la solicitud de actualizaciones, se establece qué clase es la encargada de procesar éstas. En el caso de Mis Contactos, la clase GPSListener es la encargada de monitorear los cambios de ubicación del dispositivo para poder procesar los mismos actualizando los datos correspondientes en la aplicación.

```
public class MisContactosMapa extends MapActivity implements
OnClickListener {
.......
private void habilitarGPSUsuario() {
   LocationManager mLocalizacion = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);

   mGestorPosicion = new GPSListener();
   mLocalizacion.requestLocationUpdates(LocationManager.GPS_PROVIDER,
3000, 10, mGestorPosicion);

   Location mPosicion = null;
   // verificar si hay GPS disponible
   boolean vGPSEncendido =
mLocalizacion.isProviderEnabled(LocationManager.GPS_PROVIDER);

   if (vGPSEncendido) {
      // gps encendido, obtiene la ultima posición conocida del
      // dispositivo
      mPosicion =
mLocalizacion.getLastKnownLocation(LocationManager.GPS_PROVIDER);
      if (mPosicion != null && mUsuario != null) {
         mUsuario.setLocalizacion(new GeoPoint((int)
(mPosicion.getLatitude() * 1E6),(int) (mPosicion.getLongitude() *
1E6)));
      }
   }
……
}
……
```

```
}
```

Código 27. Ejemplo utilización de servicio GPS.

```
private class GPSListener implements LocationListener {

   public void onLocationChanged(Location pLocation) {
     Contacto vUsuario = Global.getUsuario();
     // genera un geopoint con la nueva localizacion
     GeoPoint vGeoPoint = new GeoPoint((int)
(pLocation.getLatitude() * 1E6),(int) (pLocation.getLongitude() *
1E6));
     // actualiza al usuario con la nueva localizacion
     vUsuario.setLocalizacion(vGeoPoint);
     ……..
   }
   ……..
}
```

Código 28. Ejemplo clase de monitoreo de GPS.

Además de lo anteriormente expuesto, para acceder al servicio de GPS del dispositivo es necesario habilitar el permiso de acceso al mismo desde el archivo manifiesto de la aplicación incluyendo la siguiente línea que figura en el segmento de Código 29.

```
<uses-permission
Android:name="Android.permission.ACCESS_FINE_LOCATION" />
```

Código 29. Archivo manifiesto. Permiso de acceso al servicio de localización.

9.5.7 Conexion a Internet

La aplicación necesita el acceso a Internet tanto para acceder al servicio de Google Maps para la descarga de mapas como para poder comunicarse con el servidor. Para esto se dispone de una conexión y un mecanismo de monitoreo del estado de la misma. La clase MiReceiver.java, que extiende de BroadcastReceiver, sobrescribe el método onRecive() para capturar las modificaciones en el estado de la conexión notificadas por Android.

```
public void onCreate(Bundle pSavedInstanceState) {
   ……..
   registerReceiver(new MiReceiver(), new
IntentFilter("Android.net.com.CONNECTIVITY_CHANGE"));

   mConexion = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
     mConexion.setNetworkPreference(ConnectivityManager.TYPE_WIFI);
   boolean vInternetDisponible = mConexion.getActiveNetworkInfo() !=
null && mConexion.getActiveNetworkInfo().isAvailable()
          &&   mConexion.getActiveNetworkInfo().isConnected();

   if (vInternetDisponible) {
```

```
     // bindea el servicio que se encarga de mantener actualizada la
     // localizacion de los contactos
     bindService(new Intent(IRegistroServicio.class.getName()),
mConector, Context.BIND_AUTO_CREATE);
     Intent vServicioActualizacion = new Intent(this,
ServicioActualizacion.class);
     // lanza servicio
     startService(vServicioActualizacion);

   }
   ……….
};
```

Código 30. Ejemplo de utilización de connexion a Internet.

```
private class MiReceiver extends BroadcastReceiver {
   public void onReceive(Context pContext, Intent pIntent) {
     boolean vInternetDisponible = mConexion.getActiveNetworkInfo()
!= null  && mConexion.getActiveNetworkInfo().isAvailable()  &&
mConexion.getActiveNetworkInfo().isConnected();
     if (vInternetDisponible) {
       // si hay interent disponible ejecuto lanzo servicio
   // de actualizacion de ubicacion
       // de contactos desde el servidor
       bindService(new Intent(IRegistroServicio.class.getName()),
       mConector, Context.BIND_AUTO_CREATE);
       Intent vServicioActualizacion = new Intent(pContext,
ServicioActualizacion.class);
       pContext.startService(vServicioActualizacion);
     }
   }
}
```

Código 31. Ejemplo clase de monitoreo de estado de conexión a Internet.

Para tener acceso a las conexiones y a las notificaciones de cambio de estado de las mismas, se incorporan en el manifiesto las líneas que figuran en el segmento de Código 32.

```
<uses-permission Android:name="Android.permission.INTERNET" />
<uses-permission Android:name="Android.permission.ACCESS_WIFI_STATE"
/>
<uses-permission Android:name="Android.permission.CHANGE_WIFI_STATE"
/>
<uses-permission
Android:name="Android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
Android:name="Android.permission.CHANGE_NETWORK_STATE" />
```

Código 32. Archivo manifiesto. Permiso de utilización de Internet, WI-FI y cambios en la conexión.

9.5.8 Llamadas telefónicas

Para realizar llamadas telefónicas se lanza un intent con tal fin. El objetivo de éste es delegar una determinada acción al sistema, en este caso la

realización de una llamada. Android cuenta con un gestor de llamadas telefónicas que se especifica en el intent mediante el parámetro Intent.ACTION_CALL. Una vez creado se establece el número telefónico al que se desea llamar.

```
public boolean onMenuItemSelected(int pFeatureId, MenuItem pItem) {
   Intent vIntent;
   switch (pItem.getItemId()) {
      ……….
      // selecciono llamar al contacto actual
      case R.id.mnuLlamar:
         vIntent = new Intent(Intent.ACTION_CALL);
         Uri vUri = Uri.parse("tel:" +
mContactos.getContactoActual().getNumero());
         vIntent.setData(vUri);
         this.startActivity(vIntent);
         return true;
      ……….
      }
      return super.onMenuItemSelected(pFeatureId, pItem);
   }
```

Código 33. Ejemplo utilización de intent de llamada telefónica.

Además de la declaración de un intent para que una aplicación pueda realizar llamadas telefónicas se dota a la misma del permiso para poder hacerla. Dicho permiso se incluye en el archivo manifiesto con la línea que figura en el segmento de Código 34.

```
<uses-permission Android:name="Android.permission.CALL_PHONE" />
```

Código 34. Archivo manifiesto. Permiso de llamadas.

9.5.9 Envío de mensajes de texto

Al igual que para la realización de llamadas telefónicas, para el envío de mensajes de texto en Android se crea un intent, en este caso con el parámetro Intent.ACTION_VIEW. A éste se asigna el número de teléfono del destinatario y el tipo de mensaje.

```
case R.id.mnuSMS:
   // lanza activity ACTION_VIEW que permite enviar un sms a un
   // contacto
   vIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("smsto:"
+ mContactos.getContactoActual().getNumero()));
   vIntent.setType("vnd.Android-dir/mms-sms");
   vIntent.putExtra("address",
mContactos.getContactoActual().getNumero());
   startActivity(vIntent);
   return true;
```

Código 35. Ejemplo uso de intent para envío de mensajes de texto.

Para el envío de mensajes de textos en necesario declarar el permiso correspondiente en el manifiesto mediante la línea que figura en el segmento de Código 36.

```
<uses-permission Android:name="Android.permission.SEND_SMS" />
```

Código 36. Archivo manifiesto. Permiso de envío de mensajes de texto.

9.5.10 Envío de correos electrónicos

Para el envío de correos electrónicos se crea un intent específico para tal fin, al igual que para las llamadas telefónicas y envío de mensajes.

El tipo de intent para el envío de mails es Intent.ACTION_SEND. A este intent se le suministran las direcciones de correo electrónico de los destinatarios y el tipo de dato asociado al mail.

```
case R.id.mnuEmail:
   // lanza activity ACTION_SEND de Android que permite enviar mails
   vIntent = new Intent(Intent.ACTION_SEND);
   String[] vDestinatarios = new String[] {
mContactos.getContactoActual().getEMail() };
   vIntent.putExtra(Intent.EXTRA_EMAIL, vDestinatarios);
   vIntent.setType("message/rfc822");
   this.startActivity(Intent.createChooser(vIntent,"Enviar un email
a: " + mContactos.getContactoActual().getNombre()));
   return true;
```

Código 37. Ejemplo uso de intent para envío de mails.

Los permisos necesarios que se declaran en el manifiesto para el envío de mails se muestran en el segmento de Código 38.

```
<uses-permission Android:name="Android.permission.GET_ACCOUNTS"/>
<uses-permission Android:name="Android.permission.MANAGE_ACCOUNTS"/>
<uses-permission Android:name="Android.permission.ACCOUNT_MANAGER"/>
```

Código 38. Archivo manifiesto. Permiso de uso de cuentas de correo.

9.5.11 Geocodificación inversa

La geocodificación inversa es el proceso mediante el cual se obtiene una calle, número de puerta, etc., a partir de determinadas coordenadas. Este proceso permite convertir una coordenada obtenida por un GPS en una dirección postal que es más fácil de entender por el usuario final.

Android permite realizar este proceso utilizando funcionalidades ofrecidas por Google.

La aplicación Mis Contactos utiliza la geocodificación para mostrar al usuario las direcciones estimadas en las cuales se encuentran sus contactos. Para ello se declara un objeto de tipo Geocoder y luego se invoca a su método getFromLocation() con las coordenadas sobre las cuales se desea aplicar el proceso de geocodificación inversa.

```
  Geocoder vGeocoder = new Geocoder(Global.getContexto(),
Locale.getDefault());
  try {
    List<Address> vDirecciones = vGeocoder.getFromLocation(
mContacto.getLocalizacion().getLatitudeE6() / 1E6,
mContacto.getLocalizacion().getLongitudeE6() / 1E6, 1);
    if (vDirecciones.size() > 0) {
      mReverseGeoCoding = "Direccion:\n";
      for (int i = 0; i < vDirecciones.get(0).
getMaxAddressLineIndex(); i++)
        mReverseGeoCoding = mReverseGeoCoding + "\t" +
vDirecciones.get(0).getAddressLine(i) + "\n";
    }

  } catch (IOException e) {
    e.printStackTrace();
  }
```

Código 39. Ejemplo de geocodificación.

9.5.12 Servicio de actualización

La aplicación cada cierto intervalo de tiempo establecido y en el caso de contar con una conexión a Internet, realiza una comunicación con el servidor para informar sobre la ubicación actual del dispositivo, solicitando la últimas ubicaciones registradas de sus contactos. Esta comunicación se realiza mediante la clase ServicioActualizacion que extiende de la clase Service y ejecuta un hilo que luego de comunicarse con el servidor, para solicitar ubicaciones actualizadas de sus contactos e informar la suya, duerme un intervalo de tiempo antes de realizar nuevamente la petición.

```
 public class ServicioActualizacion extends Service {

  /*************************************************************/
  /* onCreate */
  /*************************************************************/
  /*
   * Evento que se ejecuta al crearse el servicio, inicializa las
variables globales
   * @see Android.app.Service#onCreate()
   */
  @Override
  public void onCreate() {
    super.onCreate();
```

```
      mServidor = getResources().getString(R.string.servidor);
      mIntervaloActualizacion = new Integer(getResources().getString(
           R.string.intervaloActualizacion));
      mConexion = new ConexionHTTP(mServidor);

      // implementa el metodo run del hilo
      mTarea = new Runnable() {
        public void run() {
        .............
          while (true) {
                 String vDataOut = mListaRetrollamadasRemota
                      .getBroadcastItem(0)
                      .notificarInicioConexion();
                 if (vDataOut.length() > 0) {
                   // setea los mensajes y se los envia al servidor
                   mConexion.setDataOut(vDataOut);
                   // si existe conexion con el servidor recupera
                   // la respuesta y la procesa
                   if (mConexion.conectar())
                     mListaRetrollamadasRemota.getBroadcastItem(
                          0).notificarFinConexion(
                          mConexion.getDataIn());
                 }

               }
               mListaRetrollamadasRemota.finishBroadcast();

               try {
                  Thread.sleep(mIntervaloActualizacion);
               } catch (InterruptedException e) {
                  e.printStackTrace();
               }
             }
          }
      };

      mHiloServicio = new Thread(mTarea);
      // inicia ejecucion de hilo
      mHiloServicio.start();

   }

}
```

Código 40. Ejemplo de uso de hilos como servicio de actualización.

Para las conexiones con el servidor se utiliza la clase ConexionHTTP la cual hace uso del DefaultHttpClient inicializado luego de la autenticación de la cuenta del usuario con el servidor App Engine.

Las peticiones del cliente hacia el servidor se realizan creando un mensaje HttpPost que es enviado en el método execute del objeto DefaultHttpClient.

```
/*************************************************************/
/* conectar() */
```

```
/*************************************************************/
* Envia datos en formato xml y recupera y procesa respuesta
*/
public boolean conectar() {
     // arreglo donde se pasan los mensajes al servidor
   List<BasicNameValuePair> vMensajes = new
ArrayList<BasicNameValuePair>();
   vMensajes.add(new BasicNameValuePair("XML", mDataOut));
   try {
     UrlEncodedFormEntity vEntidad = new
UrlEncodedFormEntity(vMensajes,  HTTP.UTF_8);
     // los mensajes se envian post al servidor
     HttpPost vPeticion = new HttpPost(mServer);

     vPeticion.setEntity(vEntidad);

     HttpParams vHttpParametros = new BasicHttpParams();
     HttpConnectionParams.setConnectionTimeout(vHttpParametros,
10000);
     HttpConnectionParams.setSoTimeout(vHttpParametros, 10000);
     DefaultHttpClient vClienteHttp = Global.getDefaultHttpClient();
     vClienteHttp.setParams(vHttpParametros);
     HttpContext vLocalContext = new BasicHttpContext();

     // se envia la peticion al servidor desde el cliente
     HttpResponse vRespuesta = vClienteHttp.execute(vPeticion,
vLocalContext);
     // se recibe la respuesta del servidor
     BufferedReader vEntrada = new BufferedReader(new
InputStreamReader(vRespuesta.getEntity().getContent()));
     String vLinea;
     mDataIn = "";
     while ((vLinea = vEntrada.readLine()) != null) {
          mDataIn = mDataIn.concat(vLinea);
     }
     return true;
   } catch (Exception e) {
     e.printStackTrace();
   }
   return false;
};
```

Código 41. Ejemplo de conexión con el servidor para envío de solicitudes HTTPost.

9.5.13 Simulación

El SDK de Android permite crear emuladores de dispositivos móviles sobre los cuales se puede probar y depurar la aplicación sin la necesidad de contar con un dispositivo móvil. Todo el proyecto fue desarrollado utilizando un emulador creado con el SDK de Android para realizar las pruebas correspondientes.

9.5.14 Instalación

Como resultado de la compilación de la aplicación en Eclipse se genera un archivo .apk que se encuentra en la carpeta bin del proyecto. El archivo generado misContactos.apk es el único fichero necesario para la instalación de la aplicación en un dispositivo móvil.

Existen aplicaciones de exploración de documentos tales como ASTRO o AppInstaller que se pueden bajar desde el Market de Android, permitiendo buscar el archivo en el dispositivo y poder ejecutarlo para la instalación de la aplicación.

Otra opción para la instalación de la aplicación puede ser escaneando el código QR que se puede obtener desde el siguiente sitio Web http://miscontactosservidor.appspot.com/miscontactosservidor con un lector de códigos de barras tal como Barcode Scanner.

Figura 35. Código QR instalador de Mis Contactos.

CAPÍTULO 10 Conclusiones y Trabajos Futuros

En este último Capítulo se exponen las conclusiones en referencia a los objetivos definidos inicialmente y se mencionan posibles mejoras e incorporación de funcionalidades a la aplicación.

10.1 Conclusiones finales

Haciendo una revisión de los objetivos generales y particulares establecidos, se resumen tendencias, aspectos y características principales que emergen de la investigación, a modo de conclusión.

10.1.1 Sobre objetivos generales

Investigar paradigmas y tecnologías de desarrollo de aplicaciones móviles

Existe una gran diferencia en la forma en que los usuarios de escritorio y los usuarios de los dispositivos móviles hacen uso de una aplicación y esta diferencia se debe en gran medida a las características de estos dispositivos.

Debido al incremento en la necesidad de aplicaciones móviles, los desarrolladores se enfrentan con nuevos desafíos y para afrontarlos tratan de aplicar la experiencia adquirida en el desarrollo de aplicaciones para otros ámbitos.

Para el desarrollo de aplicaciones móviles existe una gran variedad de plataformas. En el momento de seleccionar una plataforma se deben tener en cuenta las características de los dispositivos sobre los cuales correrá la aplicación y el lenguaje con el que el desarrollador está familiarizado. Entre las plataformas más utilizadas se encuentran:

- *J2ME*: es una versión reducida de Java orientada a dispositivos móviles.
- *BREW*: está basada en los lenguajes C y C++ ofreciendo una herramienta para el desarrollo de aplicaciones dinámicas y altamente gráficas en ambientes móviles.
- *.NET Compact Framework*: creada por Microsoft, está basada en los servicios Web XML y fue desarrollada teniendo en cuenta las limitaciones de los dispositivos móviles.

- *Android*: Es de libre distribución, utiliza una versión del lenguaje Java y se debe contar con una máquina virtual especialmente diseñada para la ejecución de esta plataforma.

Para poder realizar pruebas de las aplicaciones móviles existen emuladores que simulan tanto el SO como las características de los dispositivos móviles para los cuales se desarrolla una aplicación. En el caso de la plataforma Android, se ha investigado el SDK de Android que incluye un emulador AVD (Android Virtual Device) el cual puede ser utilizado para testear las aplicaciones desarrolladas en esta plataforma sin necesidad de contar con un dispositivo real.

Desarrollar una aplicación móvil utilizando la plataforma Android de Google

Una vez que se adquirió el conocimiento necesario para el diseño y desarrollo de aplicaciones móviles y se investigó la plataforma de desarrollo Android de Google, se procedió al desarrollo de la aplicación Mis Contactos. El objetivo del desarrollo no fue su utilidad final sino el análisis y la aplicación experimental de las principales características ofrecidas por esta plataforma.

10.1.2 Sobre objetivos particulares

Analizar las características de dispositivos móviles, capacidades y limitaciones de los mismos

Los dispositivos móviles se pueden clasificar según su funcionalidad en dispositivos de comunicación, de computación, reproductores y grabadores multimedia y consolas de juego portátil.

La tendencia actual en el diseño de dispositivos móviles es la creación de smartphone o teléfonos inteligentes, en lugar de dispositivos que se encuentren limitados al envío o recepción de llamadas o mensajes de texto. Estos teléfonos inteligentes, además de brindar las prestaciones de los teléfonos tradicionales, agregan nuevas características que los acercan más a un computador portátil.

Si bien las características y capacidades de los dispositivos móviles se encuentran en constante innovación, existen aún limitaciones frente a los computadores de escritorio, tales como, el poder de procesamiento, el tamaño

de las pantallas o teclados reducidos. Entre tales capacidades se destacan la comodidad, práctica movilidad y el acceso a la información deseada en cualquier momento y lugar, entre otras.

Analizar las características generales de las aplicaciones móviles

El incremento en el uso de dispositivos móviles trae aparejada la necesidad de una mayor cantidad de aplicaciones móviles que cubran las necesidades de los usuarios en las diferentes plataformas existentes.

De acuerdo con su arquitectura, las aplicaciones móviles se pueden clasificar según los siguientes criterios

- *Aplicabilidad*: se refiere a si la aplicación es independiente o dependiente de una plataforma determinada. Las aplicaciones dependientes de la plataforma permiten a los desarrolladores sacar el máximo provecho del dispositivo haciendo uso de su API, con la desventaja de que todos los usuarios que no disponen de dicha plataforma no podrán utilizar la aplicación. Si se pretende una aplicación para un público en general, es deseable que la misma sea soportada por la mayor cantidad de plataformas posibles, de esta manera se logra incrementar el segmento del mercado para el cual estará disponible la aplicación.
- *Arquitectura*: se refiere a la estructura en términos de componentes de la aplicación y éstos pueden ser peer to peer cuando todos los puntos de la aplicación desempeñan tareas semejantes, o cliente/servidor cuando un cliente hace solicitudes de servicios o información a un servidor con una mayor capacidad de procesamiento que el cliente.
- *Funcionalidad*: las aplicaciones móviles se pueden clasificar en tres diferentes categorías: sistemas de mensajes, aplicaciones Web y aplicaciones similares a las de escritorio.
- *Rango*: se refiere a la amplitud de dispositivos en los cuales se podrán ejecutar las tareas pretendidas de la aplicación. La tecnología móvil SMS se encuentra presente en la mayoría de los dispositivos móviles, por lo cual una aplicación basada en un sistema de mensajes SMS estaría disponible en casi la totalidad de los dispositivos.

En el momento de desarrollar una solución se debe considerar el tipo más conveniente en función de la necesidad dada, lo cual depende de diversos factores tales como si se requiere conexión o sincronización con un servidor central, la diversidad o no de dispositivos y la capacidad de interacción con el equipo que se requiera. Entre los tipos de soluciones posibles se encuentran:

- *Stand-alone*: se desarrollan para ser instaladas y ejecutadas sobre equipos específicos y funcionan en forma desconectada de la red. Las aplicaciones de este tipo tienen una mayor velocidad de ejecución y suelen aprovechar las características de los dispositivos logrando un mejor rendimiento. Presentan limitaciones tales como incompatibilidades entre diferentes SO, instalación manual y capacidad de almacenamiento limitado.
- *Online*: se valen de páginas Web o WAP como interfaz a través de Internet para lograr su función. Se destacan entre sus ventajas un mayor rango de aplicabilidad, no requieren instalación y aprovechan la potencia de cómputo de los servidores. Entre sus limitaciones se pueden mencionar la necesidad de una conexión permanente y acceso a información más lento que si éstas estuviesen almacenadas en el dispositivo.
- *Smart Client*: combinan las ventajas de Stand-alone y Online, se deben instalar en los dispositivos pero utilizan una conexión para comunicarse e interactuar con un servidor Online.

En cuanto al almacenamiento, manipulación e intercambio de la información en aplicaciones móviles, éstos pueden realizarse a través de una arquitectura de documentos compartidos o de una base de datos.

La arquitectura de documentos compartidos permite la actualización de una copia del documento en el servidor (SCDS) o de las múltiples copias distribuidas en los diferentes clientes (MCDS). En el caso de las aplicaciones que necesitan actualizar partes de estos documentos, es necesario que la aplicación conozca la estructura interna del documento para poder realizar la administración y actualización de los datos. Una aplicación que hace uso de una base de datos, puede llevar a cabo actualizaciones con mayor nivel de

detalle debido a que la aplicación encargada de administrar la base de datos conoce su estructura interna.

Investigar la tecnología de arquitectura wireless para aplicaciones móviles

Casi la totalidad de las comunicaciones de los dispositivos móviles necesitan redes de comunicaciones inalámbricas. Entre las diferentes tecnologías inalámbricas utilizadas por los dispositivos se destacan Wi-Fi 802.11, Bluetooth, infrarrojo, GSM, GPRS y 3G. Cada una de estas tecnologías brinda conexiones con diferentes características de transmisión de datos entre las cuales es posible mencionar el ancho de banda, la seguridad, fiabilidad y rango de alcance.

Una aplicación que requiera una conexión puede disponer de varios métodos de conexión a la red y deberá decidirse qué tecnología utilizar.

Investigar patrones de diseño para aplicaciones móviles

Ya que el uso de patrones brinda un beneficio indiscutible en el proceso de diseño y desarrollo de aplicaciones móviles, es necesario tener conocimiento de los diferentes patrones existentes para aplicarlos en aquellos casos en los cuales sea conveniente.

Investigar la plataforma Android para el desarrollo de aplicaciones móviles y aplicar los conocimientos adquiridos en el desarrollo de la aplicación final

Se realizó una breve introducción a la historia de Android y su filosofía de código abierto. Esta filosofía brinda ventajas en el desarrollo de aplicaciones sobre sus competidores.

Los desarrollos en esta plataforma se realizan en Java lo cual es una ventaja para aquellos desarrolladores familiarizados con el lenguaje.

La plataforma permite desarrollar rápidamente aplicaciones portables y reutilizables, así como controlar las funcionalidades ofrecidas por los dispositivos móviles tales como llamadas, mensajes de texto y manejo de conexiones, entre otras.

El componente principal de Android es su máquina virtual Dalvik diseñada para asegurar que múltiples instancias se ejecuten de manera eficiente en un mismo dispositivo.

Las aplicaciones en Android están integradas por componentes lo que ayuda a modelizarlas funcionalmente. Estos componentes pueden ser interfaces de usuario (Activity), procesos que se ejecutan en segundo plano (Services), proveedores de contenidos (Content Providers) y receptores de difusión (Broadcast Receiver).

Además, mediante el uso de Intents, las aplicaciones pueden delegar determinada funcionalidad en el sistema. Para ello, la aplicación expresa la funcionalidad deseada y es el sistema el encargado de seleccionar la aplicación más adecuada para llevarla a cabo.

Otra de las características de Android es que permite el acceso y manejo de los recursos de los dispositivos de manera sencilla, tales como GPS, cámara fotográfica y WI-FI entre otros.

Instalando las herramientas necesarias en el entorno de desarrollo Eclipse se puede contar con un entorno gratuito, potente y completo con el cual desarrollar y probar las aplicaciones en Android.

La SDK de Android posee una amplia documentación y existe una gran cantidad de información sobre Android disponible en Internet, que junto con la característica de ser código abierto hacen de la plataforma una herramienta más que interesante para los desarrolladores de aplicaciones móviles.

Una vez conocidas las características de Android, su API y entorno de desarrollo, se procedió a desarrollar una aplicación bajo esta plataforma.

La aplicación Mis Contactos permite ubicar las localizaciones de una serie de contactos en mapas obtenidos desde Google Maps. El objetivo principal de este desarrollo no fue tanto su utilidad final, sino el análisis y aplicación de las principales características que Android ofrece.

En Mis Contactos se utilizan diferentes componentes básicos tales como Activity o Service; también se controlan algunos recursos del dispositivo móvil tales como el GPS, para conocer la ubicación propia, o Wi-Fi, para intercambiar información de localización con otros usuarios.

Para la aplicación se diseñaron varias interfaces de usuario en documentos XML y para enviar SMS, correos electrónicos o llamadas telefónicas se lanzan intents.

10.2 Trabajos futuros

Como posibles líneas futuras de trabajo de la actual investigación se prevén:

- Ampliar las funcionalidades ofrecidas por la aplicación Mis Contactos: Se propone permitir delimitar una zona del mapa, con el objetivo de notificar al usuario cuando un contacto determinado ingresa en dicha zona; proveer facilidades para dibujar sobre el mapa rutas posibles hacia un determinado contacto, especificando la distancia a recorrer hasta llegar a su encuentro; notificar la llegada de solicitudes de amistad y permitir el intercambio de mensajes entre contactos a modo pizarra de mensajes.
- Realizar nuevas aplicaciones que utilicen funcionalidades de Android que no fueron abarcadas en el desarrollo de Mis Contactos, tales como manejo de chat, cámara fotográfica, acelerómetro o Bluetooth.

CAPÍTULO 11 Referencias

[1]. Alejandro Botero López, Hugo Giraldo Arenas, Alexandra Moyano Romero, “Limitaciones del desarrollo de aplicaciones en dispositivos móviles”, Universidad Javeriana, Colombia, Artículo, 2003.

[2]. Arturo Baz Alonso, Irene Ferreira Artime, María Álvarez Rodríguez, Rosana García Baniello, “Dispositivos Móviles”, Universidad de Oviedo, Tesis, 2003.

[3]. Nirav Mehta, Mobile Web Development, Packt Publishing ed., Editado: Birmingham, Reino Unido, 2008.

[4]. Krešimir Fertalj, Marko Horvat, Comparing Architectures Of Mobile Applications, Faculty of Electrical Engineering and Computing University of Zagreb, Zagreb, 2008.

[5]. Alejandro Méndez Carvajal, “Prototipo de Aplicación Móvil para consulta e información de novedades de un portafolio financiero a través de un teléfono celular”, Universidad El Bosque Facultad De Ingeniería, Bogotá D.C., 2008.

[6]. Ing. Darío Dorio, “Identificación y Clasificación de Patrones en el Diseño de Aplicaciones Móviles”, Universidad Nacional de La Plata Facultad de Informática, La Plata, Tesis, 2009.

[7]. Firtman Maximiliano, Desarrollos Móviles con .Net, MP Ediciones S.A. ed., Editado: Buenos Aires, Argentina, 2005.

[8]. Reto Meier, Professional Android Application Development, Wrox ed., Editado: New York, Estados Unidos, 2009.

[9]. Google. (Julio 2010) Android Developers. [Online]. http://developer.Android.com/

[10]. Steven Snell. (Junio 2009) Smashing Magazine. [Online]. http://www.smashingmagazine.com/2009/01/13/mobile-web-design-trends-2009/

[11]. Java Móvil. (Agosto 2010) Android Tutoriales. [Online]. http://www.javamovil.info/Android/developers

[12]. Wikipedia. (Diciembre 2010) Mobile Application Development. [Online]. http://en.wikipedia.org/wiki/Mobile_application_development

[13]. Wikipedia. (Agosto 2010) Dispositivo Móvil. [Online]. http://es.wikipedia.org/wiki/Dispositivo_móvil

[14]. Tabasco, Blog. (Junio 2010) 2010 y el boom de los App-store. [Online]. http://tabascogroup.com/aplicaciones-moviles