Tecnología y desarrollo en dispositivos móviles

Josep Prieto Blázquez
Robert Ramírez Vique
Julián David Morillo Pozo
Marc Domingo Prieto

PID_00176751

Material docente de la UOC

Universitat Oberta de Catalunya
www.uoc.edu

Josep Prieto Blázquez

Licenciado en Informática por la Universidad Politécnica de Cataluña en el año 1993. Doctor en Informática por la Universitat Oberta de Catalunya en el año 2009. Desde el año 1998 trabaja de profesor de los Estudios de Informática y Multimedia en la UOC. Desde el año 2009 es subdirector de los Estudios de Informática, Multimedia y Telecomunicación de la UOC. Su línea de investigación se centra principalmente en la prospectiva y las aplicaciones tecnológicas a nivel de las TIC; en este sentido, ha participado en más 15 proyectos nacionales e internacionales relacionados con wireless y herramientas de aprendizajes en entornos virtuales.

Robert Ramírez Vique

Ingeniero informático por la Universidad Politécnica de Cataluña y máster por la Universidad Ramón Llull. Es profesor de la UOC desde hace más de seis años, donde trabaja como especialista en las asignaturas relacionadas con las tecnologías móviles. Es desarrollador de aplicaciones para diferentes arquitecturas y tiene experiencia en varios sectores (*eCommerce*, sector industrial, etc.) y tecnologías.

Julián David Morillo Pozo

Ingeniero informático por la Universidad Politécnica de Cataluña en el 2002 y doctor en Informática del Departamento de Arquitectura de Computarores por la Universidad Politécnica de Cataluña en el 2009 (DAC-UPC). Desde el año 2005 es profesor del Departamento de Arquitectura de Computadores de la UPC. Imparte clases en la Facultad de Informática de Barcelona (FIB) y en la Escuela Técnica Superior de Ingenieros de Telecomunicaciones de Barcelona (ETSETB). Desde el 2005 es consultor de la UOC. Sus líneas de investigación se centran en redes vehiculares, en redes tolerantes a retardo, en redes cooperativas y en redes de sensores multimedia.

Marc Domingo Prieto

Ingeniero técnico en Informática de sistemas y máster por la Universidad Politécnica de Cataluña (UPC). Ha trabajado realizando análisis de seguridad tanto en tecnologías de comunicación inalámbrica como en plataformas de dispositivos móviles. Actualmente trabaja como ayudante de investigación en el grupo de seguridad de la UOC KISON.

El encargo y la creación de este material docente han sido coordinados por el profesor: Josep Prieto Blázquez (2011)

Primera edición: septiembre 2011

Av. Tibidabo, 39-43, 08035 Barcelona
Diseño: Manel Andreu
Realización editorial: Eureca Media, SL
Depósito legal: B-23.648-2011

Introducción

Este material está claramente orientado a iniciar a los ingenieros informáticos en el desarrollo de aplicaciones sobre dispositivos móviles, con el objetivo de hacer que los conocimientos adquiridos puedan ser un valor añadido importante en su carrera profesional. La evolución de estas tecnologías en los últimos años ha sido espectacular.

Por una parte, las últimas dos décadas hemos vivido una revolución de las comunicaciones inalámbricas que ha facilitado la movilidad de las personas al reducir la dependencia del cable para comunicarnos. Por otra parte, en la última década estamos viendo una evolución espectacular de las prestaciones y características de los dispositivos móviles, llegando en muchos casos a ser un posible sustituto del ordenador portátil o de sobremesa.

Finalmente, durante los últimos años ha habido una explosión de las herramientas y lenguajes de programación para desarrollar aplicaciones sobre dispositivos móviles, así como la creación de nuevas maneras de compartir y vender estas aplicaciones a partir de mercados específicos llamados *tienda de aplicaciones* o *AppStores*. Todo ha hecho posible que numerosos programadores estén desarrollando aplicaciones para móviles de una manera rápida, barata y fácilmente comercializable. Nunca ha sido tan fácil crear aplicaciones y poder tener un escaparate de alcance mundial para poder venderlas.

Es por eso por lo que la mayor parte de los contenidos de esta asignatura están destinados al desarrollo de pequeñas aplicaciones y servicios sobre dispositivos móviles, que utilizan las tecnologías de comunicaciones inalámbricas como medio de comunicación. Además, también se introducen los conceptos de la seguridad de la comunicación y de la información que hay detrás y de las mismas aplicaciones, ya que es un factor clave para que muchos negocios puedan existir sobre dispositivos móviles utilizando las tecnologías inalámbricas.

Objetivos

Con el estudio de este módulo se pretende que el estudiante alcance los objetivos siguientes:

1. Entender qué es la comunicación inalámbrica y cuál es su alcance. Ver qué ventajas conlleva el hecho de utilizarlas.

2. Conocer los diferentes tipos de dispositivos móviles y cuáles son sus características.

3. Tener un conocimiento amplio y variado de las alternativas para el desarrollo de aplicaciones móviles.

4. Conocer las peculiaridades del diseño de aplicaciones móviles, especialmente las debidas a las limitaciones de los dispositivos.

5. Ser capaces de poder dirigir un proyecto relacionado con las tecnologías de desarrollo sobre móvil, sabiendo qué hacer en cada una de sus fases y proporcionando las herramientas necesarias para afrontar el proyecto con garantías.

6. Conocer la problemática concreta de la seguridad en el desarrollo de aplicaciones sobre dispositivos móviles.

7. Saber cuáles son las prácticas de seguridad recomendadas cuando se utiliza un dispositivo móvil.

Contenidos

Marc Domingo Prieto

1. La problemática de la seguridad
2. Comunicaciones inalámbricas
3. Sistema operativo
4. Aplicaciones
5. Usuario
6. Prácticas de seguridad

Introducción a los sistemas de comunicación inalámbricos

Josep Prieto Blázquez

PID_00176752

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

Actualmente, estamos inmersos en la que se denomina *revolución tecnológica de las comunicaciones inalámbricas*, una revolución similar a la que protagonizaron en su momento la electricidad, la televisión, el ordenador o las mismas comunicaciones con cable, que supusieron nuevos modelos de negocio.

Una de las principales ventajas de esta tecnología es la movilidad, no depender del cable. El hecho de que el punto de entrada en la red de comunicaciones no esté ligado a una ubicación fija y que el medio de transmisión ya esté preparado favorece su expansión, que puede ser más rápida que la de cualquier otro tipo de tecnología. Existe un ejemplo que lo corrobora: en solo cinco años de existencia, la telefonía móvil ya tuvo más usuarios que la telefonía fija.

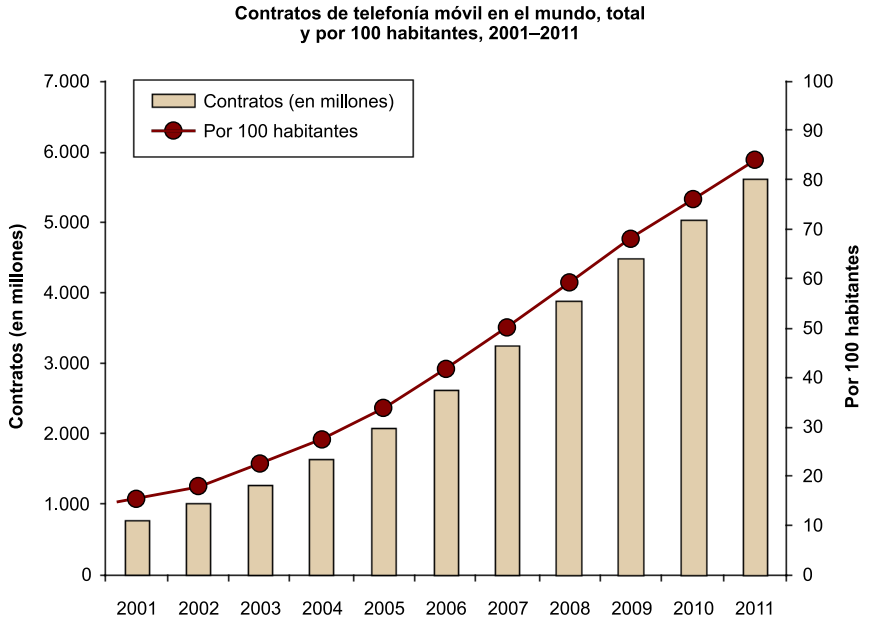
Difusión de las tecnologías móviles

Actualmente, más de un 75% de la población tiene un contrato de telefonía móvil. Esto quiere decir que la telefonía móvil de voz cuenta con más de 5.000 millones de usuarios en 212 países diferentes.

Por otra parte, más del 60% de la población mundial se conecta a Internet con una conexión inalámbrica.

El enfoque de este módulo está claramente orientado a iniciar a los ingenieros informáticos en las tecnologías de las comunicaciones inalámbricas, con el objetivo de ofrecerles un valor añadido importante en su carrera profesional como desarrolladores de ampliaciones o servicios sobre dispositivos móviles.

Internet también se ha beneficiado de esta tecnología, hecho que ha dado paso a lo que se conoce como *Internet móvil*, que permite que dispositivos móviles y personas se conecten a la Red desde cualquier lugar y en cualquier momento, lo que ha facilitado la aparición de nuevos servicios y aplicaciones sobre estos dispositivos.



Fuente: Base de datos ITU World Telecommunication/ICT Indicators

Objetivos

Con el estudio de este módulo se pretende que el estudiante alcance los objetivos siguientes:

1. Adquirir una visión global de las comunicaciones inalámbricas y conocer cuáles son sus rasgos característicos.

2. Saber cuál es el medio de transmisión de las comunicaciones inalámbricas.

3. Diferenciar los distintos tipos de comunicaciones inalámbricas.

4. Identificar las ventajas de la comunicación inalámbricas.

5. Tener una visión histórica de las comunicaciones inalámbricas.

1. Redes de computadores

En una red de computadores, podemos distinguir cuatro elementos importantes que intervienen en su definición:

1) El **protocolo de comunicación** define el lenguaje y el conjunto de reglas que facilitan la comunicación entre el emisor y el receptor, con el objetivo de que se puedan entender e intercambiar información. Existen muchos protocolos, pero seguramente el más conocido y más extendido entre los ordenadores es el TCP/IP[1] que utiliza Internet.

(1)TCP/IP: *transport control protocol/Internet protocol.*

2) La **topología** define cómo los nodos de comunicación están interconectados entre sí. Las topologías de red más comunes son en bus, estrella, anillo o punto a punto.

3) La **seguridad** es el elemento que permite garantizar la confidencialidad, la autenticación y la integridad de los datos.

4) El **medio de transmisión** es el elemento que diferencia más claramente las tecnologías de comunicación con hilos de las inalámbricas. Es el medio por el que viaja la señal que transfiere los datos.

Actualmente, las comunicaciones con cable (guiadas) utilizan distintos medios de transmisión, entre otros, el par trenzado (UTP[2] o STP[3]), el cable coaxial, la fibra óptica o los cables de alta tensión.

(2)UTP: *unshielded twiested pair.*

(3)STP: *shielded twiested pair.*

El medio de transmisión de las comunicaciones inalámbricas (no guiadas) es el espectro electromagnético que coloquialmente denominamos *aire*.

1.1 El espectro electromagnético

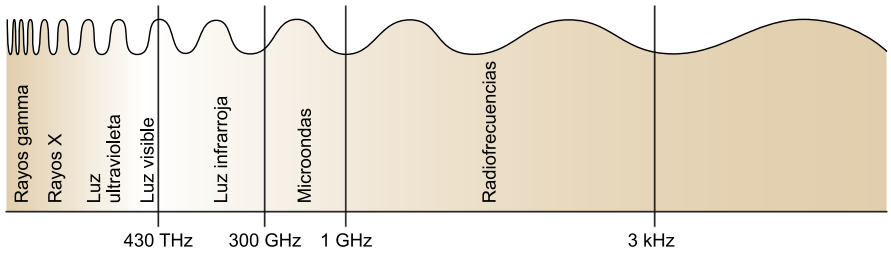
El espectro electromagnético es el rango de frecuencias de todas las ondas electromagnéticas que se pueden propagar a través del espacio libre, ordenadas según su longitud de onda y su frecuencia.

Como el propio nombre indica, estas ondas tienen un componente magnético y otro eléctrico. La forma más familiar de radiación electromagnética es la luz visible.

Ejemplos de frecuencias

La luz visible tiene una frecuencia de 500.000 GHz y la radio FM, de entre 80 y 110 MHz.

Espectro electromagnético



Ordenación de frecuencias

Desde el año 1999, el comité del IEEE 802.16 trabaja para definir los estándares de las comunicaciones inalámbricas de este rango de frecuencias. Para saber más sobre esto, podéis visitar la siguiente dirección:

http://standards.ieee.org/wireless

Los rangos de frecuencias más utilizados en las comunicaciones inalámbricas son los siguientes:

(4)PDA: *personal digital assistant.*

- **Infrarrojos (IR)**. Se utilizan en comunicaciones punto a punto de corto alcance, son muy direccionables y no pueden atravesar obstáculos. Este medio se utiliza habitualmente en el mando a distancia de la televisión y hasta hace unos años era también un sistema de comunicación que se utilizaba a menudo para conectar dispositivos situados el uno al lado del otro (un PDA[4] con el ordenador o con un móvil y el teclado con el ordenador). Es el rango de frecuencia más alto para comunicaciones inalámbricas.

- **Microondas (MW)**. Este rango de frecuencias es adecuado para transmisiones de largo recorrido (comunicaciones por satélite, comunicaciones terrestres punto a punto como alternativa al cable coaxial o la fibra óptica, y también la mayoría de las tecnologías inalámbricas más habituales que existen actualmente y que explicaremos brevemente en esta asignatura, como UMTS, Bluetooth o WLAN). Las microondas suelen ser direccionales y utilizan una parte del espectro con frecuencias más pequeñas que los infrarrojos.

- **Radiofrecuencias (RF)**. Es el rango que utilizan las transmisiones de radio (FM, AM) y televisión digital terrestre (TDT). Las radiofrecuencias son omnidireccionales y pueden atravesar obstáculos sin ningún problema.

Otras frecuencias

Existen otros rangos de frecuencias del espectro electromagnético, como la luz ultravioleta, los rayos X o los rayos gamma, que podrían tener mejores prestaciones que los infrarrojos, los microondas y las radiofrecuencias, dada su frecuencia tan alta, pero no se utilizan porque pueden llegar a ser peligrosos para los seres vivos y, además, son difíciles de producir y modular.

Para ordenar la utilización del espectro electromagnético, existen acuerdos nacionales e internacionales que regulan quién puede utilizar qué frecuencia. Hemos de pensar que el espectro es limitado y que hay muchos servicios que lo quieren utilizar: la radio, la televisión, los teléfonos inalámbricos, las compañías telefónicas, los agentes de policía, los militares, las redes de área local, etc.

La ITU[5] en el ámbito mundial y la FCC[6] en Estados Unidos se encargan de regular el uso de las frecuencias del espectro electromagnético.

(5)ITU: International Telecommunications Union. http://www.itu.int.

Las bandas autorizadas por estos organismos internacionales se denominan bandas IMS[7], y a la hora de conceder autorizaciones tienen en cuenta que la potencia de transmisión no sea perjudicial para la salud.

(6)FCC: Federal Communications Commission. http://www.fcc.gov.

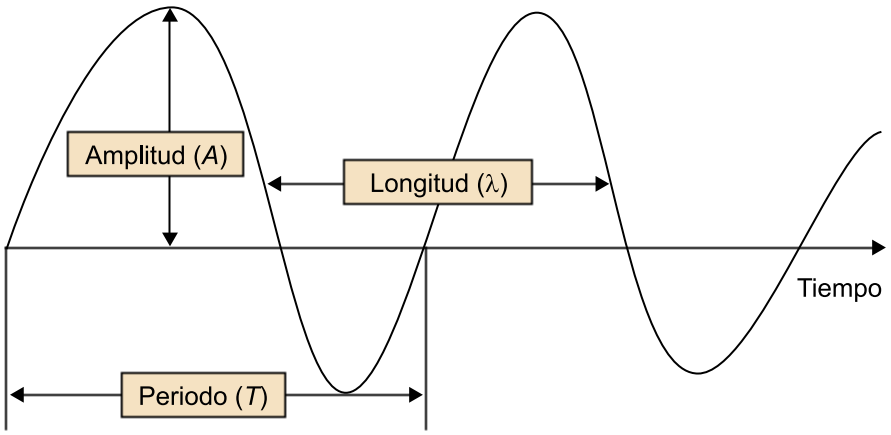
(7)IMS: *industrial, scientific and medical bands*.

1.2 Componentes de una onda

Para poder entender las comunicaciones inalámbricas es necesario conocer los conceptos fundamentales que definen una onda electromagnética:

- **Frecuencia (*f*)**. Número de oscilaciones por segundo de una onda o señal, se mide en *herz*. Una onda que realiza cinco ciclos por segundo tiene una frecuencia de 5 Hz.
- **Período (T)**. Cantidad de tiempo que tarda una onda en completar un ciclo: $T = 1/f$.
- **Fase (φ)**. Posición relativa en el tiempo dentro del período simple de una onda.
- **Longitud de onda (λ)**. Espacio que ocupa un ciclo completo de una onda, medido en metros: $\lambda = c/f$, donde c es la velocidad de la luz en el vacío (aproximadamente $3 \cdot 10^8$ metros/segundo).
- **Amplitud (a)**. Máximo valor o potencia de una onda en el tiempo, típicamente medido en voltios o decibelios.

Componentes de una onda



2. Comunicaciones inalámbricas

Antes de empezar a abordar los diferentes módulos de esta asignatura, es importante entender el alcance de la comunicación inalámbrica.

En un sentido amplio y general, entendemos por **comunicaciones inalámbricas** aquellas comunicaciones entre dispositivos (móviles o no) o entre personas que intercambian información utilizando el espectro electromagnético.

Enlace de interés

Para saber más sobre la comunicación Bluetooth podéis visitar la siguiente dirección de Internet:
http://www.bluetooth.com

Ejemplos de comunicaciones inalámbricas

La definición de comunicaciones inalámbricas engloba desde una comunicación Bluetooth entre un teléfono móvil y un ordenador portátil hasta una comunicación de dos terminales de telefonía móvil GSM. Incluso la comunicación verbal entre dos personas sería una comunicación inalámbrica: utilizan el aire como un canal para el intercambio de información.

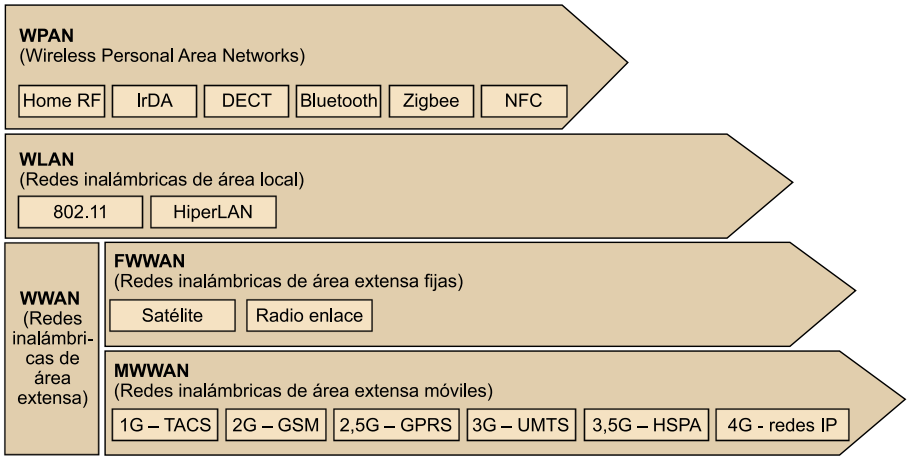
En esta asignatura nos centraremos en las comunicaciones inalámbricas relacionadas con las nuevas tecnologías y con las nuevas posibilidades de negocio.

2.1. Clasificación

Según la documentación que se consulte, se pueden encontrar diferentes clasificaciones de las comunicaciones inalámbricas. En esta asignatura, las clasificaremos atendiendo a su alcance y a la manera de controlar el acceso a la red.

Según el alcance, podemos establecer tres grandes grupos:

- **Redes de área personal inalámbrica** (WPAN: *wireless personal area networks*).
- **Redes de área local inalámbrica** (WLAN: *wireless local area networks*).
- **Redes de área extendida inalámbrica** (WWAN: *wireless wide area networks*). Podemos diferenciar dos tipos de WWAN, según quién controle su acceso:
  - **Comunicación fija** (FWWAN: *fixed wireless wide area networks*).
  - **Comunicación móvil** (MWWAN: *mobile wireless wide area networks*).



2.1.1. Redes personales inalámbricas (WPAN)

Las WPAN presentan una importante limitación de alcance: los dispositivos que pretenden comunicarse han de estar poco separados. Generalmente, se acepta como límite el espacio de una habitación o un despacho.

Enlace de interés

El grupo de trabajo IEEE 802.15 define los estándares referentes a las WPAN. Los podéis consultar en la siguiente dirección:

http://grouper.ieee.org/groups/802/15/

Usos de las redes WPAN

Las redes WPAN es una tecnología que ha llegado de manera progresiva a nuestra vida cotidiana con el objetivo de hacer las comunicaciones más cómodas y más fáciles de utilizar: la tecnología Bluetooth permite comunicar una impresora y un ordenador sin ningún cable, siempre que estén a una distancia de aproximadamente diez metros; mediante la tecnología Wi-Fi la distancia puede llegar a ser de hasta cien metros.

Las tecnologías más utilizadas de WPAN son las siguientes: Bluetooth, DECT[8], IrDa[9], NFC[10] y Zigbee.

(8)DECT: *digital enhanced cordless telecommunicatios.*

(9)IrDA: Infrared Data Association.

(10)NFC: *near field comnunication.*

Bluetooth

Bluetooth es una especificación regulada por el grupo de trabajo IEEE 802.15.1, que permite la transmisión de voz y datos entre diferentes dispositivos mediante un enlace de radiofrecuencia en la banda ISM de 2,4 GHz.

Bluetooth, rey vikingo

El nombre Bluetooth proviene del rey vikingo Harald Blatand (siglo X), que unificó y controló Dinamarca y Noruega. De aquí viene la inspiración del nombre: con esta tecnología se pretende unificar e interconectar dispositivos.

Se cree que una de las aficiones de este rey era comer moras y por eso tenía el "tinte" azul de los dientes (*bluetooth* quiere decir 'diente azul' en inglés).

Bluetooth permite conectar inalámbricamente diferentes dispositivos electrónicos, como asistentes digitales personales (PDA), teléfonos móviles, ordenadores portátiles, etc., lo que facilita, abarata y garantiza la interoperabilidad entre dispositivos de diferentes fabricantes.

Bluetooth define un alcance corto (alrededor de 10 m) y, opcionalmente, un alcance medio (alrededor de 100 m).

Evolución histórica

La evolución histórica de la tecnología Bluetooth es la siguiente:

- **1994**: Ericsson promueve un estudio de comunicaciones inalámbricas de bajo coste y baja potencia.
- **1998**: se crea el SIG (Special Interest Group), formado inicialmente por Ericsson, IBM, Intel Nokia y Toshiba.
- **1999**: aparece la versión 1.0 del estándar.
- **2001**: aparece la versión 1.1 del estándar.
- **2002**: el IEEE produce el estándar 802.15,1, que es compatible con la versión 1.1 de Bluetooth.
- **2004**: aparece la versión 2.0 del estándar, que destaca por el aumento de la velocidad de transferencia (hasta 3 Mbps).
- **2007**: aparece la versión 2.1 del estándar, que destaca por las mejoras en cuestiones de seguridad.
- **2009**: aparece la versión 3.0 del estándar, que destaca por el aumento considerable de la velocidad de transferencia (hasta 300 Mbps).
- **2011**: aparece la versión 4.0 del estándar, que destaca por la reducción significativa del consumo de batería.

En una red Bluetooth, cualquier dispositivo puede actuar como maestro o como esclavo:

- El dispositivo maestro se encarga de definir cómo se establece la comunicación físicamente (frecuencia de salto, fase, etc.).

- Los dispositivos esclavos coordinan sus transmisiones según las especificaciones del maestro. Normalmente, el primero que solicita el servicio actúa como maestro, excepto cuando la red ya ha sido establecida.

Enlace de interés

Para saber más sobre la comunicación Bluetooth, podéis visitar la siguiente dirección de Internet:

http://www.bluetooth.com

DECT

La tecnología *digital enhanced cordless telecommunicatios* (DECT) aparece como una necesidad de que las comunicaciones analógicas de la telefonía de principios de la década de los ochenta evolucionaran hacia un contexto digital. La transmisión digital inalámbrica ofrece una serie de ventajas respecto a la analógica: menos interferencias, más capacidad de dispositivos en una misma zona, más seguridad (se puede cifrar la información) y más movilidad (se pueden establecer mecanismos para saltar de una red a otra, característica denominada *roaming*).

El DECT opera en la banda de frecuencias de 1.880 a 1.900 MHz

El estándar DECT aparece oficialmente a principios de 1988 impulsado por el ETSI. Inicialmente, se centró en la definición del radioenlace entre los dispositivos inalámbricos y las estaciones fijas, y en los protocolos y estándares necesarios para desarrollar funciones de traspaso (*handover*) entre estaciones base.

El estándar DECT, que originalmente admitía transferencias de datos de hasta 552 Kbps, ha evolucionado hasta permitir transferencias de 2 Mbps.

Enlace de interés

ETSI es la sigla de European Telecommunications Standards Institute. Podéis encontrar información sobre este estándar en la siguiente dirección:

http://portal.etsi.org/

Más de cien países han reservado bandas de frecuencias para la transmisión de datos con el DECT. Además, en un gran número de países se opera en una banda de frecuencias protegida, es decir, libre de interferencias con otras tecnologías.

Usos de DECT

Los aparatos inalámbricos de telefonía de uso doméstico son los que utilizan más frecuentemente el DECT y, además, suelen operar con un rango de 50 m.

IrDa

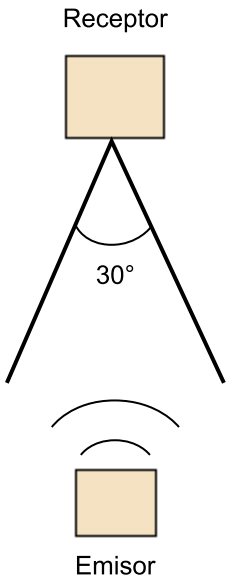
La Infrared Data Association (IrDA) es una asociación que integra más de ciento sesenta compañías. El estándar IrDA utiliza el espectro de frecuencia de infrarrojo para transmitir información.

Enlace de interés

Para saber más sobre el IrDA, podéis visitar la página web siguiente:

http://www.irda.org

El uso de la tecnología IrDA se ha extendido mucho, sobre todo en los años noventa y a principios de siglo, a causa de su bajo coste de implementación y su bajo consumo de batería. Además, es muy flexible y capaz de adaptarse fácilmente a un gran número de aplicaciones y dispositivos, como a asistentes digitales personales (PDA), teléfonos, impresoras u ordenadores portátiles.



Los dispositivos que utilizan la IrDA se comunican mediante el uso del diodo LED (*light emitting diode*). Es necesario que estos dispositivos estén alineados los unos con los otros. La desviación máxima permitida es de 30°.

NFC

La tecnología *near field communication* (NFC) permite la transmisión de datos de una manera simple entre diferentes dispositivos mediante un enlace de radiofrecuencia en la banda ISM de 13,56 MHz.

Dado que la conexión se produce cuando dos dispositivos NFC están muy próximos entre sí, a menos de 20 centímetros, la comunicación es inherentemente segura.

NFC fue aprobado por el estándar ISO 18092 en el 2003. Philips, Sony y Nokia formaron el NFC Foro para avanzar en el desarrollo de las especificaciones NFC y velar por su interoperabilidad.

La tecnología NFC es una extensión del estándar ISO/IEC-14443 para tarjetas de proximidad sin contacto que combina la interfaz de una tarjeta inteligente y un lector en un único dispositivo, lo que la hace compatible con toda la infraestructura de pago sin contacto que existe actualmente.

Aunque la tecnología NFC permite el intercambio de datos entre dispositivos, no está dirigida a la transmisión masiva de datos, como por ejemplo Bluetooth, sino a la comunicación entre dispositivos con capacidad de cálculo, como teléfonos móviles, PDA o PC, ya que es una tecnología complementaria para proporcionar otros servicios, como puede ser la identificación y validación de personas.

Proyectos con NFC

En España, se ha utilizado la tecnología NFC como método de pago en un proyecto de colaboración entre Visa, La Caixa y Telefónica.

Zigbee

Zigbee es un estándar de comunicaciones inalámbricas, regulado por el grupo de trabajo IEEE 802.15.4 en el 2004, que permite habilitar redes inalámbricas con capacidades de control, y monitorizar que sean seguras, de bajo consumo energético y de bajo coste de procesador, de manera bidireccional.

Zigbee

El origen del nombre *Zigbee* surgió de una colmena de abejas que se comunicaban dando vueltas alrededor de su colmena.

ZigBee es promovida por la ZigBee Alliance, una comunidad internacional de más de cien compañías, como Motorola, Mitsubishi, Philips, Samsung, Honeywell y Siemens, entre otras. De hecho, ZigBee no es una tecnología, sino un conjunto estandarizado de soluciones que pueden ser implementadas por cualquier fabricante.

Enlace de interés

Para saber más sobre el estándar Zigbee, podéis visitar la dirección siguiente:
www.zigbee.org/

2.1.2. Redes locales inalámbricas (WLAN)

Una WLAN es una red de cobertura geográfica limitada, velocidad de transmisión relativamente alta, bajo nivel de errores y administrada de manera privada, que se comunica básicamente mediante microondas.

Ventajas de una WLAN

La necesidad de una WLAN no se justifica por una posible mejora en ancho de banda o en fiabilidad, seguridad o eficiencia de las comunicaciones, sino por la comodidad que proporciona al usuario y la movilidad que le permite, y también por su fácil y rápida instalación.

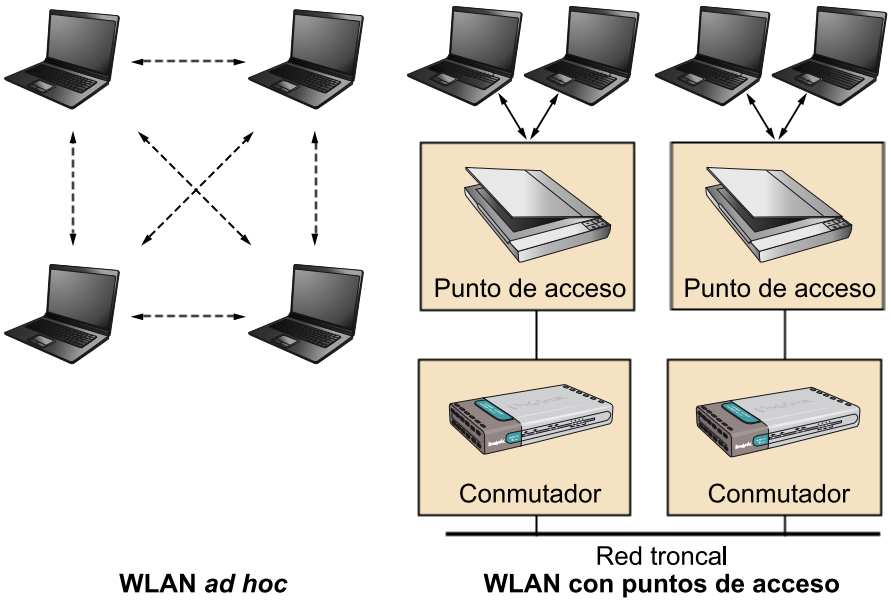
Las WLAN son una extensión y/o una alternativa a las LAN con cables. Los usuarios de una WLAN pueden acceder a los recursos que les ofrece la LAN sin tener que depender de infraestructuras de red (cableado, conectores, etc.).

La gran difusión de las WLAN se debe a las importantes ventajas que presentan respecto a las LAN:

- **Movilidad**: los usuarios de una WLAN pueden acceder a información en tiempo real desde cualquier lugar de la organización.
- **Instalación simple**: no hay que preocuparse por la instalación de cables dentro del radio de cobertura.
- **Flexibilidad**: permite acceder a lugares que una LAN cableada no alcanzaría nunca.

- **Bajo coste**: aunque el coste inicial de instalación de las WLAN puede ser superior a las LAN con cable, a largo plazo puede suponer un ahorro, sobre todo en entornos con cambios frecuentes de ubicación de los dispositivos.

- **Escalabilidad**: las WLAN se pueden configurar con diferentes topologías de una manera sencilla según la necesidad del entorno. Podemos tener las WLAN *ad hoc* (donde los dispositivos se van añadiendo a la red) y las WLAN con puntos de acceso conectados a la red principal.



A pesar de las ventajas mencionadas anteriormente, las WLAN tienen una serie de limitaciones y requisitos, como:

- **Velocidad**: las WLAN deben poder transmitir información a velocidades comparables a las LAN (más de 500 Mbps).

- **Retardos**: son importantes en cualquier aplicación, pero especialmente en las transmisiones inalámbricas.

- **Accesos difíciles**: dentro de un edificio podemos encontrar factores que amortiguan la señal. Un dispositivo móvil puede recibir mucha menos potencia que otro.

- **Consumo**: los dispositivos móviles se suelen alimentar con baterías; por lo tanto, hay que diseñarlos para que tengan un consumo eficiente (modo reposo, modo bajo consumo, poco gasto en la transmisión de paquetes, etc.).

- **Máximo número de nodos y máxima cobertura**: una WLAN puede necesitar soportar centenares de nodos. El área de cobertura típica de una

WLAN es de entre 10 y 100 m^2, lo que implica retardos de propagación inferiores a 1.000 nseg.

- **Seguridad**: el medio en el que se transmite la información (ondas electromagnéticas) es abierto para cualquiera que esté en el radio de cobertura. Para garantizar la seguridad, se utilizan algoritmos de cifrado.

- **Interferencias**: se pueden producir a causa de dos transferencias simultáneas (colisiones) o de dos emisores que comparten la misma banda de frecuencia. Las colisiones también se producen cuando varias estaciones que esperan que el canal esté libre empiezan las transmisiones al mismo tiempo. A diferencia de las redes locales con hilos, en las WLAN se produce un efecto de nodo oculto que conlleva un aumento de colisiones.

Las tecnologías más utilizadas de WLAN son principalmente las distintas variantes del IEEE 802.11; aunque también existen otras, como la HIPERLAN.

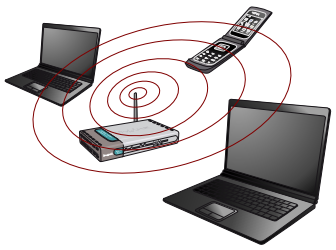
IEEE 802.11

El IEEE 802.11 es una familia de estándares para redes locales inalámbricas desarrollada por el IEEE, que fue definida en 1997 (en el año 1999 se definieron los estándares 802.11a y 802.11b). El estándar garantiza la interoperabilidad entre diferentes fabricantes. Es decir, por ejemplo, que una tarjeta WLAN para PC de un fabricante funcione con un punto de acceso de otro fabricante.

Enlace de interés

Podéis obtener más información sobre este estándar en la dirección siguiente:
http://grouper.ieee.org/groups/802/11/

El estándar 802.11 describe la funcionalidad de las capas y subcapas y las relaciones entre ellas, pero no especifica cómo se tienen que hacer; solo indica cómo se debe comportar el equipo y deja vía libre al fabricante en la manera de implementarlo.



El objetivo principal del estándar 802.11 es garantizar la funcionalidad de las aplicaciones sin tener que considerar si la comunicación es o no inalámbrica.

El estándar 802.11 es una familia de especificaciones, entre las cuales destacamos las siguientes:

- **IEEE 802.11a**: soporta velocidades de hasta 54 Mbps y utiliza la banda de frecuencias de 5 GHz. Este protocolo está orientado a la transmisión de paquetes, pero no soporta funciones de calidad de servicio.

- **IEEE 802.11b (inicialmente denominado Wi-Fi)**: soporta velocidades de hasta 11 Mbps y utiliza la banda de frecuencias de 2,4 GHz.

- **IEEE 802.11g**: soporta velocidades de hasta 54 Mbps. Es una evolución del IEEE 802.11b y utiliza la misma banda de frecuencias de 2,4 GHz.

- **IEEE 802.11i**: se creó para superar la vulnerabilidad de seguridad para protocolos de autenticación y de codificación. El estándar incluye los protocolos 802.1x, TKIP y AES y se implementa con WPA2.

- **IEEE 802.11n**: soporta velocidades de hasta 600 Mbps y puede trabajar en dos bandas de frecuencia: 2,4 GHz (la que utilizan 802.11b y 802.11g) y 5 GHz (la que utiliza 802.11a). 802.11n es compatible con dispositivos basados en todas las especificaciones anteriores de 802.11. El hecho de que trabaje en la banda de 5 GHz le permite alcanzar un mayor rendimiento, ya que está menos congestionada.

HiperRLAN

El *high performance radio local area network* (HiperLAN) es un estándar de redes locales inalámbricas desarrollado por el ETSI.

La primera versión del estándar, HiperLAN1 (HiperLAN Type 1), surgió en el año 1996 y admitía velocidades de hasta 20 Mbps. La evolución de este estándar, que apareció en el año 2000, se denomina HiperLAN2 (HiperLAN Type 2) y admite velocidades de hasta 54 Mbps. Los dos estándares operan en la banda de frecuencias de 5 GHz.

2.1.3. Redes de gran alcance inalámbricas (WWAN)

Las WWAN permiten la conexión de redes y usuarios de zonas geográficamente distantes. Podemos distinguir dos tipos:

WWAN e IEEE

El grupo de trabajo 802.16 del IEEE define los estándares WWAN.

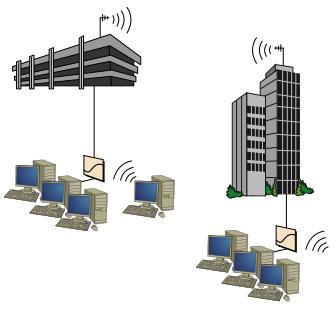
- WWAN fijas, que utilizan radioenlace o satélite.
- WWAN móviles, que utilizan las compañías u otros servicios públicos en la transmisión y recepción de señales.

Sin ningún tipo de duda, las redes WWAN móviles (MWWAN) son las que han vivido una expansión más espectacular en los últimos años. Actualmente las MWWAN son el sistema de comunicación inalámbrico más utilizado, ya que es el que utilizan las operadoras de telefonía móvil y cuenta con más de 5.000 millones de usuarios en todo el mundo.

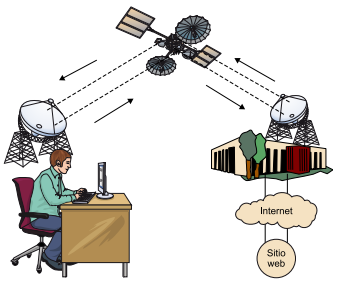
WWAN fijas (FWWAN)

Las redes WWAN fijas pueden utilizar dos tecnologías:

- **Radioenlace**. Utilizando radioenlaces se pueden conectar redes separadas geográficamente con diferentes bandas del espectro electromagnético (infrarrojos, microondas, láser, etc.), que pueden ser de punto a punto o de punto a multipunto.



- **Satélite**. Las comunicaciones por satélite cubren una gran superficie de la Tierra, tienen un gran ancho de banda y el coste de la transmisión es independiente de la distancia; presentan el inconveniente de los retardos de propagación de la señal.



Actualmente, la mayor parte de las redes de satélite se utilizan para la difusión de televisión. El uso de estas redes para la transmisión de datos inalámbricas es muy limitado, dado que es necesario tener en cuenta los grandes gastos que conllevan en equipamiento, los problemas del retardo que se produce al propagarse la señal y el coste elevado por minuto de transmisión.

WWAN móvil (MWWAN)

En las redes MWWAN el terminal que envía y recibe la información está en movimiento. En estas redes normalmente hay muchos usuarios conectados simultáneamente (acceso múltiple) que utilizan los servicios.

Actualmente en Europa existen diferentes tecnologías de MWWAN, agrupadas por generaciones, donde las más destacadas son las cinco siguientes:

1) 2G (segunda generación). Tecnología de segunda generación, utilizada para describir las redes móviles digitales, como las GSM, que sustituyeron a las redes móviles analógicas de primera generación. Básicamente estaban diseñadas para comunicaciones de voz, mensajería instantánea (SMS) y, esporádicamente, para transmisión de datos básicos que requieren muy poco ancho de banda. La generación abarca el sistema GSM:

- **GSM**[11]. El Group Special Mobile fue el organismo que se encargó de la configuración técnica de una normativa de transmisión y recepción para la telefonía móvil. En Europa, las bandas de frecuencias ISM que se utilizan son 900 MHz y 1.800 MHz. Esta tecnología apareció en el año 1990 con una velocidad de transmisión de 9,6 kbps. GSM opera por comunicación de circuitos; esto quiere decir que existe una fase de establecimiento de la conexión que añade tiempo de espera y que la llamada siempre estará abierta, aunque no haya transferencia de datos, mientras no se cierre la conexión.

(11)GSM: *global system for mobile communications*.

2) 2.5G (segunda generación y media). Considerada una tecnología intermedia entre 2G y 3G basada en las actualizaciones tecnológicas de las redes móviles GSM para aumentar la velocidad de transmisión de datos y su eficacia. La generación abarca los sistemas GPRS y EDGE:

- **GPRS**[12]. Es una técnica de conmutación de paquetes que empezó a utilizarse en el 2001 y que se integró con la estructura actual de redes GSM. Esta tecnología permite una velocidad de datos de entre 56 y 115 kbps. Sus ventajas son múltiples y se aplican fundamentalmente a las transmisiones de datos que requieren tráfico discontinuo, como por ejemplo Internet y

(12)GPRS: *general packet radio service*.

mensajería electrónica (SMS y MMS). Con esta tecnología, desaparece el concepto de tiempo de conexión y dejan paso al de cantidad de información transmitida, y se pasa de conmutación de circuitos a conmutación de paquetes. Los proveedores de servicio de telefonía móvil podrán facturar por los paquetes realmente enviados y recibidos. El ancho de banda podrá ser entregado a la carta, en función de las necesidades de la comunicación.

- **EDGE**[13]. También conocida como EGPRS (Enhanced GPRS), es una tecnología que apareció en el 2003 y considerada una evolución del GPRS. EDGE proporciona un ancho de banda superior a la de GPRS, entre 236 y 384 kbps, que permite ejecutar aplicaciones que requieren una mayor velocidad de transferencia de datos, como vídeo y otros servicios multimedia.

(13)EDGE: *enhanced data rates for gsm of evolution*.

3) 3G (tercera generación). Las tecnologías de 3G son la respuesta a la especificación IMT-2000 de la Unión Internacional de Telecomunicaciones (ITU) para disponer de banda ancha en telefonía móvil y transmitir un volumen de datos importante mediante la red. Con la tercera generación serán posibles las videoconferencias, descargar vídeos, ver televisión en tiempo real y poder realizar la mayoría de las operaciones desde el móvil. La generación abarca el sistema UMTS:

- **UMTS**[14]. El estándar UMTS está basado en la tecnología WCDMA. UMTS está gestionado por la organización 3GPP versión 4, también responsable de GSM, GPRS y EDGE. UMTS se comercializó por primera vez en el 2005 y su velocidad máxima de transmisión de datos es 1,92 Mbps.

(14)UMTS: *universal mobile telecommunications system*.

3GPP

El Third Generation Partnership Project (3GPP) se creó para conducir la preparación y el mantenimiento de una gama completa de especificaciones técnicas aplicables a un sistema móvil 3G basado en las redes GSM centrales evolucionadas.

4) 3.5G (tercera generación y media). De la misma manera que el 2.5G, es considerada una tecnología intermedia entre 3G y 4G, con el principal objetivo de aumentar considerablemente la velocidad de transmisión de datos por las necesidades actuales de los clientes consumidores. Es, por lo tanto, la evolución de 3G y se considera el paso previo de la cuarta generación 4G. La generación abarca los sistemas HSPA y HSDPA:

(15)HSPA: *high speed packet access*.

- **HSPA**[15]. Es la combinación de tecnologías posteriores y complementarias a 3G, como HSDPA o HSUPA. Teóricamente, admite velocidades de hasta 14,4 Mbps en bajada y hasta 2 Mbps en subida, dependiendo del estado o la saturación la red y de su implantación.

- **HSDPA**[16]. Es la optimización de la tecnología espectral UMTS/WCDMA, incluida en las especificaciones de 3GPP versión 5 y consiste en un nuevo canal compartido en el enlace descendente (*downlink*) que mejora significativamente la capacidad máxima de transferencia de información hasta llegar a tasas de 14,4 Mbps, soportando tasas de transmisión media próximas a 1 Mbps. Es totalmente compatible con UMTS y la mayoría de los proveedores UMTS dan soporte a esta tecnología.

(16)HSDPA: *high speed downlink packet access*.

WCDMS

En el *wideband code division multiple access* (WCDMS) –o acceso múltiple por división de código de banda ancha– los datos y la voz se transmiten en banda ancha, divididos en paquetes antes de la transmisión. Estos paquetes se reúnen en el terminal antes de presentar la información.

5) 4G (cuarta generación). El WWRF[17] define 4G como una integración de red que funciona con la tecnología de Internet donde toda la red es IP, combinándola con otros usos y tecnologías, como WiFi y WiMAX. En estos momentos, 4G no es una tecnología o estándar definido, sino una colección de tecnologías y protocolos que permiten el máximo rendimiento y con una red inalámbrica más barata. 4G incluye técnicas inalámbricas de alto rendimiento, como MIMO[18] y para el acceso radio abandona el acceso tipo CDMA característico de UMTS (3G) para pasar a OFDMA[19] para optimizar el acceso. La generación abarca los sistemas LTE y WiMax:

(17)WWRF: Wireless World Research Forum.

(18)MIMO: *multiple-input multiple-output*.

(19)OFDMA: *orthogonal frecuency division multiple access*.

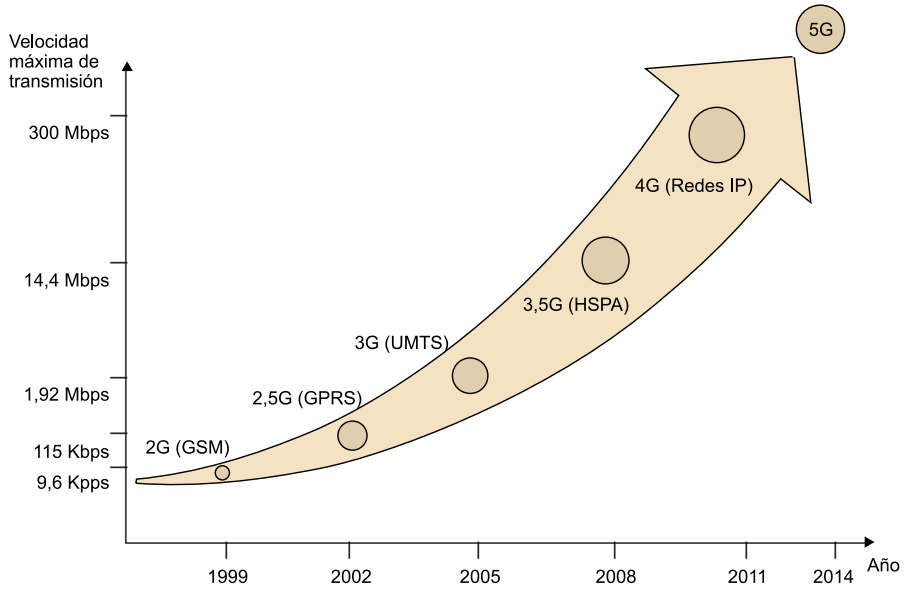
- **LTE**[20]. Es el estándar de la norma 3GPP versión 8, 9 y 10, definida como una evolución de la norma 3GPP UMTS (3G) y un nuevo concepto de arquitectura evolutiva (4G). LTE es la clave para el despegue de Internet móvil, ya que posibilita la transmisión de datos a más de 300 Mbps en movimiento, lo que permite la transmisión de vídeos o TV de alta definición.

(20)LTE: *long term evolution*.

- **WIMAX**[21]. Es una tecnología, entre WLAN y WWLAN, que permite hacer conexiones a grandes distancias, con grandes anchos de banda y sin necesitar línea de visión directa entre antenas. WiMAX cumple los estándares IEEE 802.16 y es compatible con otros estándares, como el IEE 802.11, para establecer sistemas de telecomunicaciones conjuntos.

(21)WiMax: *worldwide interoperability for microwave access*.

Evolución de las tecnologías móviles MWWAN



2.2. Ventajas de las comunicaciones inalámbricas respecto a las tradicionales

A continuación, detallamos algunas de las ventajas que supone la utilización de la tecnología inalámbrica, respecto a la comunicación tradicional con cable:

- **Accesibilidad y flexibilidad**. Las comunicaciones inalámbricas llegan a lugares donde los cables no tienen acceso.

- **Coste**. Las comunicaciones inalámbricas nos ahorran el coste asociado a la instalación del cableado y los derivados de los cambios de entorno físico, que podrían ser todavía más importantes.

- **Movilidad**. Las comunicaciones inalámbricas permiten tener información en tiempo real y en cualquier lugar del mundo. Esta funcionalidad puede permitir a muchas empresas mejorar su productividad y sus posibilidades de negocio.

- **Comodidad**. El hecho de poder prescindir de los cables que conectan los dispositivos hace que con el uso de comunicaciones inalámbricas se adquiera una importante comodidad.

La importancia de la comodidad

Solo con los dispositivos de entrada y salida de un ordenador podemos encontrar hasta once conexiones: teclado, ratón, monitor, altavoces, micrófono, cable de red, módem, cámara digital, impresora, escáner y palanca de control (*joystick*). Si eso lo multiplicamos por el número de ordenadores que tiene una empresa, entenderemos mejor qué queremos decir al hablar de comodidad.

- **Escalabilidad**. Las comunicaciones sin cables se adaptan fácilmente a los cambios de topología de la red y, además, la reubicación de los terminales se facilita enormemente.

2.3. Limitaciones de las comunicaciones inalámbricas respecto a las tradicionales

Las limitaciones principales que podemos encontrar en las comunicaciones inalámbricas son las siguientes:

- **Consumo**. Los terminales móviles suelen trabajar con baterías que limitan la potencia de transmisión de los dispositivos, lo que repercute directamente en el alcance de las redes.

- **Capacidad de transferencia limitada**. El espectro electromagnético es un recurso limitado.

- **Calidad**. Las transferencias inalámbricas se ven sometidas a interferencias y ruidos.

- **Seguridad**. La utilización del espectro electromagnético como medio de comunicación implica que cualquier persona puede acceder a la información sin ningún tipo de limitación física.

3. Pasado, presente y futuro de las comunicaciones inalámbricas

3.1. El pasado de las comunicaciones inalámbricas

Si nos remontamos a miles de años atrás, comprobaremos que nuestros antepasados ya utilizaban el humo como sistema de comunicación inalámbrico para largas distancias.

Evidentemente, no empezaremos nuestro repaso desde tan lejos; nos fijaremos en acontecimientos relacionados con la electricidad y la electrónica, que son los antecedentes que más nos interesan en relación con el tema que nos ocupa:

- En el año 1896 el italiano Guglielmo Marconi transmitió y recibió la primera señal de radio local en Italia.

- En el año 1924 la NBC estableció la primera red de radio con veinticuatro estaciones.

- En el año 1925 se realizó la primera demostración de televisión.

- Desde el año 1947 se hacían pruebas del servicio de telefonía móvil, pero hasta 1983 no se comercializó el primero.

- El origen de las WLAN se remonta al año 1979, cuando se publicaron los resultados de un experimento hecho por ingenieros de IBM que crearon una red local con infrarrojos en una fábrica.

- A principios de la década de los noventa, un consorcio de empresas de primera línea (IBM, Intel, Toshiba, Ericsson y Nokia) crearon la tecnología Bluetooth, que, posteriormente, el lEEE ha incorporado al estándar 802.15.1.

- En el año 1994 apareció el primer borrador del estándar IEEE 802.11.

- En el año 1995 MoviStar y Airtel empezaban a operar sobre GSM en España; tres años más tarde, Amena se incorporó a esta tecnología.

- En el año 2001 se proporcionó servicio sobre GPRS en Europa.

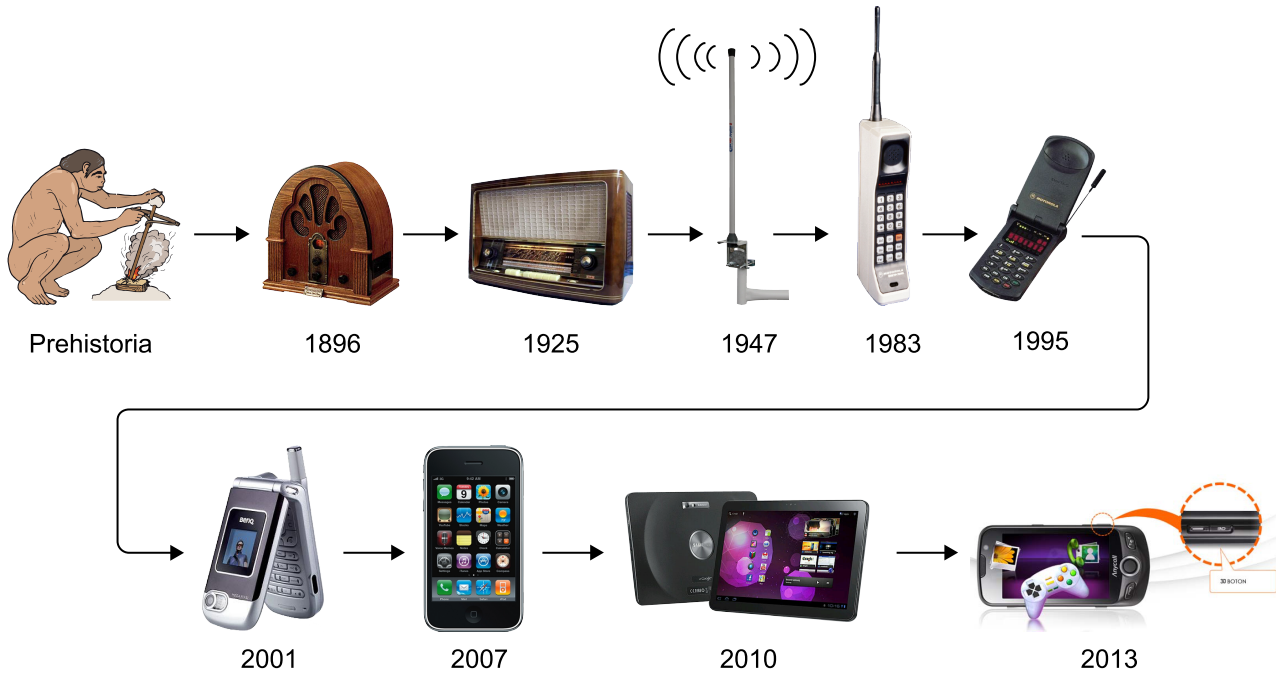
- En el año 2002 se hizo el lanzamiento comercial de 3G (UMTS) en la mayoría de los países de Europa.

- Desde el 2005 las redes UMTS evolucionan a partir de las tecnologías HS-PA con el principal objetivo de aumentar la velocidad de transmisión de datos.

- En el año 2006 apareció el primer borrador de 802.11n, que soportaba velocidades próximas a 600 Mbps.

- En el año 2011 aparece el estándar Bluetooth 4.0, que destaca por la reducción significativa del consumo de batería y por una velocidad de transmisión máxima superior a 300 Mbps.

- En el año 2012 se empiezan a hacer las primeras pruebas sobre 4G.



Evolución de las comunicaciones inalámbricas

3.2. Presente y futuro de las comunicaciones inalámbricas

Las comunicaciones inalámbricas han tenido un crecimiento espectacular en los últimos años. En este momento hay más de cinco mil millones de contratos de comunicaciones móviles MWWLAN en todo el mundo; hecho que ha permitido un aumento muy significativo de las aplicaciones y servicios que se realizan a partir de este medio de comunicación, que está al alcance de la mayoría de las personas. La tecnología móvil es, con mucha diferencia, el sector de las tecnologías de la información y las comunicaciones que más cambios está viviendo en los últimos veinte años.

Paralelamente, los dispositivos móviles, como smartphone o tabletas, también se han beneficiado de este crecimiento y actualmente ya se pueden encontrar en el mercado una gran variedad de dispositivos móviles que permiten ejecutar y visualizar aplicaciones de altas prestaciones en red, como por ejemplo, televisión en alta definición y juegos 3D, que necesitan gran capacidad de cálculo.

De todas maneras es importante destacar que en estos momentos estamos en plena fase de evolución de esta tecnología de comunicación, ya que se espera que en el año 2020 haya alrededor de cincuenta mil contratos de comunicación móvil con unas prestaciones de ancho de banda de datos que puede llegar a 1Gbps, hecho que todavía disparará más las posibilidades de nuevas aplicaciones que permitirán generar un elevado volumen de negocio a partir de esta infraestructura.

En los últimos años, dos de las nuevas profesiones más demandadas son: el experto en tecnología móvil y el desarrollador de aplicaciones móviles. Estos profesionales tienen que ser capaces de construir aplicaciones móviles a medida, diseñar estrategias de negocio con una arquitectura móvil y garantizar la seguridad de la información entre los diferentes dispositivos.

Con respecto al desarrollo de aplicaciones móviles, actualmente existen dos tendencias. La primera requiere conocimientos de programación específicos dependiendo de la plataforma nativa del dispositivo móvil, por ejemplo, *Objective-C* para el iOS de iPhone, Java para Android o Java para BlackBerry. La segunda tendencia es programar en *HTML5*, que permite el desarrollo web multi-plataforma; es decir, que el mismo código de programación se puede utilizar en iOS de iPhone, Android o BlackBerry.

Resumen

Uno de los acontecimientos más importantes dentro de las tecnologías de la información de los últimos años ha sido la expansión de las comunicaciones inalámbricas como método de intercambio de información. Una de sus principales ventajas recae en la no dependencia de cableado, ya que el punto de entrada a la red de comunicaciones no se encuentra ligado a una ubicación física. El medio de transmisión ya está listo, sin que sea necesaria la creación de la infraestructura previa.

Poco a poco, esta tecnología ha ganado mucho protagonismo dentro de las posibilidades de realizar negocio, y han sido los servicios y las aplicaciones que se desarrollan sobre ellas los que han marcado el presente y determinarán también su futuro.

Actividades

1. Buscad información de las novedades de los diferentes grupos de trabajo del IEEE que se ocupan de la estandarización de las redes inalámbricas (IEE 802.15, IEEE 802.11, IEEE 802.16).

2. Buscad información sobre qué tecnologías inalámbricas de las que hemos visto en este módulo están disponibles actualmente para las grandes operadoras de telefonía, y a qué velocidad se puede transmitir información mediante cada una de ellas.

3. ¿Qué servicios y qué aplicaciones pensáis que pueden llegar a ser un valor añadido gracias a utilizar las comunicaciones inalámbricas?

Glosario

3GPP3 (*third generation partnership project*) *m* Tecnología que se creó para conducir la preparación y el mantenimiento de una gama completa de especificaciones técnicas aplicables para un sistema móvil 3G basado en las redes GSM centrales evolucionadas.

bandas IMS (*industrial, scientific and medical bands*) *f pl* Bandas de frecuencias autorizadas por organismos internacionales en las que se tiene en cuenta que la potencia de estas frecuencias esté dentro de un margen no perjudicial para la salud.

Bluetooth *m* Tecnología inalámbrica de alcance muy limitado, normalmente no superior a 10 metros, que permite conectar dos dispositivos a una velocidad muy aceptable. Fue creado por un consorcio de empresas de primera línea, como IBM, Intel, Toshiba, Ericsson y Nokia.

DETC (*digital enhanced cordless telecommunicatios*) *f* Tecnología de área personal que se utiliza con frecuencia en los aparatos de telefonía inalámbrica de uso doméstico, que suelen operar con un rango no superior a 50 metros.

EDGE (*enhanced data rates for GSM of evolution*) *m* Tecnología que proporciona un ancho de banda superior a GPRS, entre 236 kbps y 384 kbps, que permite ejecutar aplicaciones que requieren una mayor velocidad de transferencia de datos, como vídeo y otros servicios multimedia.

espectro electromagnético *m* Rango de frecuencias de todas las ondas electromagnéticas que pueden propagarse a través del espacio libre, ordenadas según su longitud de onda y su frecuencia.

FCC (Federal Communications Comimission) *f* Agencia federal de Estados Unidos responsable de regular la industria de telecomunicaciones, incluida la gestión de frecuencias y el desarrollo de reglas de utilización del espectro.

GSM (*global system for mobile communications*) *m* Estándar WWAN de segunda generación (2G) que permite la transmisión de voz, datos y mensajes cortos SMS (*short message system*).

GPRS (*general packet radio service*) *m* Estándar WWAN de generación 2.5 que utiliza la infraestructura de radio de GSM para alcanzar velocidades de transferencia de 115 Kbps. Permite servicios de pago en función de la cantidad de información enviada y, gracias a su tecnología de conmutación de paquetes, permite estar siempre conectado a los recursos de la Red.

HomeRF *m* Estándar de redes de área personal que permite la transferencia de datos y voz inalámbricamente; facilita la integración de dispositivos como el ordenador y la telefonía.

HSDPA (*high speed packet access*) *m* Optimización de la tecnología espectral UMTS/WCDMA, incluida en las especificaciones de 3GPP versión 5, que consiste en un nuevo canal compartido en el enlace descendente (*downlink*) que mejora significativamente la capacidad máxima de transferencia de información hasta llegar a tasas de 14,4 Mbps, soportando tasas de transmisión media próximas a 1 Mbps.

HSPA (*high speed downlink packet access*) *m* Combinación de tecnologías posteriores y complementarias de 3G, como HSDPA o HSUPA.

IEEE (Institute of Electrical and Electronics Engineers) *m* Asociación internacional de ingenieros, formada por más 300.000 miembros y más de 300 países, que regula los estándares de comunicación.

infrarrojo (IR) *m* Rango de frecuencias superiores a 300 GHz del espectro electromagnético, utilizadas en comunicaciones punto a punto de corto recorrido; son muy direccionables y no pueden atravesar obstáculos.

IrDa *f* Asociación de 160 compañías que se centran en elaborar estándares para comunicaciones inalámbricas por infrarrojos (IR).

ITU (Internacional Telecommunications Union) *f* Órgano internacional responsable de gestionar las diferentes frecuencias del espectro electromagnético.

LTE (*long term evolution*) *m* Evolución de la norma 3GPP UMTS (3G) y un nuevo concepto de arquitectura evolutiva (4G).

microondas (MW) *f pl* Parte del espectro electromagnético de frecuencias superiores a 1 GHz e inferiores a 300 GHz, donde se encuentran muchas comunicaciones inalámbricas, entre otras las WLAN y las comunicaciones por satélite.

NFC (*near field communication*) *m* Tecnología que permite la transmisión de datos de una manera simple entre diferentes dispositivos mediante un enlace de radiofrecuencia en la banda ISM de 13,56 MHz.

radiofrecuencia (RF) *f* Parte del espectro electromagnético entre las frecuencias de 10 MHz y 300 MHz, donde se encuentran, entre otras, las señales de radio y televisión.

UMTS (*universal mobile telephony system*) *m* Estándar WWAN de tercera generación (3G) que permite la transmisión inalámbrica de servicios multimedia y acceso a Internet a alta velocidad.

WCDMA (*wideband code division multiple access*) *m* Tecnología en que los datos y la voz se transmiten en banda ancha, divididos en paquetes antes de la transmisión. Estos paquetes se reúnen en el terminal antes de presentar la información.

WiMAX (*worldwide interoperability for microwave access*) *m* Tecnología que permite hacer conexiones a grandes distancias, con grandes anchos de banda y sin necesitar línea de visión directa entre antenas.

WLAN (*wireless local area networks*) *m pl* Redes locales inalámbricas que permiten la transmisión de datos digitales inalámbricamente entre dispositivos (ordenadores, periféricos, etc.) fijos y/o móviles. Son un complemento y/o una alternativa a las redes locales con hilos. Tienen un alcance medio (centenares de metros) y han de poder operar con gran velocidad (comparable con las LAN con hilos), fiabilidad y seguridad. Tienen un coste superior a las WPAN.

WPAN (*wireless personal area networks*) *m pl* Redes personales inalámbricas que permiten que los dispositivos personales (teléfonos móviles, agendas electrónicas, accesorios, etc.) se comuniquen. Tienen un alcance limitado (pocos metros), de bajo coste y los dispositivos generalmente están dotados de baterías y gran movilidad.

WWAN (*wireless wide area networks*) *m pl* Redes de gran alcance inalámbricas que permiten la conexión de dispositivos geográficamente muy alejados. Los dispositivos pueden ser fijos (se utilizan radioenlaces o satélites) o móviles (redes GSM, GPRS o UMTS). Dado que tienen un gran alcance, puede haber muchos usuarios conectados a los servicios simultáneamente.

Zigbee *m* Tecnología que permite habilitar redes inalámbricas con capacidades de control y monitorización que sean seguras, de bajo consumo energético y de bajo coste de procesador, de manera bidireccional.

Introducción a los dispositivos móviles

Julián David Morillo Pozo

PID_00176753

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

Si pensamos en dispositivos móviles, lo primero que nos viene a la cabeza es un teléfono móvil. Pero en la actualidad son varios los dispositivos móviles disponibles en el mercado: PC portátiles, PocketPC, tabletas, etc.

Esta diversidad comporta una importante problemática para quien debe programarlos, ya que cada uno tiene unas características particulares: dispone de una memoria determinada o ha de soportar un lenguaje y un entorno específicos.

Por todo ello, en este módulo veréis las características generales de los dispositivos móviles para, después, clasificarlos. A nivel más técnico, veremos los componentes específicos que pueden tener y repasaremos las redes a las que pueden acceder.

Objetivos

Mediante el estudio de este módulo, pretendemos que consigáis los siguientes objetivos:

1. Que entendáis qué son los dispositivos móviles y cuáles son sus características.

2. Que conozcáis los tipos de dispositivos móviles existentes.

3. Que tengáis una visión general de las diferencias que puede haber entre dispositivos móviles en función de sus características.

4. Que tengáis una visión histórica de la evolución de los dispositivos móviles.

1. Características generales de los dispositivos móviles

Una gran cantidad de dispositivos electrónicos se clasifican actualmente como dispositivos móviles, desde teléfonos hasta *tablets*, pasando por dispositivos como lectores de RFID[1]. Con tanta tecnología clasificada como móvil, puede resultar complicado determinar cuáles son las características de los dispositivos móviles.

[(1)]RFID (*radio frequency identification*): identificación por radiofrecuencia.

Antes de describir detalladamente algunos dispositivos móviles, vamos a concretar el concepto de dispositivo tratado en esta asignatura. A continuación detallamos las características esenciales que tienen los dispositivos móviles:

Terminología

En inglés existe una amplia gama de términos para referirse a los dispositivos: *information device*, *information appliance*, *consumer electronic*, *small device*, *handheld* o *palmtop*, entre otros.

- Son aparatos pequeños.
- La mayoría de estos aparatos se pueden transportar en el bolsillo del propietario o en un pequeño bolso.
- Tienen capacidad de procesamiento.
- Tienen conexión permanente o intermitente a una red.
- Tienen memoria (RAM, tarjetas MicroSD, *flash*, etc.).
- Normalmente se asocian al uso individual de una persona, tanto en posesión como en operación, la cual puede adaptarlos a su gusto.
- Tienen una alta capacidad de interacción mediante la pantalla o el teclado.

En la mayoría de los casos, un dispositivo móvil puede definirse con cuatro características que lo diferencian de otros dispositivos que, aunque pudieran parecer similares, carecen de algunas de las características de los verdaderos dispositivos móviles. Estas cuatro características son:

1) movilidad
2) tamaño reducido
3) comunicación inalámbrica
4) interacción con las personas

En los siguientes subapartados os explicamos en detalle estas características.

1.1. Movilidad

Se entiende por movilidad la cualidad de un dispositivo para ser transportado o movido con frecuencia y facilidad. Por tanto, el concepto de movilidad es una característica básica. Los dispositivos móviles son aquellos que son lo suficientemente pequeños como para ser transportados y utilizados durante su transporte.

Cualquier dispositivo móvil debería funcionar y poder ser usado de forma fiable mientras nos movemos, independientemente de la proximidad de una fuente de energía (enchufe) o de una conexión física a Internet. Para facilitar la portabilidad, los dispositivos móviles suelen contener baterías recargables, que permiten varias horas o más de operación sin necesidad de acceso a un cargador o a una fuente de energía externa. Aunque son móviles, estos dispositivos pueden sincronizarse con un sistema de sobremesa para actualizar aplicaciones y datos.

Ejemplo de movilidad

Un comercial de una empresa farmacéutica carga en su PDA, antes de salir de la oficina, los datos de los clientes que tiene que visitar. Durante su visita, actualiza o modifica la información, y una vez terminada su ruta, ya en la oficina, actualiza los datos en la aplicación corporativa.

1.2. Tamaño reducido

Se entiende por tamaño reducido la cualidad de un dispositivo móvil de ser fácilmente usado con una o dos manos sin necesidad de ninguna ayuda o soporte externo. El tamaño reducido también permite transportar el dispositivo cómodamente por parte de una persona.

Como hemos visto anteriormente, existen varios términos en inglés que hacen referencia a los dispositivos móviles. En concreto, se denomina a algunos dispositivos *handhelds* o *palmtops* debido a que, en mayor o menor medida, sus dimensiones son parecidas a las de una mano. Por el contrario, otros dispositivos móviles ligeramente más grandes, como podría ser el caso de los *tablets*, no suelen denominarse así. Un dispositivo móvil típico lo podemos llevar con una mano y cabe en un bolsillo o en un pequeño bolso. Algunos dispositivos móviles se pueden desplegar o desdoblar desde un modo portable o compacto a un tamaño ligeramente superior y descubrir teclados o pantallas más grandes. Los dispositivos móviles pueden tener pantallas táctiles o pequeños teclados numéricos para recibir datos de entrada y mantener, al mismo tiempo, su tamaño pequeño e independencia de dispositivos de interfaz externos.

Netbooks

Los *netbooks* se consideran a menudo dispositivos móviles debido a su similitud en cuanto a funcionalidad, pero si el tamaño del dispositivo impide un manejo cómodo (por ejemplo, sin la ayuda de una mesa) o limita la portabilidad, no se puede considerar un verdadero dispositivo móvil y, por lo tanto, no lo consideraremos como tal en esta documentación.

1.3. Comunicación inalámbrica

Otro concepto importante es el término *inalámbrico*[2]. Por comunicación inalámbrica se entiende la capacidad que tiene un dispositivo de enviar o recibir datos sin la necesidad de un enlace cableado.

(2)En inglés, *wireless*.

Por lo tanto, un dispositivo inalámbrico es aquel capaz de comunicarse o de acceder a una red sin cables (por ejemplo, un teléfono móvil o una PDA).

Los dispositivos móviles son, normalmente, capaces de comunicarse con otros dispositivos similares, así como con otros ordenadores y sistemas. Un dispositivo móvil típico es capaz de acceder a Internet, ya sea mediante redes Bluetooth o WiFi, y muchos modelos están también equipados para acceder a redes de datos MWWAN, así como a otras redes de datos inalámbricas.

Ejemplo

Un comercial de una empresa farmacéutica podría consultar algún dato de un cliente justo antes de visitarlo si su dispositivo tiene acceso a alguna red.

Los mensajes de texto, los correos electrónicos y todo tipo de contenidos multimedia son formas estándar de comunicación mediante dispositivos móviles, que además permiten realizar y recibir llamadas telefónicas. Algunos dispositivos móviles especializados, como los dispositivos RFID y los lectores de códigos de barras, se comunican directamente con un dispositivo central.

Los conceptos de *móvil* y *sin cables* muchas veces se confunden. Por ejemplo, un dispositivo móvil con datos en él y aplicaciones para gestionar dichos datos puede ser móvil, pero no tiene por qué ser inalámbrico, ya que puede necesitar un cable para conectarse al ordenador y obtener o enviar datos y aplicaciones. Si el dispositivo puede conectarse a una red inalámbrica para obtener datos mientras vamos por la calle o estamos en la oficina, entonces será también inalámbrico.

1.4. Interacción con las personas

Se entiende por interacción el proceso de uso que establece un usuario con un dispositivo. Entre otros factores, en el diseño de la interacción intervienen disciplinas como la usabilidad y la ergonomía.

Un usuario realiza la interacción con un dispositivo móvil mediante la interfaz de usuario. Existen grandes diferencias en cuanto a la interacción que hay entre un PC[3] y los dispositivos móviles (e, incluso, entre distintos dispositivos móviles). Por ejemplo, la forma de utilizar los enlaces (algo tan básico en Internet) es muy distinta entre un usuario que maneja un puntero de PDA y un usuario que avanza con teclado de un teléfono móvil saltando entre enlaces. Eso sin contar con los distintos tipos de teclado móviles, por no mencionar

(3)PC (*personal computer*)

que, en ambos casos, desaparece el evento *mouseover* y, por tanto, los *tooltips* y otras pistas para deducir el destino de un enlace que normalmente utilizamos cuando accedemos a Internet desde un ordenador.

Otros conceptos importantes, como la distancia en pantalla, el orden de los elementos, el tamaño de los textos y las imágenes, la forma de escanear visualmente las páginas, etc., cambian de unos dispositivos a otros.

Teniendo en cuenta las grandes diferencias físicas (pantalla, teclados, punteros, etc.), que acarrean formas diferentes de interaccionar, y el número, cada vez mayor, de capacidades de los dispositivos móviles, podemos dudar si será posible, e incluso acertado, alcanzar la famosa web única.

2. Tipos de dispositivos móviles

El término dispositivo móvil cubre un amplio rango de dispositivos electrónicos de consumo. Normalmente, por dispositivo móvil nos referimos a un dispositivo que puede conectarse a Internet. No obstante, algunas veces también se clasifican cámaras digitales y reproductores MP3 o MP4 estándares como dispositivos móviles. La categoría de dispositivos móviles incluye los dispositivos que presentamos en este apartado, así como otros que no trataremos aquí porque no son importantes para los objetivos de esta asignatura. Algunos de estos dispositivos son los siguientes:

- teléfonos móviles
- organizadores y asistentes personales digitales (*personal digital assistant*)
- *web-enabled phones*
- *two-way pagers*
- *smartphones*
- *handheld PC*
- *tablet PC*
- *tablets*
- libros electrónicos (*e-books*)

Mucho ha cambiado desde 1996, cuando se lanzó al mercado el PalmPilot. Incluso en aquel momento, en que ya habían aparecido otros dispositivos que cabían en la palma de la mano (como el Apple Newton), el PalmPilot cambió la manera de entender la movilidad. Gracias a él, los usuarios tenían la opción de usar un dispositivo pequeño, *palm-sized*, para guardar sus planificaciones, calendarios, listas de cosas para hacer, así como para ejecutar otras aplicaciones simples. Esta fue claramente una opción que gustó a los usuarios, como señaló la buena acogida de los dispositivos Palm. En el año 2000, la gran mayoría de los dispositivos *palm-sized* estaban basados en el Palm OS[4].

(4)Palm OS (*palm operating system*).

PalmPilot

Debido al éxito de los dispositivos Palm, muchas otras compañías lanzaron ofertas de dispositivos móviles para intentar llevarse un pedazo del mercado emergente. A medida que entraban compañías nuevas entraron en el mercado, se fueron introduciendo nuevos dispositivos con nuevas características. En este apartado os mostramos las categorías principales de dispositivos móviles.

2.1. *Handheld* PC

El concepto de *handheld* PC es muy antiguo. El diseño puede ser similar al de un portátil, en el que la pantalla se dobla sobre el teclado y crea una carcasa compacta alrededor del dispositivo. Por esta razón, los *handheld* PC fueron comúnmente conocidos como ordenadores *clamshell*[5]. Los dispositivos *clamshell* aparecieron mucho antes de que estuvieran disponibles los primeros PDA.

(5)En inglés, cubierta

A mediados de los años ochenta, Psion presentó un organizador-agenda que ofrecía la capacidad de ejecutar aplicaciones. Permitía a los usuarios ejecutar aplicaciones financieras, científicas y de datos de forma local en el dispositivo. Estas eran aplicaciones añadidas a la función de calculadora, que era la función principal. Incluso teniendo en cuenta que este dispositivo no fue demasiado popular, supuso el origen de la línea que después siguieron los *handheld* PC.

A principio de los años noventa, Psion lanzó un ordenador *clamshell* más funcional, el cual tenía un teclado y una pantalla con una interfaz gráfica de usuario (GUI) para ejecutar aplicaciones más sofisticadas. Otras compañías, como Casio, lanzaron ofertas similares, a menudo demasiado basadas en sistemas operativos propietarios específicos del dispositivo. Estos sistemas operativos no soportaban aplicaciones de terceras partes, lo que era una limitación muy importante. Poco después, Microsoft, que quería entrar en el mercado de los sistemas operativos de PDA, apareció con Windows CE. La mayoría de las compañías dejaron de lado sus sistemas propietarios y adoptaron Windows CE como opción de sistema operativo para sus dispositivos, si bien Psion fue una excepción notable. La implementación específica de Windows CE para dispositivos con estas características es lo que dio lugar a los *handheld* PC.

Ya en tiempos más recientes, la mayoría de los *handheld* PC pasaron a tener una pantalla a color VGA (480 x 320) con teclados completos. Tienen la capacidad de ejecutar una gran variedad de aplicaciones cliente y basadas en web. En general, los *handheld* PC no se usan para sustituir a los portátiles, sino para complementarlos. El uso común de un *handheld* PC no es el de ordenador general, sino el de un dispositivo de recogida de información. La pantalla VGA y el teclado integrado proporcionan una captura rápida de datos, lo que permite a las empresas incrementar la productividad en procesos que antes requerían una captura manual de datos.

Hay muchas ventajas de hacer la recogida de datos de las líneas de negocios con aplicaciones que reemplacen los procesos basados en papel: el proceso es más rápido, la recogida de datos es más precisa, aumenta la productividad de los empleados, se agilizan las transacciones en los procesos del negocio y se reducen los costes operacionales. Incluso teniendo en cuenta que estos beneficios se podrían conseguir con un portátil, un *handheld* PC es ideal para el trabajo por las siguientes razones:

- Los *handheld* PC proporcionan la función de *on* y *off* instantánea, con lo que el acceso a los datos es inmediato.

- Su batería tiene un largo tiempo de vida debido a los chips de bajo consumo usados en su diseño. Una única carga de batería puede durar un día entero de uso.

- Al no tener partes móviles, soportan bien los golpes, por lo que se pueden usar en muchos entornos.

Estas características, además de que su tamaño es más reducido que el de un portátil, hacen que el dispositivo sea ideal para usuarios interesados en el acceso y la captura inmediata de datos. Estos dispositivos pueden albergar una gran variedad de aplicaciones que proporcionan un claro beneficio a usuarios que trabajan fuera del entorno de una oficina. El hecho de que a menudo pesen alrededor de quinientos gramos es una buena característica para gente que pasa mucho tiempo fuera de la oficina.

El mercado de los *handheld* PC, no obstante, ha estado permanentemente bajo presión; por un lado, por las cada vez más potentes PDA; por otro, por los cada vez más pequeños y eficientes portátiles. HP es uno de los fabricantes que más apostó por este mercado. Otros fabricantes se movieron hacia productos más parecidos a un *tablet (*como, por ejemplo, el Samsung NEXiO). El Samsung NEXiO funciona con Windows.NET 4.1 y tiene capacidades 802.11 integradas. Es un buen ejemplo de cómo el mercado de los *handheld* PC ha ido evolucionando.

Handheld PC Samsung NEXiO.

2.2. Teléfono móvil

Los teléfonos móviles simples (al contrario que los *smartphones*) representaron el punto de partida para llegar primero a los *web-enabled phones* y después a los que hoy se conocen como *smartphones*. Como dispositivos, se componen de apenas algunos componentes. Son los siguientes:

- un micrófono
- un altavoz
- una pantalla de cristal líquido o plasma
- un teclado
- una antena
- una batería
- una placa de circuitos

El móvil posee un microprocesador que realiza cálculos a gran velocidad. El microprocesador trata todas las tareas del teclado y de la pantalla, gestiona los comandos y controla las señales de la estación de base, además de coordinar las demás funciones.

Las ventajas que presenta un teléfono móvil como tipo de dispositivo móvil son varias:

- muy extendido
- ligero y transportable.
- económico
- prestaciones de comunicación innatas

Por el contrario, también muestra algunos inconvenientes:

- poca potencia de proceso
- poca memoria
- capacidades de visualización limitada
- interacción avanzada difícil

2.3. *Personal digital assistant*

Un *personal digital assistant* (más conocido como PDA) es, como su propio nombre indica, un organizador digital. Los PDA (a veces llamados ordenadores de bolsillo) combinan elementos de ordenador, teléfono, fax, Internet y *networking* en un único dispositivo. Un PDA puede funcionar como teléfono móvil, fax, navegador web y organizador personal. Como características comunes, los PDA ofrecen básicamente calendarios, blocs de notas y agendas para teléfonos, por lo que han sido los sustitutos tradicionales de las agendas clásicas. También permiten descargar correos electrónicos y otros materiales desde un ordenador, y, además, si ya están equipados con un módem, permiten acceder a Internet. Los PDA tienen capacidad de *on* y *off* instantáneo, lo que significa que el dispositivo no tiene que arrancar cada vez que se quiera usar.

Término *PDA*

Cabe decir que el término PDA se utilizó durante una cierta época, pero actualmente está en desuso.

Ranura de expansión

Los PDA más modernos ya incorporan un módem para la conexión inalámbrica. Los dispositivos que no lo tenían, normalmente tenían una ranura de expansión en la que se podía conectar dicho módem

A diferencia de los ordenadores portátiles, la mayoría de los primeros PDA tenían un lápiz como dispositivo de entrada, en lugar de un teclado. Esto significa que incorporan características de reconocimiento de escritura a mano. Algunos PDA pueden también reaccionar a comandos de voz mediante tecnologías de reconocimiento de voz. Existen PDA disponibles en versiones con *stylus* o teclado (llamado *datapad*).

Normalmente se componen de una pantalla, que suele ser táctil, un procesador, memoria y un sistema operativo. Además, permiten, como ya hemos dicho, conexión con el ordenador de sobremesa, lo que posibilita salvaguardar los datos y exportarlos a bases de datos (o a aplicaciones más elaboradas) o transferir nuevas aplicaciones al asistente.

Ha habido una amplia variedad de asistentes personales. Si nos fijamos en la pantalla, han existido desde los que son monocromos o, como mucho, presentan una escala de colores, hasta los que poseen más de sesenta y cinco mil. El tamaño también cambia de un modelo a otro y según el tipo de tecnología utilizada: los basados en matrices activas presentan una mejor calidad que los basados en matrices pasivas, los cuales consumen menos energía. Por esta razón, las baterías suelen ser recargables y extraíbles. La memoria oscilaba entre los dos y los sesenta y cuatro megabytes en los primeros modelos. La primera cantidad es suficiente para aplicaciones básicas como el bloc de notas, el calendario, la agenda y algunas utilidades más. Si lo que se desea es almacenar ficheros grandes (como fotos, bases de datos o programas de gran tamaño) es imprescindible una memoria de mayor capacidad.

PDA y multimedia

Los PDA de última generación son excepcionales para jugar y entretenerse, leer libros, ver fotos, escuchar música e incluso reproducir películas. La memoria se puede ampliar con tarjetas.

Inicialmente, la conexión al PC se realizaba mediante un cable, pero posteriormente esta se pudo realizar sin necesidad de cable, de manera inalámbrica. La sincronización se lleva a cabo mediante infrarrojos o radio (como es el caso de Bluetooth). De esta manera, los usuarios pueden intercambiar información (entradas de una agenda, correos electrónicos, etc.) simplemente situándolo cerca de un ordenador de sobremesa. Sin embargo, la conexión inalámbrica va todavía más allá, pues los PDA pueden tener conectividad a una red WiFi de área local o usar un módem CDPD[6] para acceder a Internet, lo que aumenta sus posibilidades, como son las de navegación por Internet o el envío y recepción de correo electrónico, entre otras.

(6)CDPD (*cellular digital packet data)*

Los PDA, con respecto a los móviles tradicionales, presentan algunas ventajas:

- Las pantallas son más grandes y la visualización se mejora.
- La interacción con el usuario es más fácil (fundamentalmente porque la pantalla es táctil).
- Son más potentes (desde un punto de vista computacional).

Sin embargo, también presentan algunos inconvenientes:

- Necesitan accesorios para comunicarse.
- El precio es mayor que el de los teléfono móvil tradicionales.

Los dispositivos más comunes en este mercado han sido el Palm y el Pocket PC. Los dispositivos Palm fueron inicialmente los líderes del mercado, aunque esto dejó de ser así a medida que los dispositivos Pocket PC mejoraban sus capacidades y se adaptaban mejor a las necesidades de los usuarios empresariales. Como hemos comentado de manera general, tanto unos como otros tienen pantallas táctiles con soporte para reconocimiento de caracteres. De los dispositivos Palm aparecieron versiones con pantallas en monocromo y en color, mientras que los Pocket PC ofrecían normalmente pantallas en color.

Los dispositivos Palm son algo más pequeños que los Pocket PC y, por lo tanto, muy fáciles de transportar (suelen pesar alrededor de doscientos veinticinco gramos).

Aunque el concepto inicial de los dispositivos Palm parecía atractivo, el rendimiento y las capacidades de almacenamiento estaban por debajo de lo que muchas veces era deseable. Muchos dispositivos basados en Palm OS tenían entre ocho y treinta y dos megabytes (MB) de memoria y procesadores que funcionaban a 33 megahercios (MHz) o menos. Tradicionalmente, eso era aceptable para las aplicaciones que se ejecutaban en estos dispositivos, pero a medida que las necesidades para ejecutar aplicaciones aumentaron, estas limitaciones empezaron a suponer un problema. A medida que estas aplicaciones evolucionaban, requerían más capacidad de almacenamiento de datos y de procesamiento. Además, debido a que en estos dispositivos se ejecutaban aplicaciones Java, requerían mayor capacidad de proceso para trabajar con los requisitos de las JVM (*Java virtual machine*) y las aplicaciones relacionadas. Una nueva hornada de dispositivos Palm que ejecutaban el Palm OS 5.0 superó este problema mediante la utilización de procesadores basados en ARM, que funcionan a 206 MHz. Esto colocó a estos dispositivos en una mejor posición en el mercado con respecto a los dispositivos Pocket PC.

La mayoría de los dispositivos basados en Pocket PC tenían características ligeramente mejores en cuanto a rendimiento. Estos dispositivos tenían entre treinta y dos y sesenta y cuatro megabytes de memoria y procesadores X-Scale que funcionaban a 400 MHz. Esta capacidad de cálculo adicional era necesaria para ejecutar el sistema operativo Windows CE, que es más "pesado" que su contrapartida, el Palm OS. Un efecto negativo de este incremento en la capacidad de proceso es su efecto en el consumo de batería. Los dispositivos Palm pueden durar a menudo varios días con una sola carga de batería, mientras que los dispositivos Pocket PC tenían que ser recargados a diario, por lo general.

El mercado original de muchos de estos dispositivos era el de asistente digital personal, de ahí el nombre PDA. Palm consiguió un gran sector del mercado de PDA ofreciendo una interfaz de usuario intuitiva para acceder a aplicaciones comunes, tales como calendarios, listas de contactos y listas de tareas pendientes. Además, ofreció un programa de desarrollo bien recompensado para promocionar la proliferación de aplicaciones basadas en Palm OS. Este rango de aplicaciones, junto con el relativo bajo coste de los dispositivos Palm, hizo de los dispositivos Palm OS una opción atractiva para muchos usuarios de PDA.

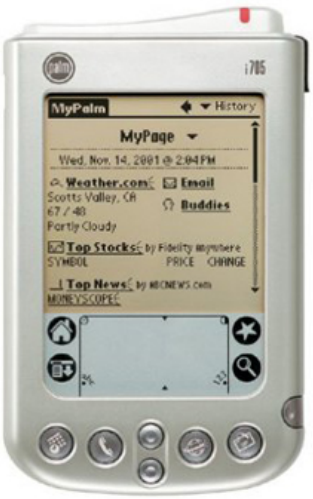
Los dispositivos Pocket PC aparecieron en el mercado unos cuantos años después de los dispositivos Palm con un público objetivo diferente en mente: usuarios de gama alta que buscaban una interfaz de usuario rica con capacidades multimedia. Estos dispositivos también tenían como público objetivo los usuarios que querían usar sus productos Microsoft Office, como Microsoft Word y Excel, en sus PDA. Para conseguir estos objetivos, los Pocket PC necesitaban más capacidades *hardware*, lo que también afecta al precio. En general,

la mayoría de los dispositivos Pocket PC eran entre un 30% y un 50% más caros que los dispositivos Palm OS. Sin embargo, esto cambió con la entrada de Dell en el mercado de los Pocket PC con dispositivos más baratos.

Para distinguir mejor las ofertas para los usuarios domésticos y los profesionales, Palm Inc. creó dos familias de dispositivos: Tungsten y Zire. La familia de productos Palm Tungsten está destinada a profesionales de empresas para quienes la movilidad es imprescindible. Estas soluciones combinan un *hardware* potente y soluciones *software* para solventar los problemas a los que estos profesionales se tienen que enfrentar. La familia de productos Palm Zire está orientada al mercado de usuarios domésticos y ofrece dispositivos baratos con diseños fáciles de usar. Palm introdujo estas familias de productos a finales del 2002 con el objetivo de mantener su cuota de mercado líder en el sector de los dispositivos de mano (*handheld*).

Cuando hablamos de conectividad inalámbrica, la mayoría de los dispositivos PDA (sobre todo los primeros) requerían un módem como componente añadido. Esto se consigue normalmente mediante una ranura de expansión. En cualquier caso, el módem añade un coste y una complejidad extra a la solución inalámbrica PDA. En la mayoría de los casos, es necesario que los usuarios establezcan una conexión a la red inalámbrica cuando haya que transferir datos y que se desconecten cuando finalicen. Esto significa que no están siempre conectados a la red, por lo que enviar datos al dispositivo es más complejo.

La conexión manual a redes inalámbricas cambió con la introducción de dispositivos inalámbricos integrados como el Palm i705. Estos dispositivos venían con módems integrados para un rápido y fácil acceso a los datos. Estos dispositivos integrados crecieron en popularidad, tanto entre usuarios domésticos como entre profesionales. La facilidad de uso y las conexiones de red siempre activas los convirtieron en opciones muy atractivas para una gran variedad de aplicaciones móviles.



Palm i705 con conexión inalámbrica integrada

Los siguientes años fueron muy interesantes debido a que el mercado de dispositivos PDA continuó evolucionando. Los fabricantes diseñaron dispositivos enfocados a los profesionales que necesitaban dispositivos de altas prestaciones para ejecutar aplicaciones más avanzadas. Con la introducción de Windows CE .NET y Palm OD 5.0, parecía muy claro que este mercado iba a continuar siendo competitivo durante algún tiempo. Esto benefició al consumidor debido a las nuevas prestaciones que iban apareciendo para atraer su atención. Un ejemplo de esta nueva generación de dispositivos es el HP iPAQ.

HP iPAQ

Desde prácticamente el 2001 ya se observaba una convergencia entre dispositivos PDA y dispositivos de voz. A finales del 2001 y durante el 2002, muchos vendedores lanzaron dispositivos combinados de voz y PDA basados en el Palm OS. Además, Pocket PC Phone Edition y Microsoft Smartphone 2002 usaban Windows CE como su sistema operativo y proporcionaban muchas funcionalidades presentes en PDA Pocket PC, con funcionalidades adicionales de voz.

2.4. *Web-enabled phone*

Los teléfonos móviles son, con diferencia, los dispositivos *wireless* más usados en el mercado. Por lo general, el uso principal era el de las llamadas de voz, pero con los mensajes de texto primero y otras tecnologías inalámbricas para acceder a Internet después, las aplicaciones de datos se han generalizado. Gracias a su gran popularidad, los *web-enabled phones* se han convertido en un objetivo inmediato para las aplicaciones con acceso inalámbrico a Internet.

La figura muestra un *web-enabled phone* típico. Como podéis ver, dispone de una pantalla muy limitada (normalmente de entre cuatro y doce líneas de texto), con el teclado típico *keypad* de doce botones para la entrada de datos. Estas limitaciones hacían de los teléfonos móviles una opción muy pobre para navegar por Internet debido a que la cantidad de datos que puede visualizarse o introducirse es muy limitada.

Nokia 8390 *Web-enabled phone*

No obstante, el punto fuerte de los teléfonos móviles era su uso extendido, por lo que resultaban muy interesantes para el mercado de aplicaciones de usuario. Algunos ejemplos incluyen niveles de *stock*, información de tráfico, información sobre vuelos, compra de entradas y titulares de noticias. En todas estas aplicaciones se requiere una cantidad limitada de datos para que funcionen correctamente. La cantidad de datos recibida no es excesiva, por lo que el dispositivo la puede mostrar de una forma fácil de entender. Si, por el contrario, pretendemos implementar aplicaciones más ambiciosas en cuanto a volumen de datos o en cuanto a la forma de mostrarlos, necesitaremos dispositivos más capacitados.

Como es de esperar, los teléfonos móviles permiten a los usuarios conectarse a redes inalámbricas. Una vez conectados, los usuarios pueden usar el móvil tanto para llamadas de voz como para aplicaciones de datos. Además, gracias a que pueden estar siempre encendidos, los teléfonos móviles son ideales para aplicaciones de mensajes de texto. Ya que, por lo general, estos mensajes están limitados a una longitud de ciento sesenta caracteres, la capacidad de entrada de datos de estos dispositivos es adecuada. Otra ventaja de estos dispositivos es la larga vida de la batería. Con su capacidad de procesamiento limitada, los teléfonos celulares pueden conservar energía y, de esta manera, durar más tiempo encendidos que dispositivos más sofisticados como *smartphones* o PDA. Hay que tener en cuenta, no obstante, que mientras más se use el dispositivo para comunicación *wireless*, más rápido se consumirá la batería.

El uso principal de estos dispositivos era para voz, por lo que la calidad de la comunicación por voz, la cobertura de red y los paquetes de llamadas eran, normalmente, más prioritarios que los servicios de datos a la hora de escoger uno de estos dispositivos.

2.5. *Smartphone*

Los *smartphones* combinan los conceptos de teléfono móvil y ordenadores *handheld* en un único dispositivo. Los *smartphones* permiten guardar información (por ejemplo, correos electrónicos) e instalar programas, además de usar un teléfono móvil en un único dispositivo. Por ejemplo, un *smartphone* podría considerarse como un teléfono móvil con funciones de PDA integradas en el dispositivo o viceversa.

Funciones de los *smartphones*

Los *smartphones* o teléfonos inteligentes son teléfonos que soportan más funciones que un teléfono común. Entre estas funciones suelen encontrarse la de gestor de correo electrónico, la funcionalidad completa de organizador personal, y suelen estar pensados para acceder de manera continua a Internet. Actualmente se les añade como función común la posibilidad de instalar programas adicionales.

Uno de los mayores atractivos de los *smartphones* es su simplicidad. El usuario medio puede tener su dispositivo funcionando en cuestión de minutos sin tener que preocuparse de una configuración complicada.

Otro atractivode estos dispositivos es que los usuarios pueden ampliar las características del dispositivo descargando nuevas aplicaciones mediante la conexión inalámbrica. Esto se llama aprovisionamiento *over the air* (OTA). Tanto los fabricantes de dispositivos móviles como los operadores de telefonía han creado y están creando mercados o escaparates donde los desarrolladores pueden subir sus aplicaciones para posteriores descargas. Los usuarios pueden bajarse las aplicaciones que les interesen por un precio reducido (usualmente de entre uno y cinco euros). El beneficio se divide entre el desarrollador y el proveedor del mercado. Los juegos han sido tradicionalmente las aplicaciones más descargadas.

2.5.1. *Smartphone* de gama baja

Los *smartphones* se llaman así por su capacidad de ejecutar aplicaciones locales y realizar llamadas de voz. Las primeras versiones de estos dispositivos fueron los dispositivos denominados *smartphones* de gama baja. Estos dispositivos, al igual que los *web-enabled phones*, son principalmente dispositivos de voz. El soporte para aplicaciones es algo limitado debido a las limitaciones de almacenamiento, a la capacidad de procesamiento y a los tamaños de las pantallas. Son dispositivos similares a los *web-enabled* en cuanto a la forma y al tamaño de pantalla, pero proporcionan funcionalidades un poco más avanzadas.

Además, como estos dispositivos aún tienen pequeños procesadores y poca memoria, son capaces de aguantar días funcionando con una sola carga de batería. Por supuesto, el hecho de que estos precursores de los *smartphones* ofrezcan capacidades de teléfono celular, además del soporte a aplicaciones, los hace ideales para aquellos que quieren limitar el número de dispositivos con los que tienen que cargar.

Gran parte del interés que despertó Java 2 Micro Edition (J2ME) se debió a su utilidad para desarrollar aplicaciones para estos dispositivos.

Nokia vendió millones de *smartphones* basados en J2ME en el 2003. Este número, que no es para nada despreciable, representó una gran oportunidad para el desarrollo de aplicaciones para los programadores de J2ME. También indicaba que habría un gran número de estos dispositivos en el mercado.

J2ME

No solo se vendieron dispositivos Nokia basados en J2ME. Motorola y otros fabricantes también lanzaron dispositivos J2ME al mercado.

2.5.2. *Smartphone* de gama alta

A medida que el mercado de las aplicaciones inalámbricas fue madurando, se produjo un movimiento hacia dispositivos mucho más potentes llamados *smartphones* de gama alta. Los fabricantes líderes de teléfonos celulares (Nokia, Sony Ericsson y Motorola, entre otros) empezaron a producir estos dispositivos, que en principio estaban pensados para el mercado profesional. Estos dispositivos proporcionan funcionalidad para la comunicación por voz, además de aplicaciones cliente ligeras, pero útiles. Esto los convirtió en una buena op-

ción para aquellas personas que no querían cargar con diversos dispositivos, pero que querían poder seguir disfrutando de una gran variedad de aplicaciones. Un ejemplo es el *smartphone* Sony Ericsson P800.

Smartphone Sony Ericsson P800

La definición de un *smartphone* se encuentra entre la de un teléfono celular y un PDA. Los primeros (sobre todo antes de la aparición masiva de dispositivos con pantalla táctil) tenían normalmente un mecanismo deslizante para mostrar la pantalla completa y el teclado. Cuando están cerrados, parecen teléfonos móviles normales, con el típico *keypad* de doce teclas y con una pequeña parte de la pantalla al descubierto. Cuando están abiertos, tienen tamaños de pantalla que van de 640 x 200 a 320 x 240. A veces disponen también de un teclado para la entrada de datos. Los procesadores de estos dispositivos ya eran lo suficientemente potentes como para ejecutar tanto aplicaciones locales como aplicaciones con acceso inalámbrico a Internet sofisticadas. Los navegadores de estos dispositivos tienen, por lo general, un soporte para gráficos en color y multimedia, y usan tanto WML[7] como HTML[8].

(7)WML (*wireless markup language*)

(8)HTML (*hypertext markup language*)

Al igual que los teléfonos móviles, los *smartphones* se pueden usar durante días con una sola carga de batería. Hasta hace poco, los servicios que proporcionaban no eran demasiado adaptables o personalizables. El usuario no tenía la posibilidad de añadir aplicaciones o de cambiar el contenido disponible en el dispositivo. Esto cambió cuando apareciendo *smartphones* más modernos, que ofrecían sistemas operativos completos y una gran cantidad de memoria disponible para aplicaciones de terceros. Los primeros sistemas operativos para *smartphones* más comunes fueron Symbian OS, Palm OS, Pocket PC Phone Edition y Microsoft Smartphone 2002. El soporte a J2ME es común también.

Lanzamiento del iPhone

La situación ha cambiado mucho en pocos años, como se puede ver, por ejemplo, en el caso del iPhone, que se lanzó primero en EE. UU.

En Norteamérica, los primeros *smartphones* no tuvieron la misma aceptación que en Europa y Asia debido, entre otras razones, al soporte de red inalámbrica, al sistema operativo utilizado y a los planes de mercado y de distribución de los fabricantes. Por lo general, los fabricantes de *smartphones* lanzaban sus nuevos dispositivos en Europa antes de venderlos en Norteamérica. Un buen ejemplo de esto fue el Nokia Communicator. El Nokia 9210 se lanzó en Europa a finales del 2001, mientras que su contrapartida, el Nokia 9290, no se lanzó en Norteamérica hasta mediados del 2002.

Otra categoría de dispositivo que se incluye en la categoría de *smartphone* es el PDA con soporte a voz integrado. Tanto Kyocera como Handspring lanzaron dispositivos Palm OS con voz integrada. RIM lanzó la BlackBerry 6710/6750 con soporte integrado para comunicación por voz, y aparecieron una gran variedad de dispositivos Pocket PC Phone Edition de vendedores como Audio-Vox y Samsung. Todos estos productos híbridos hicieron que se fuera difuminando la línea entre las dos clases de dispositivos: PDA inalámbricos y *smartphones* de gama alta.

Como exponentes de la generación moderna de *smartphones*, se pueden citar el iPhone 4, el iPhone 5 y el Samsung Galaxy II.

El iPhone incorpora prestaciones como videollamadas, pantalla retina, multitarea, grabación y edición en alta definición y una cámara de 5 megapíxeles con flash LED, entre otras.

iPhone

El Samsung Galaxy II fue el siguiente teléfono en incorporar un procesador de doble núcleo después del LG Optimus 2X y, si cabe, el rendimiento del Samsung Galaxy II es todavía mejor. Es cierto que LG tomó la delantera con el lanzamiento de su terminal de doble núcleo, pero el Samsung Galaxy II, además de contar con esta potencia, se beneficia del *software* con el que funciona y de las actualizaciones de este: Android. El Galaxy II funciona realmente rápido y con fluidez. Lo bueno de este teléfono es que los fabricantes han tomado lo que ya tenían y lo han mejorado sin cambiar lo que ya era bueno; simplemente han cogido el Samsung Galaxy S y le han añadido potencia, velocidad, una cámara de más calidad, RAM y la última versión de Android. El frontal se parece al del iPhone. Es mucho más parecido en las fotografías que en la realidad, ya que la parte trasera es diferente y es muy delgado. No obstante, podemos decir que no destaca por su diseño innovador.

Samsung Galaxy II

2.6. *Tablet* PC

Un *tablet* PC es un tipo de ordenador que tiene una pantalla con la que se puede interaccionar directamente (por lo general, con un *stylus*). La escritura a mano se digitaliza y se puede convertir a texto estándar mediante herramientas de reconocimiento de escritura, o se puede guardar como texto escrito a mano. Normalmente, también puede desplegar un teclado táctil en la pantalla que se puede usar con un *stylus* o con los dedos. En este teclado, las teclas pueden estar dispuestas como en un teclado QWERTY estándar o de forma diferente. De manera opcional, los *tablet* PC pueden tener accesorios como, por ejemplo, un teclado externo para facilitar el trabajo de sobremesa.

El mercado de los *tablet* PC es muy interesante. Durante muchos años, los dispositivos de esta categoría no han sido especialmente exitosos. Estos dispositivos estaban pensados para trabajos de campo, como una alternativa competitiva a los portátiles. En cuanto al tamaño, son ligeramente más pequeños que un portátil y algunos de ellos tienen la capacidad de cambiar su apariencia (portátil o *tablet)*. En la posición de portátil tienen un teclado para la entrada de datos, y cuando están en el formato *tablet*, tienen una pantalla táctil con entrada de datos basada en un lápiz. Un ejemplo de *tablet* PC es el Acer TravelMate 100.

Acer TravelMate100 *tablet* PC, cuando cambia de modo portátil a modo *tablet*.

Los *tablet* PC han ejecutado tradicionalmente Windows XP como su sistema operativo y tienen el conjunto de accesorios periféricos habituales de un portátil. También tienen capacidades de procesamiento y almacenamiento simi-

lares. Por lo tanto, podemos considerarlos como una evolución de los portátiles, con todas sus características, con características de *tablet* añadidas y con una mayor duración de la batería.

Aplicaciones Microsoft

Microsoft desarrolló bastantes nuevas aplicaciones (como el Tablet PC Journal) para sacarle partido a las características de las pantallas táctiles. El *software journal* permite a los usuarios introducir notas y diagramas con su propia escritura a mano, que puede entonces convertirse en texto estándar al tocar un botón. Es, por tanto, algo así como un papel electrónico.

La presión de los portátiles por un lado, y, sobre todo, la tremenda evolución que han experimentado los *tablets* puros, hacen que este tipo de dispositivos no haya tenido mucho éxito.

2.7. Tableta

Después del repaso histórico a los diferentes tipos de dispositivos móviles, llegamos a las tabletas, que han dejado de ser un producto del futuro para instalarse como una realidad presente en el mercado tecnológico. Un mercado dominado incontestablemente por el iPad de Apple.

iPad de Apple (EFE)

Los fabricantes se apresuran a sacar provecho de la atención que el iPad de Apple ha generado, lo que causa que aparezca una nueva tableta cada poco tiempo. No se puede hacer un análisis de cada tableta, pero a continuación se recopila una visión general del mundo de la tableta. En el 2011 el mercado contaba con más de cien modelos entre los que poder escoger. A continuación se enumeran los modelos más significativos, además del propio iPad.

El iPad es probablemente la tableta que menos explicación necesita. Con un millón de iPads vendidos en el primer mes de su introducción, Apple ha tomado rápidamente la posición de liderazgo en la categoría de tableta.

Como puntos a favor del iPad se puede citar su elegante hardware, el App Store, que es ideal para la reproducción de multimedia; una amplia selección de juegos; un procesador rápido, pantalla multitoque, y la duración de la batería. Se vende desde 499 dólares.

Como puntos negativos se puede mencionar que los usuarios deben comprar software exclusivamente de Apple, que carece de compatibilidad con Adobe Flash, y que el soporte de hardware es limitado.

El iPad ya cuenta con su segunda versión en el mercado mientras muchas otras tabletas no han hecho más que nacer. Sin embargo, estas tabletas llegadas más tarde cuentan, en muchas ocasiones, con utilidades que no tiene la última joya de Apple.

Aspecto del más delgado iPad 2 (Monica M. Davey / EFE)

RIM lanzó su Blackberry Playbook, que, con unas dimensiones de 19,3 x 13 centímetros y 10 milímetros de grosor, resulta similar a un smartphone gracias a su pantalla de 7 pulgadas (el resto se encuentra entre las 9,7 y las 10,1 pulgadas). Cuenta con una resolución de 1.024 x 600 píxeles, un procesador Cortex-A9, una memoria de un gigabyte de RAM y una capacidad de almacenamiento de entre 16 y 64 gigabytes, según modelos. Dispone de un sistema operativo propio, QNK, pero podrá ejecutar aplicaciones desarrolladas para el sistema operativo Android en su versión 2.3, adquiridas en Blackberry App World, donde también se podrá disponer de una aplicación para ejecutar programas basados en Java. En cuanto a conectividad a las redes, cuenta con Wi-Fi y 3G (4G en EE.UU.). Su precio oscila entre los 500 y los 699 dólares, según el modelo seleccionado. En España está disponible desde el 19 de abril del 2011.

La Samsung Galaxy Tab 10.1 (a la que Apple acusó de estar copiando el iPad), con unas dimensiones de 24,6 x 17 centímetros y 10,9 milímetros de grosor y una pantalla de 10,1 pulgadas, tiene una resolución de 1.200 x 800 píxeles. Entre sus prestaciones están un procesador Nvidia Tegra 2 Dual Core, a un gigahercio de frecuencia de reloj y capacidad de almacenamiento de 16 y 32 gigabytes. El iPad ha hecho que Samsung renueve su producto con una nueva versión, además de preparar el lanzamiento de la Galaxy Tab 8.9 (con pantalla de 8,9 pulgadas). Los nuevos modelos contarán con cambios, como la reducción de grosor. En el mercado norteamericano, el precio se sitúa entre los 499 y los 599 dólares para el modelo de 10,1 pulgadas, y entre los 469 y los 569 euros para el de 8,9 pulgadas, en sus versiones con conectividad Wi-Fi. Además, lleva por defecto conectividad 3G y 4G en los países en los cuales esté desarrollado este protocolo.

Samsung Galaxy Tab

Motorola, por su parte, cuenta con Motorola Xoom. Con unas dimensiones de 24,3 x 16,17 centímetros y 12,7 milímetros de grosor, su pantalla es de 10,1 pulgadas y una resolución de 1.280 x 800 píxeles. El procesador es un Nvidia Tegra 2 Dual Core a un gigahercio y dispone de un gigabyte de memoria. Su capacidad de almacenamiento es de 32 gigabytes. Cuenta con el sistema operativo Android. En España se pueden encontrar dos versiones: solo con conectividad Wi-Fi o un modelo con Wi-Fi y 3G. El precio de la tableta liberada es de unos 579 euros.

HP Touchpad es la tableta de HP. Muchos expertos coinciden en que puede ser el más firme competidor del iPad, ya que nace de la adquisición de la empresa Palm (autora de algunas de las mejores agendas electrónicas de la primera década del siglo y desarrolladora del sistema operativo móvil WebOS, que ahora HP implanta en su tableta) por parte de HP. Sus dimensiones son 24,2 x 19 centímetros y tiene un grosor de 13,7 milímetros, con una pantalla de 9,7 pulgadas y una resolución de 1.024 x 768 píxeles. Utiliza un procesador Qualcomm Snapdragon dual core con 1,2 gigahercios de frecuencia de reloj y un gigabyte de memoria RAM. Dispone de versiones con almacenamiento de 16 y 32 gigabytes. Se lanzó al mercado en el 2011.

	SNAPKEYS THE IMAGINARY INTERFACE	2i	QWERTY (en pantalla)
Número de teclas	Ninguna		101
Pantalla disponible	100%		10%-50%
Escribir sin ver	Instantanea		Casi nunca
Aprender a escribir	Fácilmente		Díficilmente
Movimiento dedos	Mínimo		Demasiado
Velocidad entrada datos	Muy rápida		Lenta
Interfaz	En su mente		Compleja
Escribir textos largos	Es un placer		Es cansado
Escribir en movimiento	Sí		Difícil
Divertido	Como videojuego		Jamás
Rediseño	No se requiere		Necesario

3. Características específicas o componentes de los dispositivos móviles

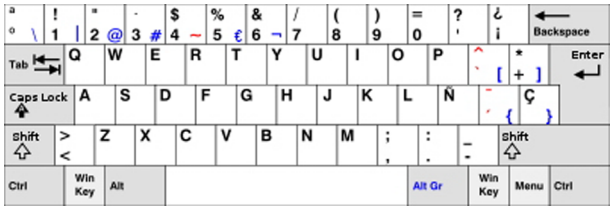
En este apartado os explicaremos los diferentes componentes que puede tener un dispositivo móvil, como el teclado, la pantalla, los sensores, los conectores o las baterías, y os explicaremos los diferentes tipos que puede haber de cada componente, así como sus principales características.

3.1. Teclado de un dispositivo móvil

En este subapartado os mostraremos los diferentes tipos de teclado que puede tener un dispositivo móvil y las características que tiene cada uno de ellos. Para ello, utilizaremos ejemplos de dispositivos con diferentes teclados. Como veréis a continuación, existen dos grandes familias de teclados: los físicos y los virtuales. Estos últimos se caracterizan por el hecho de que el teclado aparece dibujado en la pantalla táctil del dispositivo y, por lo tanto, no representa un componente físico del mismo.

3.1.1. Teclado QWERTY

Cuando hablamos de un teclado QWERTY, en realidad estamos hablando de una distribución de teclas como la que se muestra en la siguiente figura. El nombre *QWERTY* viene de la misma disposición del teclado; si observáis la figura, veréis que las seis primeras letras de izquierda a derecha de la fila superior forman esa palabra.



Teclado QWERTY

Durante muchos años, enviar correos electrónicos desde un móvil fue algo reservado a unos pocos usuarios. Además, estos correos raramente consistían en más que unas cuantas palabras, debido a la dificultad de componer el mensaje. Con la aparición de la generación del texto y de las redes sociales, una gran cantidad de usuarios pueden enviar infinidad de mensajes, algunos de gran longitud. Los principales fabricantes vieron en esto una oportunidad, y ahora podemos encontrar cada vez más dispositivos con teclados QWERTY completos. Con estos teclados es mucho más fácil escribir texto que con técnicas

(9)T9: *Text on 9 keys*

de texto predictivo o con los teclados T9[9], que explicaremos más adelante. A continuación, enumeramos algunos teléfonos móviles con teclados QWERTY completos.

Uno de los pioneros en el uso de teclados QWERTY en dispositivos móviles, RIM[10], ha disfrutado de una gran aceptación entre los usuarios del mundo de los negocios gracias a sus dispositivos BlackBerry. El ejemplo de la foto es un dispositivo de 13,5 mm de grosor que, a pesar de ello, mantiene un teclado QWERTY completo. Las teclas son pequeñas, pero no tanto como para resultar incómodas, y con cada pulsación se produce un clic como respuesta. Con este teclado se pueden escribir correos electrónicos, textos, documentos de Word, etc., a una gran velocidad.

(10)RIM (*research in motion*)

Teclado QWERTY completo de un dispositivo BlackBerry

Otro ejemplo de dispositivo con teclado QWERTY es el HTC Touch Pro. Este dispositivo integra un teclado QWERTY y una pantalla táctil. En este caso, el teclado está escondido detrás de la pantalla, y es necesario deslizar la pantalla para poder verlo. Aunque no hay espacio físico entre las teclas, estas están diferenciadas de forma individualizada y presentan una buena respuesta. Este teléfono se puede considerar grande en cuanto a tamaño y el teclado QWERTY que presenta es el más parecido al teclado de un PC que se puede encontrar, con teclas para números, flechas y una selección de signos de puntuación.

HTC Touch Pro

Siguiendo con los ejemplos de teclados QWERTY, la familia de modelos LG KS presenta un teclado QWERTY deslizante igual que el del HTC Touch Pro, que se muestra en la siguiente figura. En este caso, las teclas están lo suficientemente separadas como para que se pueda escribir rápidamente sin errores. Sin embargo, en este teclado no hay teclas para números o flechas y, por lo tanto, es necesario pulsar la tecla *shift* y las letras correspondientes para escribir dígitos o usar las teclas de cursor.

Móvil LG con teclado QWERTY

Otro dispositivo con características similares en cuanto al teclado es el Xperia X1. La diferencia es que en este dispositivo, mientras deslizamos el teclado hacia fuera, la pantalla táctil se inclina ligeramente en forma de arco y da la sensación de que estamos usando un portátil. Debido a la gran longitud del dispositivo, el teclado es particularmente amplio, con lo que escribir texto resulta cómodo. A pesar del tamaño del dispositivo, el teclado no presenta teclas numéricas individuales.

Xperia X1

La familia de dispositivos LG Prada presenta otro ejemplo de teclado completo QWERTY deslizante. Como se puede ver en la siguiente figura, las teclas están ligeramente levantadas con un buen grado de separación. Sin embargo, aunque hay teclas para las flechas, las teclas numéricas están, una vez más, integradas en una selección de teclas alfabéticas.

LG Prada

Podéis observar, por lo tanto, que los teclados QWERTY no están reñidos con las pantallas táctiles. Tanto si queremos interaccionar con la pantalla, como si queremos teclear, existen dispositivos que ofrecen las dos opciones. Por ejemplo, se puede preferir la libertad de una pantalla táctil para navegar por Internet y la comodidad de un teclado QWERTY para escribir correos electrónicos más largos.

Además de los ya vistos, otros dispositivos con ambas características son el HTC Touch Pro 2, el Nokia N900 y el T-Mobile G1, que se pueden ver en las siguientes imágenes.

El teclado QWERTY del HTC Touch Pro 2 tiene cinco líneas, de las que una de ellas está dedicada a los números. Las teclas son espaciosas y hay un botón de mensajes que nos lleva directamente a los textos y a los correos electrónicos.

HTC Touch Pro 2

En cuanto al Nokia N900, su teclado QWERTY es liso y táctil, las teclas son de goma y presenta una distribución inteligente de las teclas de símbolos y puntuación.

Nokia N900

Para finalizar con los ejemplos de teclados QWERTY, el T-Mobile G1 presenta un teclado deslizante (oculto debajo de la pantalla táctil) compacto y fácil de usar.

T-Mobile G1

3.1.2. Teclados Fitaly

Los teclados Fitaly son teclados con una disposición de las teclas diferente a los teclados QWERTY. Estos teclados se basan en las probabilidades de teclear una letra. En ellos, las teclas más usadas se encuentran en el centro.

Este tipo de teclado existe básicamente en su versión *software* y está pensado para la entrada de datos con un único dedo. Era un teclado utilizado en dispositivos Palm.

Teclado Fitaly

3.1.3. Teclado T9

En este subapartado os presentamos el teclado T9, desarrollado originalmente por Tegic Communications. El teclado T9 surgió como una alternativa a los teclados QWERTY. La idea del T9 no es nueva, puesto que ya ha sido usada en teléfonos móviles, paginadores, mandos de TV y PDA.

El teclado T9 usa nueve teclas de tamaño relativamente grande, distribuidas como las del teclado de marcado de un teléfono. En lugar de tener que pulsar pequeñas teclas (en comparación con un QWERTY), se pulsan estos botones de tamaño más grande. Cada tecla tiene tres letras. La ventaja de este teclado

es que no es necesario pulsar sobre la letra exacta que se desea. Simplemente hay que pulsar la tecla que tiene la letra deseada. La base de datos del T9 intenta entonces adivinar qué palabra estamos intentando escribir. Esto puede parecer inverosímil, pero la realidad es que funciona bastante bien. Si queremos introducir una palabra que no se encuentra en la base de datos, podemos teclearla fácilmente letra por letra. La lista de palabras del usuario puede ser relativamente amplia (por ejemplo, en sistemas Palm Os este número es de unas dos mil palabras). Si sobrepasamos dicho número de palabras, automáticamente se borran las menos usadas para dejar espacio a las nuevas.

Es bastante sencillo usar un T9. Al contrario de lo que ocurre con los teclados Fitaly, no es necesario aprender nada nuevo. El manejo de un T9 se aprende rápidamente. Simplemente se trata de pulsar y pulsar y raras veces aparecen errores o palabras desconocidas. Por supuesto, en muchas ocasiones las letras que se introducen al principio de una palabra pueden dar lugar a diferentes palabras. En estos casos, es necesario mirar la lista de palabras y seleccionar la palabra deseada.

Ejemplo

Si en inglés se quiere escribir la palabra *cool*, las teclas que forman esta palabra podrían también formar *book*, *cook* y otras. Como todas son palabras reales, el sistema T9 no sabe cuál es la que se quiere hasta que se selecciona de la lista que propone. Si no se selecciona ninguna, el sistema simplemente seleccionará la palabra que se use con más frecuencia. Si en la lista hay más palabras de las que se pueden mostrar en la pantalla, aparecerá un menú desplegable para seleccionar la palabra.

Además del teclado alfabético normal, hay teclados para números, puntuación y símbolos. Un teclado T9 es, en general, rápido y sencillo de usar.

En la siguiente tabla podéis ver los resultados de un pequeño experimento que realizó un usuario. El experimento consistía en escribir tan rápido como fuera posible la siguiente frase: "Meeting in conference room 215 at 2:30 pm".

Tiempo empleado en escribir una frase con diferentes tipos de teclado

Método	Segundos
QWERTY	38
Graffiti	40
Fitaly	43
T9	51

En realidad, estos resultados no aportan demasiada información. Al final, el hecho de usar un tipo de teclado u otro depende de las preferencias del usuario; cada usuario utilizará el método con el que se sienta más cómodo.

Uno de los problemas de los teclados T9 es que el programa de predicción gastará recursos de nuestro dispositivo (memoria y procesador, básicamente), además del espacio que ocupe la lista de palabras del usuario.

3.1.4. Teclados *software*

Cuando hablamos de teclados *software*, nos referimos a los teclados que los dispositivos con pantalla táctil pueden presentarnos por pantalla para la introducción de texto, ya sea mediante la utilización de un lápiz o con nuestros propios dedos. Por tanto, esta categoría no excluye a las ya vistas.

Nota

Podemos tener un teclado QWERTY táctil en pantalla o un teclado Fitaly, como ya hemos visto.

Los teclados *software* han cobrado mucha importancia gracias a los dispositivos con pantalla táctil como, por ejemplo, los *tablets*. La llegada de los *tablets* obligó a replantear el uso del teclado.

En este subapartado vamos a hacer hincapié en el *software* que hay detrás del teclado táctil. Para ello, veremos algunos ejemplos.

Swype

Para empezar, encontramos el caso de Swype. Swype proporciona una forma rápida y fácil de introducir texto en cualquier pantalla táctil. Con un movimiento continuo del dedo o del lápiz a lo largo del teclado en pantalla, esta tecnología permite a los usuarios introducir palabras de forma más rápida y sencilla que otros métodos de entrada de datos (unas cuarenta palabras por minuto). La aplicación está diseñada para trabajar con una gran variedad de dispositivos (como móviles, *tablets*, consolas de videojuegos, televisiones o pantallas virtuales).

En la imagen podéis ver un ejemplo que ilustra cómo se puede generar la palabra *quick* trazando el camino mostrado en una fracción de segundo, simplemente tratando de pasar por las letras de la palabra. Una ventaja clave de Swype es que no es necesario ser demasiado preciso, lo que permite una entrada de texto muy rápida.

Teclado Swype

Siine

El segundo ejemplo de teclado *software* que vamos a tratar es Siine. Si bien con Swype se aprecia una mejora en la forma de introducir los datos, tras la filosofía de Siine está la premisa de que cambiar la forma de introducir los datos no es suficiente. Siine aporta la comunicación textual. No se trata de una simple aplicación, sino de una nueva forma de comunicación, un nuevo traductor universal, un sistema que habla "por el usuario".

Como hemos visto, algunas empresas ya se han replanteado el uso del teclado y lo han reemplazado por programas como Swype, que permite escribir con solo mover los dedos por la pantalla o Swiftkey, el cual se adelanta y piensa la palabra que vamos a escribir en ese momento (únicamente debemos introducir los primeros caracteres). El siguiente paso es la comunicación textual.

En este contexto aparece Siine, una herramienta de comunicación que permite mandar SMS desde la pantalla blanca de Android. A diferencia de *Swype* o *Swiftkey*, no solo introduce las palabras, sino que va más allá. Si se utilizan siglas, Siine lo detecta y puede decir lo que significan. Con Siine no se necesita ningún diccionario, ya que puede traducir. El sistema explota los significados incorporados. Siine es un sistema que incorpora inteligencia y significado, todo para facilitar la comunicación entre los usuarios.

Ejemplo

Si se termina con un "hasta pronto", el programa lo escribe por el usuario, ya que lo conoce y sabe que suele hacerlo así.

Snapkeys

Snapkeys es un "teclado" virtual que permite teclear textos sin mirar y de forma rápida. La peculiaridad de este "teclado" es que, sencillamente, no tiene teclas. Aunque no pueda parecer muy lógico, se puede llegar a escribir relativamente rápido sin teclado alguno en la pantalla. El concepto que hay detrás es que el teclado se lo tiene que imaginar el usuario (2i) para así poder escribir rápidamente. Los creadores aseguran que para los usuarios novatos es preciso en un 92% y para los experimentados en un 99%, y que se pueden teclear entre sesenta y ciento sesenta caracteres por minuto.

Comparativa entre Snapkeys y un teclado QWERTY virtual

Características	2i	QWERTY (en pantalla)
Número de teclas	0	101
Pantalla disponible	100%	10-50%
Escribir sin ver	Instantánea	Casi nunca
Aprender a escribir	Fácilmente	Difícilmente
Movimiento de dedos	Mínimo	Demasiado
Velocidad de entrada de datos	Muy rápida	Lenta
Interfaz	En su mente	Compleja
Escribir textos largos	Es un placer	Es cansado
Escribir en movimiento	Sí	Difícil
Divertido	Como un videojuego	Jamás
Rediseño	No se requiere	Necesario

Otra ventaja de este teclado es la gran comodidad, ya que no hay que desplazar los dedos por toda la pantalla para poder escribir; basta con tenerlos a cada lado de la pantalla.

El truco de este teclado consiste en que los desarrolladores han colocado cuatro bloques de letras en cuatro cuadrados, que se encuentran en la parte inferior de la pantalla, de modo que para teclear tenemos que pulsar encima de los cuadrados, y luego estos se vuelven transparentes sobre la interfaz que estemos utilizando, para no molestar al usuario.

Como último dato importante, deciros que esta aplicación está disponible para varias plataformas (tanto de móviles como de *tablets*).

Blindtype

Uno de los peores aspectos de Android era su teclado virtual. No era, ni mucho menos, el más cómodo del mundo, y sus predicciones dejaban bastante que desear. Google lo sabía y, por eso, compró BlindType, una *startup* dedicada a mejorar los teclados virtuales en *smartphones*.

Android lo compró porque podría ganar muchos puntos si implementaba este teclado mejorado, ya que aliviaría uno de los mayores dolores de cabeza que dan los teléfonos táctiles.

Lo malo de esta adquisición es que el desarrollo del teclado para iOS se paró y dejó a estos usuarios sin disfrutar de esta opción para el teclado. Los de Android sí que pueden disfrutar de este teclado, además de las alternativas como Swype, SwiftKey o SlideIT.

Graffiti

Graffiti no es un teclado propiamente dicho, sino un sistema de entrada de texto muy preciso en el que se emplea un lápiz. Con este lápiz se pueden escribir rápidamente letras y números en el área de escritura Graffiti. Este es un sistema que incorporan los ordenadores de mano Palm, junto con una interfaz gráfica muy intuitiva. Este sistema reconoce una gama completa de letras mayúsculas y minúsculas, así como dígitos, puntuación y símbolos especiales. En la figura podéis ver ejemplos de trazos para letras, así como el espacio y el equivalente a la tecla de borrar.

Introducción de texto con Graffiti

3.2. Pantallas de los dispositivos móviles

3.2.1. Pantallas táctiles

La tecnología está evolucionando a una gran velocidad. Prácticamente a diario, alguna compañía lanza un nuevo dispositivo. El mejor ejemplo son los teléfonos móviles. Hubo días en que la gente tenía grandes teléfonos que eran llamados *móviles*. En realidad lo eran, pero no tienen mucho que ver con los teléfonos móviles actuales. Ahora los teléfonos móviles son compactos, prácticos, tienen diseños atractivos y, en definitiva, presentan una gran cantidad de características que los hacen algo más que teléfonos móviles. Actualmente, con un móvil podemos hacer llamadas, escuchar música, ver películas, hacer fotos, etc. Además, parece que la generación de teléfonos móviles con botones se está acercando a su fin y que los dispositivos con pantalla táctil se están haciendo con el mercado. Las pantallas táctiles tienen tamaños grandes y son muy cómodas para el usuario. Las pantallas táctiles tienen sus pros y sus contras, pero para saber cuáles son es necesario que conozcáis primero los diferentes tipos de pantallas táctiles. Hay, básicamente, tres tipos de pantallas táctiles que se usan actualmente en los móviles. Os las presentamos, de manera breve, a continuación.

Pantalla táctil resistiva

Las pantallas táctiles resistivas son las más usadas en los teléfonos móviles. Son baratas y resistentes al agua y al polvo, pero se rompen con facilidad y no se pueden usar con objetos afilados. Podemos tocarlas con cualquier objeto (con los dedos, con el *stylus*, etc.). Es necesario aplicar una ligera presión para que detecte la pulsación. Tienen un tiempo de vida útil muy bastante largo (se estima que alrededor de treinta y cinco millones de pulsaciones).

La pantalla se compone, básicamente, de dos capas, entre las cuales hay una pequeña capa de puntos de plástico. Se proporciona electricidad a cada una de las capas y, en caso de contacto, se forma un circuito. La cantidad de electricidad que pasa entre las capas se mide para determinar el punto de contacto. El principal inconveniente de este tipo de pantallas táctiles es que no pueden

reconocer dos contactos en el mismo momento. Además, son más propensas a romperse (y la sensibilidad se reduce cuando esto ocurre) y solo dejan salir el 85% de la luz emitida realmente por la pantalla.

Algunos ejemplos de pantallas táctiles resistivas son las del HTC Diamond o el LG Viewty.

HTC Diamond (izquierda) y LG Viewty (derecha)

Pantalla táctil capacitiva

Las pantallas táctiles capacitivas pueden ser, básicamente, de dos tipos: uno de ellos puede reconocer múltiples contactos simultáneamente y el otro no. En cualquier caso, estas pantallas táctiles son más caras que las resistivas. Son resistentes a los golpes, a la humedad y al polvo.

> La pantalla táctil capacitiva usa una única capa (conocida como *grid*). Esta capa está cubierta por un material electroconductor que proporciona corriente continua con una cierta frecuencia. Cuando se toca la pantalla con un objeto que emite un flujo de electricidad constante, como, por ejemplo, un dedo (el cuerpo humano genera electricidad), se produce un cambio en la corriente y, de esta forma, se determina el punto de contacto.

Tiene un tiempo de vida extremadamente largo (alrededor de doscientos veinticinco millones de pulsaciones). Además, deja pasar alrededor del 92% de la luz emitida por la pantalla.

Ejemplos de pantallas táctiles capacitivas son las del Apple iPhone, el LG Prada o el Samsung F480.

iPhone (izquierda) y Samsung F480 (derecha)

Pantallas táctiles infrarrojas

Las pantallas táctiles infrarrojas son las más caras de las tres que hemos mencionado. No requieren fuerza física (un toque suave es suficiente). Además, no se ven influenciadas por el polvo, la humedad o los rasguños. También son las más duraderas. Las hay de dos tipos: ópticas y sensibles al calor.

Las ópticas usan haces infrarrojos, que no son visibles para el ojo humano. Funcionan con sensores situados encima y alrededor de la pantalla, que forman una rejilla de haces invisibles. Si un objeto (dedo o *stylus*) toca la pantalla, interrumpe los rayos en una cierta área y, de esta forma, se determina el punto de contacto. Tiene un tiempo de vida de unos siete años y presenta una seria desventaja: un ambiente con mucha luz puede tener un impacto negativo en su funcionamiento.

Las sensibles al calor son las más usadas, pero raramente se usan en pantallas. Se aplican en otros componentes de los dispositivos móviles, tales como los botones.

Móviles táctiles infrarrojos

El Samsung SGH-E900 y el Samsung U600 utilizan la tecnología de pantalla táctil sensible al calor en sus teclados.

Esta tecnología solo funciona con objetos calientes, de modo que tienen un serio problema: si se tienen los dedos fríos (como puede ocurrir en invierno) y se tocan estos teléfonos, puede pasar que el dispositivo no responda.

3.2.2. Pantallas de tinta electrónica

La tinta electrónica o papel electrónico es una tecnología que permite crear pantallas planas. Estas pantallas representan información en blanco y negro y no permiten visualizar imágenes en movimiento. En el 2007 apareció el primer papel electrónico en color.

En abril de 1997, los investigadores del Media Lab del MIT[11] crearon la compañía E Ink para desarrollar una tecnología de tinta electrónica.

(11)MIT (Massachussets Institute of Technolgy): Instituto de Tecnología de Massachusetts

En julio del 2002, E Ink presentó el prototipo de la primera pantalla con esta tecnología. Esta pantalla se comercializó en el 2004. Le siguieron otras pantallas para varias tabletas de lectura.

La tinta electrónica es actualmente la principal tecnología utilizada en los lectores de *e-books*, también conocidos como libros electrónicos.

3.3. Sensores

Una tecnología importante en el mundo de Internet y de los dispositivos móviles es la de los sensores. Por eso, en este subapartado exploraremos cómo se están relacionando los teléfonos móviles y los sensores y qué implicaciones tiene o puede tener unir estos dos mundos.

Hay dos escenarios comunes que aúnan los conceptos de sensores y teléfonos móviles:

- **Objetos cotidianos con sensores que generan datos**. El teléfono móvil lee y analiza datos como la temperatura, el ruido y la actividad.

- **El teléfono usado como un sensor en sí mismo**. Por ejemplo, el iPhone incorpora un acelerómetro, que es básicamente un sensor de movimiento. Se usa como control en juegos y también para cambiar la disposición de la pantalla de vertical a horizontal. El iPhone también tiene un micrófono (que se puede usar como sensor de ruido), un sensor de proximidad y un sensor de luz ambiental.

WideNoise

Un buen ejemplo del escenario en el que el teléfono se usa como sensor es WideNoise, una aplicación para iPhone que toma muestras de los niveles de ruido en decibelios y muestra los datos en un mapa interactivo. WideNoise es, básicamente, un sensor de sonido que usa el micrófono del iPhone.

Se pueden tomar lecturas de sonido con WideNoise y, si se desea, compartir esa información con la comunidad. Un caso de uso podría ser cuando se está buscando piso, para chequear los niveles de ruido medio del vecindario. Por lo tanto, es una de esas aplicaciones que se vuelve más útil cuantos más datos añade la comunidad de usuarios.

En cuanto al primer escenario, podemos decir que los sensores están creciendo muchísimo como fuente de datos en Internet. Prueba de ello es que las redes de sensores representan una gran oportunidad para algunas de las compañías de tecnología más importantes. Los sensores permiten la recogida de datos en tiempo real, el análisis y una mejor toma de decisiones. Los dispositivos móviles pueden ser los receptores perfectos de esa información.

Ejemplo

Podemos utilizar sensores de temperatura para alarmas de incendio en el bosque. También podemos obtener información en tiempo real de las condiciones de tráfico en una sección importante de una autopista en el teléfono móvil, vía sensores.

Estas son las dos formas principales en las que los sensores y los teléfonos móviles están interaccionando.

En el siguiente subapartado estudiaremos en detalle los sensores más paradigmáticos de la nueva generación de dispositivos móviles: los sensores de movimiento.

3.3.1. Sensores de movimiento

Cuando apareció el iPhone, una de las características más llamó la atención, aparte de su diseño y de su pantalla táctil, fue el sensor de movimiento. Aunque el iPhone es un ejemplo paradigmático, por supuesto que no es el único dispositivo que incorpora e incorporará este tipo de sensores. Fuera del mundo de los móviles, podemos encontrar, por ejemplo, el caso de la Nintendo Wii, y en la familia de dispositivos Android también encontramos dispositivos con este sensor, como, por ejemplo, en la gama de dispositivos Samsung Galaxy.

Los sensores de movimiento en dispositivos móviles han penetrado de forma rápida en el mercado y aún se están explorando las posibles aplicaciones que se le pueden dar. Por supuesto, se pueden usar como control para juegos o para detectar qué orientación de pantalla hay que mostrar en función de si sostenemos el dispositivo horizontal o verticalmente, o para monitorizar actividades deportivas. Sin embargo, puede haber más aplicaciones.

Hay referencias de sensores de movimiento que se usan para reconocer el patrón con el que camina una persona. Un ejemplo de uso podría ser el del bloqueo antirrobo del terminal (la forma diferente de caminar del ladrón alerta al teléfono, que entonces pide una contraseña).

Este tipo de sensores serían mucho más importantes si se encontrara una aplicación apetecible para los operadores; es decir, una aplicación cuyos usos no fueran únicamente locales, como los explicados anteriormente. Dado que los operadores especifican (y certifican y, a menudo, subvencionan) muchos dispositivos, tener algo apetecible tanto para el usuario final de las aplicaciones como para el operador es importante para su introducción en el mercado. Podemos pensar, por ejemplo, en las cámaras (el usuario hace fotografías y el operador consigue el beneficio del MMS[12] o del correo electrónico, o incluso en el Bluetooth (el usuario usa el manos libres en el coche de forma segura y el operador consigue más minutos de uso). Este es uno de los motivos por los que la incorporación de la tecnología WiFi a los móviles no ha ido tan rápido como hubiera cabido esperar, ya que a menudo es difícil explotarla para sus propósitos, tanto para el usuario como para el operador.

(12)MMS (*multimedia message system*)

Esto último nos lleva a preguntarnos cómo podrían ser los servicios basados en el sensor de movimiento. En este tipo de servicios podríamos decir que el contexto sería más importante que el contenido. Si los operadores tuvieran acceso a los sensores, podrían saber mucho más acerca de cómo un usuario quiere comunicarse. Podrían determinar descripciones de estado como "caminando", "en un tren", "en un coche", etc. También podemos pensar en servicios que usen datos multicontexto: si el teléfono está cargándose y no ha ha-

bido movimiento durante una hora, entonces hay una alta probabilidad de que el usuario esté fuera de la habitación o dormido. O, por ejemplo, con un servicio que detectase la combinación del patrón de movimiento de un coche y registrara que se está usando un manos libres Bluetooth, se podría inferir que el usuario no puede mirar la pantalla, y entonces se le podrían enviar las videollamadas directamente al buzón.

Estos son simplemente algunos ejemplos de lo que se podría llegar a hacer con estos sensores. Por supuesto, todos estos servicios pueden sufrir con lo que se llaman falsos positivos y falsos negativos.

En cualquier caso, el sensor de movimiento es una parte esencial del paradigma multicontexto, que podría ser el siguiente paso en los dispositivos móviles, después del contenido multimedia.

3.4. Conectores

3.4.1. Conectores Mini USB y Micro USB

A lo largo de los años se han usado varios conectores USB para dispositivos pequeños como PDA, teléfonos móviles o cámaras digitales. Estos incluyen el ya en desuso (pero estandarizado) Mini-A y los conectores estándares actuales Mini-B, Micro-A y Micro-B. Los conectores Mini-A y Mini-B tienen un tamaño de, aproximadamente, tres por siete milímetros.

Los conectores Micro-USB tienen una anchura similar, pero un grosor de aproximadamente la mitad, de manera que se pueden integrar en dispositivos portables más finos.

El conector micro-USB fue anunciado por el USB-IF[13] el 4 de enero del 2007. El conector Mini-A y el conector-receptáculo Mini-AB fueron desaprobados el 23 de mayo del 2007. Muchos dispositivos y cables aún usan mini-conectores, pero los últimos micro-conectores se están imponiendo y son los conectores más usados. Los micro-conectores, más finos, surgieron para reemplazar a los conectores Mini en los nuevos dispositivos que iban apareciendo, como *smartphones* y PDA. El diseño del micro-conector está pensado para aguantar al menos diez mil ciclos de conexión-desconexión, un valor significativamente mayor que el del diseño del mini-conector. La *universal serial bus Micro-USB cables and connectors specification* detalla las características mecánicas de los conectores Micro-A, los receptáculos Micro-AB y los conectores y receptáculos Micro-B, además de un adaptador receptáculo estándar-A a conector Micro-A.

(13) USB-IF (USB Implementers Forum)

El grupo de operadores de telefonía celular OMTP[14] propuso en el 2007 el Micro-USB como el conector estándar para datos y recarga de batería en dispositivos móviles. Esto incluye varios tipos de cargadores de batería y, así, permite que el Micro-USB sea el único cable externo necesario para algunos dispositivos.

(14)OMTP (*open mobile terminal platform)*

A principios del 2009, el Micro-USB fue aceptado y usado por casi todos los fabricantes de teléfonos móviles (como HTC, Motorola, Nokia, LG, Hewlett-Packard, Samsung, Sony Ericsson y Research In Motion) como el puerto de carga estándar en la mayor parte del mundo. La GSMA[15], junto con diecisiete fabricantes y proveedores, anunció su acuerdo para implementar un estándar para toda la industria para un cargador universal para los nuevos teléfonos móviles.

(15)GSMA (GSM *Association*)

En junio del 2009, después de una petición de la Comisión Europea y en estrecha cooperación con los servicios de la Comisión, los principales fabricantes de teléfonos móviles llegaron a un acuerdo para armonizar los cargadores para teléfonos móviles de última generación vendidos en la Unión Europea. Gracias a este acuerdo, la industria acepta proporcionar compatibilidad entre los cargadores sobre la base del conector Micro-USB. Por lo tanto, los consumidores podrán comprar teléfonos móviles sin un cargador y, de esta manera, reducir su coste. Siguiendo las órdenes de la Comisión Europea, los cuerpos de estandarización europeos CEN-CENELEC[16] y ETSI[17] han publicado los estándares necesarios para fabricar teléfonos móviles compatibles con el nuevo EPS[18] común basado en Micro-USB.

(16)CENELEC (European Committee for Electrotechnical Standardization)

(17)ETSI (European Telecommunications Standards Institute)

(18)EPS (*external power supply*)

Además, el 22 de octubre del 2009, la ITU[19] anunció que usaría Micro-USB para su UCS[20], ya que por su eficiencia energética se adapta a todos los nuevos teléfonos móviles, y basándose en la interfaz Micro-USB, añadió que los cargadores UCS tendrían una calificación de cuatro estrellas o mayor en cuanto a la eficiencia (unas tres veces más eficiente que un cargador no calificado).

(19)ITU (International Telecommunication Union)

(20)UCS (*universal charger solution*)

3.5. Baterías

Esencialmente, una batería es un recipiente de químicos que transmite electrones. Es una máquina electroquímica; o sea, una máquina que crea electricidad mediante reacciones químicas.

Las baterías tienen dos polos, uno positivo (+) y otro negativo (–). Los electrones (de carga negativa) van del polo negativo hacia el polo positivo; es decir, que son recogidos por el polo positivo. A no ser que los electrones corran del polo negativo hacia el polo positivo, la reacción química no ocurre. Esto significa que la electricidad solo se genera cuando se conectan los dos polos (por ejemplo, al usarla en un teléfono móvil, la batería casi no se gasta si está guardada en un cajón).

Las pilas modernas son, generalmente, pilas secas (usan sólidos como electrolitos) y pueden basarse en una gama muy variada de químicos.

Para los teléfonos móviles, existen tres tipos de baterías: las NiCd[21], las NiMH[22] y las de litio.

[21] NiCd (níquel-cadmio)

[22] NiMH (níquel-hidruro metálico)

Inicialmente, las baterías más comunes para los teléfonos móviles eran de NiMH, debido a que tienen un tamaño y un peso reducido. También se usan a veces las baterías de iones de litio, ya que son más ligeras y no tienen la depresión de voltaje que tienen las baterías de NiMH. Muchos fabricantes de teléfonos móviles han cambiado a las baterías de polímeros de litio. Las principales ventajas con respecto a las más antiguas de iones de litio son las siguientes: un peso incluso menor y la posibilidad de hacer que la forma de la batería no sea únicamente la de un cubo. Los fabricantes de móviles han estado experimentando con otras fuentes de energía, entre las que se incluyen células solares.

La tecnología de las baterías es complicada y cara, y esa es una de las razones por las que su precio no ha decrecido como el precio de otros componentes. El futuro de las baterías podría pasar por el uso de polímeros o de microcélulas de metanol, que aumentarían la capacidad de las baterías alrededor de cincuenta veces. Por otro lado, algunas entidades están desarrollando chips capaces de disminuir considerablemente las necesidades de energía de los teléfonos móviles.

De hecho, las baterías de polímeros de litio ya son una realidad. Utilizadas inicialmente por Ericsson y ahora difundidas por los demás fabricantes, se parecen mucho a las baterías de litio ya conocidas. La flexibilidad es su principal ventaja y prometen revolucionar, además del mercado de las baterías, el diseño futuro de los teléfonos móviles, ya que estas nuevas "perlas" moldeables podrán producirse en láminas con el espesor de un milímetro, lo que se traducirá posiblemente en teléfonos móviles cuyo diseño sea más vanguardista. Poseen, además, un ciclo de carga y descarga superior a su congénere rígida; o sea, menos espacio, menos peso y más autonomía.

3.5.1. Baterías de litio, Li-Ion o iones de litio

Las baterías basadas en iones de litio son las baterías más recientes en el mercado de los teléfonos móviles y las únicas que se utilizan en la actualidad.

Consiguen un almacenamiento muy superior de energía, de modo que se aumenta considerablemente el tiempo de actividad del teléfono móvil. Son también muy ligeras (pesan cerca de la mitad de una NiCd equivalente).

A pesar del elevado precio, las baterías de litio se han popularizado gracias a sus ventajas, las cuales han hecho que se tornen equipos de serie para muchos modelos de teléfonos móviles.

3.5.2. Baterías NiMH

Las baterías NiMH, que usan hidrógeno en su proceso de producción de energía, nacieron en los años setenta de las manos del químico Standford Ovshinsky, pero solo recientemente fueron redescubiertas para los teléfonos móviles. La inusual tecnología de las NiMH permite el almacenamiento de mucha más energía. Normalmente consigue almacenar alrededor de un 30% más de energía que una NiCd de idéntico tamaño. Estas baterías tampoco usan metales tóxicos, por lo que se las considera respetuosas con el medio ambiente.

Muchas de estas baterías están hechas con metales como el titanio, el circonio, el vanadio, el níquel y el cromo, y algunas empresas japonesas han experimentado, incluso, con otros metales como el raro lantano. Este detalle torna las baterías NiMH mucho más caras que las NiCd.

En la actualidad han caído en desuso.

3.5.3. Baterías NiCd

Las baterías de níquel y cadmio (NiCd) para teléfonos móviles están totalmente en desuso. En estas baterías, el polo positivo y el polo negativo se encuentran en el mismo recipiente. El polo positivo está cubierto con hidróxido de níquel y el polo negativo está cubierto de material sensible al cadmio. Ambos están aislados por un separador.

Las baterías NiCd van perdiendo su tiempo de duración. Cada vez que se recargan, el periodo entre recargas se va acortando. El voltaje de las NiCd tiende a caer de forma abrupta, de manera que se descargan de un momento a otro después de un largo periodo de uso.

En la actualidad solo se usan en algunos coches de radiocontrol o aeromodelismo con motor eléctrico, de modo que aprovecha su capacidad de liberar toda su energía en poco tiempo sin dañarse.

3.5.4. Cargadores

Como ya hemos visto, los teléfonos móviles obtienen generalmente la energía de baterías recargables. Hay varias formas de recargar estas baterías, entre las que se encuentran el USB, las baterías portables, los enchufes (con un adaptador AC) y los mecheros eléctricos (con un adaptador).

Los cinco principales fabricantes de dispositivos móviles presentaron en noviembre del 2008 un nuevo sistema de calificación para ayudar a los consumidores a identificar más fácilmente los cargadores más eficientes en cuanto a energía.

La mayor parte de la pérdida de energía en un teléfono móvil se produce cuando no lo estamos cargando, cuando el teléfono móvil no está conectado, pero se deja el cargador enchufado al móvil y usando energía. Para combatir esto, Nokia, Samsung, LG, Sony Ericsson y Motorola confeccionaron un sistema de calificación basado en estrellas para calificar la eficiencia de sus cargadores en la condición de no-carga antes mencionada. Empezando por cero estrellas para pérdidas inferiores a 0,5 W, el sistema llega hasta las cinco estrellas para pérdidas superiores 0,03 W (30 mW).

En el 2009 lanzaron el primer cargador inalámbrico al mercado. La ventaja de los cargadores inalámbricos es que se pueden recargar varios dispositivos a la vez con la máxima comodidad posible. Funciona de la siguiente manera: se coloca un dispositivo con su receptor encima de una especie de alfombrilla y se recarga mediante una atracción magnética.

3.6. Otras características de los dispositivos móviles

3.6.1. Cámaras

Una de las características cada vez más comunes de los dispositivos móviles es la incorporación de cámaras. Dentro de este ámbito, los dispositivos se diferencian unos de otros según las siguientes características:

- **Resolución para fotos**: La resolución es un indicador de la calidad de las fotos.

- **Resolución para vídeos**: La resolución también es un indicador de la calidad de los vídeos.

- Existencia de ***flash*** y tipo de este (por ejemplo, podemos encontrarnos con un *flash* LED, como es el caso del BlackBerry Curve 8900).

- Si tiene ***zoom* digital**: El *zoom* digital es un método para, aparentemente, disminuir el ángulo de visión de una imagen fotográfica o de un vídeo.

- Si tiene ***zoom* óptico**: Un *zoom* óptico es un objetivo que permite variar la distancia focal y, por lo tanto, abarcar un mayor o menor campo visual.

- Si el dispositivo dispone de una **cámara secundaria** para vídeo llamadas: Esta cámara se encontrará, por tanto, en la parte frontal del dispositivo.

- Si el dispositivo tiene **herramientas de *geo-tagging***: Estas herramientas permiten etiquetar nuestras imágenes con una referencia al lugar donde han sido tomadas.

- Si tiene **detección de caras**: La función de la detección de caras es detectar los rostros de las personas que se encuentran dentro del cuadro y mantener el foco fijo sobre ellas.

- Si tiene **detección de sonrisas**: El detector de sonrisas es una función que lleva más allá el sistema de detección facial. Consiste en la activación automática del obturador de la cámara cuando el sujeto encuadrado sonríe.

- Si tiene **auto *focus***: El *auto focus* es un automatismo que permite el enfoque automático de un objeto.

- Si tiene la capacidad de tomar **imágenes en 3D**.

3.6.2. *Trackballs*

Un *trackball* es un dispositivo para apuntar que consiste en una bola alojada en un hueco provisto de sensores que detectan la rotación de la bola sobre dos ejes (como un ratón mecánico puesto al revés). El usuario mueve la bola con el pulgar, el resto de dedos o la palma de la mano para mover un cursor.

Los *trackballs* grandes son comunes en estaciones de trabajo CAD[23] para facilitar la precisión. Antes de la llegada del *touchpad*, pequeños *trackballs* eran comunes en ordenadores portátiles, ya que estos dispositivos se usan a veces en situaciones en las que no se puede usar un ratón convencional.

(23)CAD (*computer aided design*)

Invención del *trackball*

El *trackball* lo inventaron Tom Cranston y Fred Longstaff como parte del sistema DATAR (*digital automated tracking and resolving*) de la Royal Canadian Navy en 1952, once años antes de que se inventara el ratón. Este primer *trackball* usaba una bola de bolos canadienses.

Algunos teléfonos móviles tienen *trackballs*, entre ellos los de la famila BlackBerry, los T-Mobiles Sidekick 3 y muchos *smartphones* HTC. Estos *trackballs* en miniatura están hechos para encajar en el grosor de un dispositivo móvil y se controlan con la punta de un dedo (el pulgar u otro).

Artemis, Sidekick, Pearl, etc., son todos ejemplos de *trackball* móvil. Estos mini-ratones son prácticos para moverse por la pantalla y, aunque pueda parecer que no tienen demasiado sentido en dispositivos con pantalla táctil, en los que se tiene un acceso directo a la misma, los *trackballs* son más precisos, ya que permiten desplazarse entre caracteres de una palabra más fácilmente. Si no, son necesarias aplicaciones de *zoom* para usar solamente la pantalla táctil.

4. Posibles redes a las que puede acceder un dispositivo móvil

Muchos teléfonos móviles disponibles actualmente soportan tanto tecnologías celulares como otras tecnologías de banda ancha inalámbricas. Por ejemplo, los dispositivos Nokia N Series y E Series soportan tanto GSM como WiFi. Dispositivos similares de Sony Ericsson, BlackBerry, Samsung, HTC, Motorola e incluso el iPhone (de Apple) proporcionan la misma doble tecnología. También se está añadiendo soporte a WiMAX, y otros dispositivos móviles que proporcionan esta doble tecnología en teléfonos CDMA están apareciendo de la mano de Kyocera y otros vendedores. Esto abre la puerta a un gran rango de aplicaciones de Internet a las que se puede acceder desde los dispositivos móviles mediante tecnologías inalámbricas de banda ancha como WiFi y WiMAX.

4.1. Redes para conseguir llamadas de voz

4.1.1. ¿Cómo se produce la comunicación?

Para poder realizar llamadas de voz, los teléfonos tienen que conectarse a una red celular. La operadora de telefonía móvil correspondiente reparte el área en varios espacios llamados células. En cada célula existe una estación base transmisora, que normalmente es una simple antena. Cada célula consigue utilizar varias decenas de canales, lo que da la posibilidad de que varias decenas de personas se comuniquen de forma simultánea por ella.

Cuando una persona se mueve de una célula a otra, pasa a utilizar la frecuencia de la nueva célula y, por lo tanto, deja libre la célula anterior para que pueda ser usada por otro usuario. Como las distancias de transmisión no son muy grandes, los teléfonos móviles pueden transmitir con poca energía y, por lo tanto, utilizar pequeñas baterías (que permiten un tamaño y un peso reducido). Es, por lo tanto, el concepto de célula lo que hace posible que existan los teléfonos móviles tal como los conocemos actualmente (de ahí la expresión "teléfonos celulares").

Skype

Gracias a aplicaciones como Skype, también es posible realizar llamadas de voz mediante, por ejemplo, una red WLAN.

4.1.2. Sistemas de telefonía móvil

En cuanto a los sistemas de telefonía, el primero de ellos fue **GSM**[24], que fue diseñado originalmente para transmitir voz, aunque con el tiempo, la tecnología posibilitó también operar en modo de transferencia de datos. Los terminales operan por conmutación de circuitos. Esto implica que hay una fase de

[24]GSM (*global system for mobile communications*)

establecimiento de conexión que conlleva tiempos de espera y que la llamada se mantenga abierta, aun cuando no existe transferencia de datos. Esta forma de transmisión es extremadamente limitada en lo que respecta a la capacidad, incluso con el uso de la tecnología HSCSD[25], que permite una velocidad máxima de 56 Kbps.

(25)HSCSD (*high speed circuit switched data*): circuito de datos conmutado de alta velocidad

El estudio de las limitaciones de GSM origina la necesidad de un sistema basado en la transmisión de datos por paquetes. En 1998, el ETSI[26], la entidad reguladora de las telecomunicaciones europeas, concluyó sus estudios sobre la definición de las normas de un nuevo sistema, el **GPRS**[27], que permite una mayor capacidad de transmisión de datos. El GPRS permite una velocidad máxima teórica de 144 Kbps, en el caso de que utilice todos los recursos del sistema. Finalmente, el GPRS hacía posible toda una nueva serie de aplicaciones dentro de los móviles, apenas accesibles hasta ese momento, tales como la visualización de sitios, FTP, IRC, animación, etc. En resumen, el GPRS aportaba los siguientes beneficios:

(26)ETSI (European Telecommunications Standards Institute)

(27)GPRS (*general packet radio service*)

- Conexión a Internet permanente (siempre "en línea").
- Establecimiento instantáneo de la conexión.
- Posibilidad de que la facturación del servicio se realizara según la cantidad de información transmitida o recibida, en lugar de contabilizar el tiempo de conexión.
- Una mayor velocidad de transmisión de datos.

Como tecnología puente entre las redes 2G y 3G encontramos la tecnología de telefonía móvil EDGE[28]. Esta tecnología también se conoce como EGPRS[29]. Por lo tanto, EDGE se considera una evolución del GPRS. Estas tecnologías trabajan en las bandas de 850, 900, 1.800 y 1.900 MHz.

(28)EDGE (*enhanced data rates for GSM evolution*): tasa de datos mejorada para la evolución de GSM

(29)EGPRS (*enhanced GPRS*)

El **UMTS**[30] ha sido el nuevo protocolo utilizado en Europa por la tercera generación de teléfonos móviles. Integrado en el proyecto de crear un estándar que pudiera ser utilizado mundialmente (al contrario que segunda generación, cuyos sistemas americano y europeo son incompatibles), el UMTS alteró la forma en que se podían utilizar los móviles, al permitir capacidades multimedia y un acceso sin límites a Internet.

(30)UMTS (*universal mobile telecommunication system*)

Además de las funciones básicas que permitían los móviles hasta entonces, como simplemente telefonear a alguien o enviar y recibir mensajes, el UMTS permitió acrecentar una nueva serie de características hasta entonces casi inaccesibles o presentes. El sistema permitía el acceso a Internet a una velocidad más rápida y, por lo tanto, la transmisión de faxes, imágenes, vídeos y datos. Permite hacer videollamadas (mientras estamos hablando, podemos visualizar en la pantalla, en tiempo real, a la persona con quien nos comunicamos, en

caso de que esta también posea un móvil UMTS). El acceso a Internet es bastante más rápido y carece de límites, de modo que podemos acceder a cualquier tipo de información desde cualquier lugar. Información, comercio y entretenimiento multimedia están disponibles en la pantalla, en un sistema que integra las redes de telecomunicaciones móviles, fijas y por satélite. Además del *roaming* a escala mundial, el UMTS permitía la convergencia de los varios tipos de redes existentes.

Según la Comisión Europea, los servicios UMTS debían poseer las siguientes características:

- Capacidad multimedia y soporte a una gran movilidad.
- Acceso eficiente a Internet.
- Alta velocidad.
- Portabilidad entre los entornos UMTS (de manera que permitiera el acceso a las redes UMTS terrestres y de satélite).
- Compatibilidad entre el sistema GSM y el UMTS. Los terminales debían poseer *dual band* o funcionar en ambos sistemas.

Esta nueva tecnología alteraba radicalmente la manera de utilizar los móviles. Permitía que los usuarios tuvieran el móvil más tiempo delante de los ojos que pegado a la oreja, debido a que este pasaba a ser un dispositivo multimedia, como la televisión o un ordenador. Al mismo tiempo, la transmisión de datos ocuparía una parte mayor del tiempo de utilización del teléfono móvil, debido a todas las posibilidades existentes (enviar faxes, correos electrónicos, etc.). La calidad de voz pasaba a ser igual que la de los teléfonos fijos. En resumen, gracias a esta tecnología, era posible tener Internet en la palma de la mano.

Como optimización de la tecnología espectral UMTS / WCDMA, aparece la tecnología HSDPA[31], también denominada 3.5G, 3G+ o turbo 3G. Esta tecnología ofrece una velocidad de subida de hasta 2 Mbps y una velocidad de bajada de 7,2 Mbps en el espectro de frecuencias de 900 a 2.100 MHz.

(31)HSDPA (*high speed downlink packet access)*

Siguiendo esta evolución, llegamos al 4G. La red 4G estará totalmente basada en el protocolo IP[32]. Es un sistema de sistemas y una red de redes. Se pretende alcanzar la convergencia entre redes de cables e inalámbricas para proporcionar velocidades de acceso de entre 100 Mbps en movimiento y 1 Gbps en reposo, y mantener la calidad de servicio punta a punta, así como la seguridad para poder ofrecer servicios de cualquier clase en cualquier momento y en cualquier lugar con el mínimo coste posible.

(32)IP (*Internet protocol)*

El WWRF[33] define 4G como una red que funciona en la tecnología de Internet que se combina con otros usos y tecnologías (tales como WiFi y WiMAX). 4G no es una tecnología o estándar definido, sino una colección de tecnologías y protocolos que permiten el máximo rendimiento con la red inalámbrica más barata.

(33)WWRF (Wireless World Research Forum)

El concepto de 4G englobado dentro del "más allá de 3G" incluye técnicas inalámbricas de alto rendimiento como MIMO y OFDM. Dos de los términos que definen la evolución de 3G, siguiendo la estandarización del 3GPP, serán LTE[34] para el acceso radio y SAE[35] para la parte núcleo de la red. Como características principales tenemos las siguientes:

(34)LTE (*long term evolution*)

(35)SAE (*service architecture evolution*)

- Para el acceso radio, abandona el acceso tipo CDMA característico de UMTS.
- Uso de SDR[36] para optimizar el acceso radio.
- Toda la red es IP.
- Las tasas de pico máximas previstas son de 100 Mbps en enlace descendente y de 50 Mbps en enlace ascendente (con un ancho de banda en ambos sentidos de 20 Mhz).

(36)SDR (*software defined radios)*

4.2. Redes para tener acceso a Internet

Para tener acceso a Internet, cualquiera de las tecnologías mencionadas en el subapartado anterior es válida, siempre que medie un contrato de datos con el operador de telefonía móvil correspondiente.

Otra opción consiste en una conexión WLAN, tecnología disponible en la gran mayoría de los móviles y *tablets* actuales. WLAN es un sistema de comunicación de datos inalámbrico flexible, muy utilizado como alternativa a las redes LAN[37] cableadas o como extensión de estas. Esta opción nos permite acceder a Internet mediante una línea ADSL doméstica o mediante otras redes (infinidad de sitios públicos como aeropuertos, restaurantes, bibliotecas o cualquier lugar con un punto de acceso y una conexión a Internet detrás).

(37)LAN (*local area network)*

4.3. Redes para geolocalización

La geolocalización es un término nuevo que se ha venido usando desde mediados del 2009 y que hace referencia a la detección de nuestra ubicación geográfica de forma automática.

Hay varias maneras de que esto suceda y, como es natural, los dispositivos móviles son los que permiten más fácilmente la actualización de nuestra posición, por su portabilidad.

Desde hace algún tiempo, los teléfonos móviles de gama alta (y unos cuantos de gama media) traen integrados receptores GPS que, mediante la red de satélites que rodean el planeta, pueden ubicarnos en cualquier punto del globo.

Aunque GPS es la tecnología específica para la geolocalización y, además, es la forma más precisa de hacerlo, no es la única. Además, esta opción no es válida si estamos dentro un edificio, donde el receptor GPS del móvil no puede recibir la señal.

Otra forma de geolocalizar sin necesidad de tener que utilizar un receptor GPS en el móvil es mediante GPRS. Con la ayuda de las torres de telefonía celular se puede calcular la intensidad de la señal y triangular la posición estimada. No funciona con la misma precisión que GPS, pero se acerca bastante.

Por último, otra forma de geolocalizar es mediante una conexión WiFi establecida en ese momento. Esta forma de geolocalización no es exclusiva de los dispositivos móviles, cualquier ordenador con una conexión a Internet (no hace falta que sea inalámbrica) puede utilizarla. Esta forma de geolocalización funciona utilizando como fuente de información la dirección IP del equipo, junto con información de los puntos de acceso WiFi a los que tengamos acceso.

4.4. Redes para comunicaciones de corta distancia

Si se considera el alcance de una LAN como corta distancia, podríamos hablar en este subapartado de la tecnología WiFi en cualquiera de sus versiones (802.11 b/g/n). No obstante, este subapartado se centrará en tecnologías WPAN.

Para empezar, una de las tecnologías que implementan algunos dispositivos móviles es RFID[38]. RFID es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas RFID. El propósito fundamental de la tecnología RFID es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio. Las tecnologías RFID se agrupan dentro de las tecnologías de identificación automática.

[38]RFID (*radio frequency identification*): identificación por radiofrecuencia

La otra gran tecnología para comunicaciones de corta distancia es Bluetooth. Bluetooth es una especificación industrial para WPAN que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de 2,4 GHz. Las aplicaciones de Bluetooth son:

Teclado Bluetooth conectado a un ordenador de bolsillo

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre estos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Precisamente, los dispositivos que con mayor frecuencia utilizan esta tecnología pertenecen a sectores de telecomunicaciones e informática personal, como PDA, teléfonos móviles, ordenadores portátiles, ordenadores personales, impresoras o cámaras digitales.

Auricular para teléfono móvil con conexión Bluetooth

Los nuevos dispositivos (como, por ejemplo, el iPhone 4) incorporan la versión 2.1 de Bluetooth con EDR[39].

[39]EDR (*enhanced data rate*)

Resumen

Los dispositivos móviles representan una nueva oportunidad de acceso a la información por parte de los usuarios. Sin embargo, la diversidad de características de *hardware* y *software* que presentan estos dispositivos complica la definición misma del término *dispositivo móvil*. Por ello, en este módulo tratamos de clarificar este concepto enumerando las características genéricas que todo dispositivo móvil debe tener. En este módulo también repasamos los tipos de dispositivos móviles existentes con un enfoque histórico para que entendáis cómo han ido evolucionando estos dispositivos.

Entrando en detalle, en este módulo os explicamos las características específicas de los dispositivos (como tipos de teclado, pantalla, etc.), así como las diferentes redes a las que uno de estos dispositivos puede tener acceso o conectarse.

Glosario

gadget *m* Un *gadget* es un dispositivo que tiene un propósito y una función específicos, generalmente de pequeñas proporciones, práctico y novedoso a la vez. Los *gadgets* suelen tener un diseño más ingenioso que el de la tecnología corriente.

Bluetooth *m* Bluetooth es una especificación industrial para redes inalámbricas de área personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de 2,4 GHz.

CENELEC *m* CENELEC (European Committee for Electrotechnical Standardization) es el organismo responsable de la estandarización en el campo de la ingeniería electrotécnica.

ETSI *m* El European Telecommunications Standards Institute (ETSI) produce estándares aplicables de forma global para las *information and communications technologies* (ICT), donde se incluyen las tecnologías fijas, móviles, radio y *broadcast*.

WiFi *f* WiFi es una marca de la *WiFi Alliance*, la organización comercial que adopta, prueba y certifica que los equipos cumplen los estándares 802.11 relacionados con redes inalámbricas de área local.

WiMAX *f* WiMAX, siglas de *worldwide interoperability for microwave access* (interoperabilidad mundial para acceso por microondas), es una norma de transmisión de datos que utiliza las ondas de radio en las frecuencias de 2,3 a 3,5 GHz.

Bibliografía

Marco, M. J.; Marco, J. M.; Prieto, J. y otros (eds.) (2010). *Escaneando la informática*. Barcelona: Editorial UOC. ISBN: 978-84-9788-110-4.

Stark, J. (2010). *Building iPhone apps with HTML, CSS, and JavaScript*. Editorial O'Reilly. ISBN: 978-0-596-80578-4

Enlaces de Internet

http://www.ehow.com/

http://leo.ugr.es/J2ME/

http://www.hispanicbic.org/

http://www.wireless-center.net/

http://www.mobilechoiceuk.com/News/

http://the-gadgeteer.com/

http://www.bizzntech.com/

http://www.readwriteweb.com/

http://seekingalpha.com/

http://en.wikipedia.org/

http://mobiles.maxabout.com/

http://my.opera.com/usability/

http://www.htc.com/es/

http://www.parchegeek.com/

http://www.widetag.com/widenoise/

Entornos de programación móviles

Julián David Morillo Pozo

PID_00176754

UOC
Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

El desarrollo de aplicaciones móviles es el proceso por el cual se desarrolla un *software* para dispositivos móviles (como *smartphones* o *tablets*). La forma de distribución de estas aplicaciones puede variar, las aplicaciones pueden venir preinstaladas en los teléfonos o pueden ser descargadas por los usuarios desde *app stores* (tiendas de aplicaciones) y otras plataformas de distribución de *software*.

En este módulo veréis los diferentes entornos de programación para aplicaciones móviles existentes. Comenzaremos por una revisión histórica de su evolución. Después, revisaremos los diversos lenguajes de programación que se pueden utilizar en estos entornos y, por último, estudiaremos a fondo algunos de los entornos más populares.

Objetivos

Con el estudio de este módulo pretendemos que consigáis los objetivos siguientes:

1. Que conozcáis y comprendáis el concepto de entorno de programación en el ámbito del desarrollo de aplicaciones para dispositivos móviles.

2. Que identifiquéis los diversos entornos de programación existentes y conozcáis sus arquitecturas, las características de los sistemas operativos usados y las técnicas de programación requeridas en cada uno de ellos.

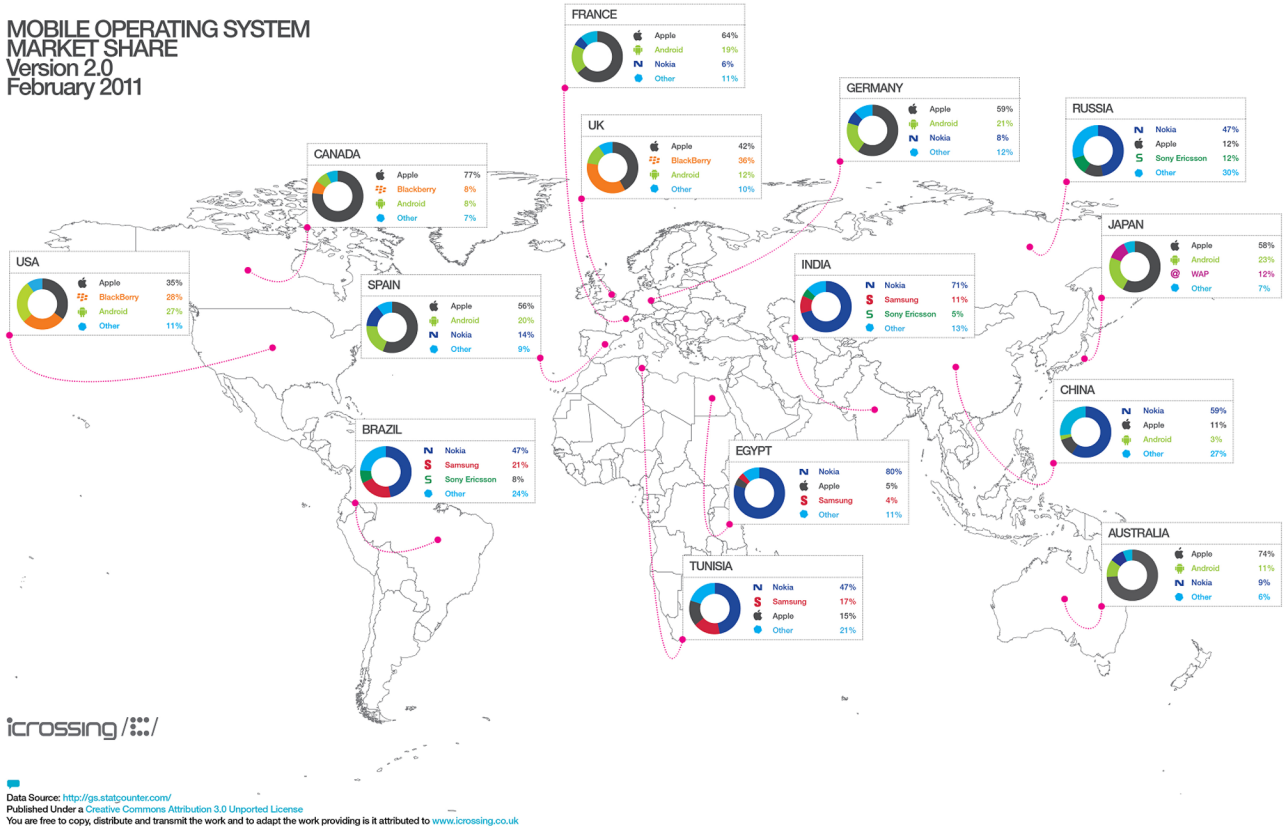
3. Que comprendáis que los entornos no funcionan de forma aislada, sino que coexisten.

4. Que seáis capaces de elegir el entorno de programación idóneo según los requerimientos de la aplicación móvil que pretendáis desarrollar.

1. Historia y evolución de los entornos de programación móviles

La industria de los dispositivos y las aplicaciones móviles es un entorno en constante cambio. Durante el 2010 vimos cómo Nokia abandonó Symbia y quedó a la espera de que MeeGo y su asociación con Windows les relanzaran en el mundo de los *smartphones*. También asistimos al espectacular crecimiento de Android, que ha pasado por delante del iOS de Apple y de BlackBerry y se ha convertido en la segunda plataforma (por detrás de Symbian).

En lo que respecta al mercado de los fabricantes de equipos originales vimos más movimientos en el 2010 que en los diez años anteriores. Apple y RIM adelantaron a algunos de los fabricantes tradicionales (Sony Ericsson, Motorola, LG) y reclamaron un puesto en el *top* 5. Según algunas estimaciones, ZTE podría unirse a ellos pronto. La siguiente figura ofrece una visión general sobre cómo está el mercado de las plataformas móviles en todo el mundo. Por países, destaca el dominio de Apple en Estados Unidos y en diversos países de Europa (como España), así como la cada vez más destacada presencia de Android. Nokia arrasa en India, en China y en otras potencias emergentes.

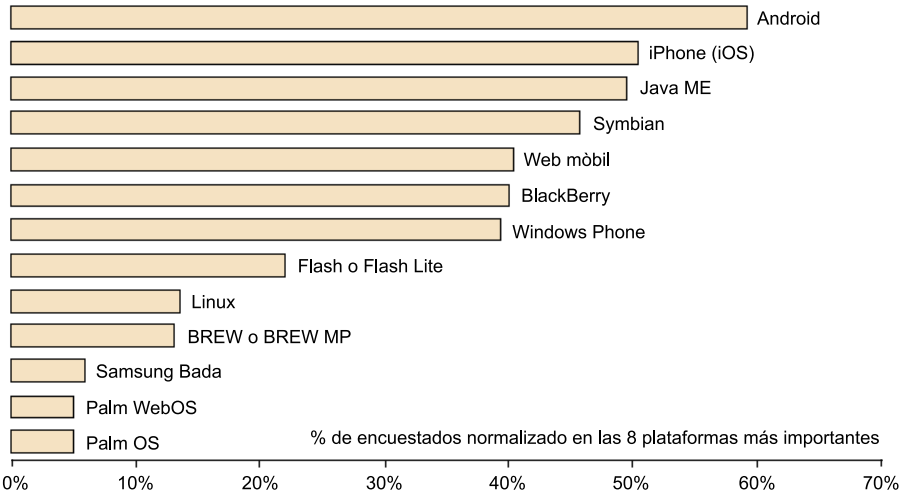


Cuotas de mercado de las diferentes plataformas móviles

La lucha por la supremacía de las plataformas móviles está candente. Android y iPhone, así como BlackBerry o Nokia, son varios de los actores que más destacan.

Por lo tanto, las principales medidas para el desarrollo de aplicaciones móviles han cambiado mucho y lo seguirán haciendo. Una de ellas es la popularidad de las diferentes plataformas entre los desarrolladores. En los últimos tiempos se ha producido una migración en las preferencias de los desarrolladores, que los ha movido desde la "vieja guardia" (Symbian, BlackBerry y Java) hacia los nuevos reyes del sector: iOS y Android. Según algunos estudios, cerca del 60% de los desarrolladores han desarrollado aplicaciones para Android, tal como se puede ver en la figura siguiente. El iOS de Apple ocupa el segundo lugar (con más del 50%), seguido por Java ME, que se encuentra en tercera posición. Así, podemos ver cómo los desarrolladores cambian el foco de su atención hacia unas plataformas y abandonan otras.

Plataformas más usadas por los desarrolladores de aplicaciones móviles en el 2010



Fuente: Mobile Developer Economics 2010 and Beyond. Producido por VisionMobile. Patrocinado por Telefónica Developer Communities. Junio 2010. Licenciado bajo licencia Creative Commons Attribution 3.0. Cualquier uso o remezcla de este trabajo debe conservar esta nota.

Sin lugar a dudas, el cambio más significativo en los últimos tiempos es que la distancia entre Android y iOS, por un lado, y el resto de plataformas, por otro, se está haciendo cada vez mayor. La *app store* de Apple contiene más de trescientas mil aplicaciones, mientras que estimaciones recientes sitúan el número de aplicaciones del Market de Android en unas ciento treinta mil.

Mientras tanto, Nokia ha estado poniendo un esfuerzo considerable en la Ovi Store y, de hecho, ha incrementado su popularidad entre consumidores y desarrolladores, aunque aún le queda un largo camino para alcanzar a los dos gigantes dispensadores de aplicaciones.

Los motivos por los que los desarrolladores se mueven hacia iOS y Android pueden ser varios, pero los más importantes son los que exponemos a continuación. Por un lado, Apple ofrece una plataforma que es relativamente fácil de aprender y de usar, con la que el desarrollador puede diseñar UI[1] muy bue-

(1)UI (*user interface*)

nas. Además, tiene la tienda de aplicaciones más grande y, aunque el problema de la certificación es un inconveniente para algunos, no existen los problemas de portabilidad y fragmentación. Android, por otro lado, ha ido ganando ímpetu en todos los campos asaltando los mercados clave de sus competidores. Por supuesto, tiene muchos inconvenientes derivados de la fragmentación, pero estos se pasan por alto muchas veces debido a la dependencia de muchos fabricantes de esta plataforma.

Otro aspecto importante a comentar es la disparidad que ha habido entre las ventas de dispositivos para cada plataforma y el número de aplicaciones disponibles. Sería de esperar que las plataformas con la mayor penetración en el mercado fueran las que lo dominaran (en términos de aplicaciones), pero nada más lejos de la realidad, por lo menos hasta el 2011.

Si tomamos como referencia el tercer cuatrimestre del año 2010, podéis ver que las dos plataformas con la penetración más baja, iOS y Android, tenían el mayor número de aplicaciones disponibles.

En el lado opuesto, mientras Java ME y Flash Lite han tenido, con diferencia, la mayor penetración en el mercado, apenas se pueden comparar con las nuevas plataformas en cuanto a volúmenes de aplicaciones.

En el cuarto cuatrimestre, este contraste se hizo incluso mayor. Tanto la tienda de Android como la de iOS crecieron prácticamente en cien mil aplicaciones cada una. Windows Phone mostró un crecimiento digno de mencionar (alcanzó cuatro mil aplicaciones en apenas dos meses), aunque le queda un largo camino para convertirse en una amenaza seria para los dos actores principales.

Por lo tanto, vemos que cada vez hay más desarrolladores dedicados a este tipo de aplicaciones. No obstante, debemos señalar que las encuestas realizadas indican que la mayoría reconocen pocos beneficios económicos, mientras que solo un 5% tiene beneficios por encima de sus expectativas.

Si bien es cierto que asistimos a un *boom* de tiendas de aplicaciones, esto no es necesariamente una bendición para los desarrolladores. Muchos de ellos se enfrentan a problemas para que los potenciales usuarios descubran sus aplicaciones, las cuales se quedan enterradas bajo otras miles de aplicaciones. Podríamos decir que es como ir a una tienda de discos con doscientos mil CDs: solo se mira en el *top* 10.

En este sentido, una opción para los desarrolladores es adoptar una estrategia de **escaparate múltiple** mientras se adapta el modelo de beneficio a tiendas de aplicaciones específicas. Por ejemplo, hasta ahora ha sido muy difícil vender contenido en Android, por lo que parece que la mejor estrategia para esta plataforma, por lo menos para darse a conocer, es que la aplicación sea gratis.

Por lo tanto, la conclusión que podemos sacar de todo esto es que hay que estar muy atento para ver cómo evolucionan factores importantes como el desarrollo de aplicaciones, los beneficios, la distribución, la venta al por menor, la portabilidad y la fragmentación, entre otros.

2. Aplicaciones web y aplicaciones nativas

Antes de entrar a fondo en los entornos de programación de aplicaciones para dispositivos móviles, vamos a establecer de forma resumida el ámbito en el que nos vamos a mover. A continuación definiremos términos clave y compararemos las ventajas e inconvenientes de los dos paradigmas de desarrollo más comunes.

Para empezar, vamos a definir qué se entiende por aplicación web y por aplicación nativa. Vamos a considerar las ventajas e inconvenientes de cada una de ellas.

Una **aplicación web** es, básicamente, un sitio web específicamente optimizado para un dispositivo móvil. Las características que definen una aplicación web son las siguientes: la interfaz de usuario se construye con tecnologías web estándar, está disponible en una URL[2] (pública, privada o protegida por una contraseña) y está optimizada para los dispositivos móviles. Una aplicación web no está instalada en el dispositivo móvil.

(2)URL (*uniform resource locator)*: localizador uniforme de recursos

En el caso de la aplicación web, el sitio puede ser cualquiera, desde una web-anuncio de un pequeño negocio estándar a una calculadora de hipotecas o un controlador diario de calorías (el contenido es irrelevante).

Las **aplicaciones nativas**, por el contrario, están instaladas en el dispositivo móvil, tienen acceso al *hardware* (altavoces, acelerómetro, cámara, etc.) y están escritas en algún lenguaje de programación compilado (como, por ejemplo, el Objective-C).

Diferentes aplicaciones tienen diferentes requisitos. Algunas aplicaciones se adaptan mejor a las tecnologías web que otras. Conocer las ventajas e inconvenientes de cada paradigma os ayudará a decidir qué camino es el más apropiado para cada situación.

La principal ventaja del desarrollo de aplicaciones nativas es que se puede acceder a todas las características *hardware* del dispositivo.

A continuación enumeramos los inconvenientes del desarrollo de aplicaciones nativas:

- La aplicación solo funcionará en la plataforma escogida.
- Hay que desarrollarla usando el lenguaje de programación establecido para la plataforma.
- Es más complicado distribuir parches o actualizaciones que solucionen errores.
- El ciclo de desarrollo es más lento.

Objective-C

Para implementar una aplicación nativa para iPhone hay que programar en Objective-C

Las ventajas del desarrollo de aplicaciones web son las siguientes:

- Los desarrolladores web pueden usar sus propias herramientas.
- Se pueden usar los conocimientos de diseño y desarrollo web que ya se tengan.
- La aplicación funcionará en cualquier dispositivo que tenga un navegador web.
- Se pueden solucionar errores en tiempo real.
- El ciclo de desarrollo es más rápido.

Los inconvenientes del desarrollo de aplicaciones web son los siguientes:

- No se puede acceder a todas las características del dispositivo móvil.
- Puede ser difícil conseguir efectos sofisticados en la interfaz de usuario.

Qué aproximación es la mejor en cada caso es un debate interesante. La naturaleza de los dispositivos móviles, que cada vez más están permanentemente conectados, hace que la línea entre aplicaciones web y aplicaciones nativas se difumine. Incluso hay varios proyectos (entre los que PhoneGap es el más notable) que desarrollan soluciones que permiten a los desarrolladores web coger una aplicación web y empaquetarla como una aplicación nativa, ya sea para iPhone o para otra plataforma móvil.

3. Enumeración de los diferentes entornos

Al igual que el sistema operativo de un ordenador, un sistema operativo móvil es la plataforma *software* que determina las funciones y las características disponibles en el dispositivo, como el control de los teclados, la seguridad inalámbrica, la sincronización con aplicaciones, el correo electrónico, los mensajes de texto, etc. El sistema operativo móvil determina también qué aplicaciones de terceras partes se pueden instalar en el dispositivo. Por lo tanto, cada sistema operativo define unos entornos sobre los que podemos crear aplicaciones. En este apartado vamos a hacer un repaso de los más importantes.

Firmware

El sistema operativo de un dispositivo se conoce en inglés como *firmware*.

3.1. Entornos para dispositivos de diferentes vendedores

En este subapartado estudiaremos las plataformas *software* que pueden funcionar en diferentes plataformas *hardware* de diferentes fabricantes. En concreto, explicaremos la historia y las características principales de las siguientes:

- Java ME
- Symbian
- Android
- Windows Mobile
- Qt framework
- BREW.
- Palm OS

3.1.1. Java ME

En 1999, Sun desarrolló una versión de Java especialmente diseñada para dispositivos móviles, Java 2 Micro Edition, basada en una máquina virtual llamada KVM. Esta primera versión solo contenía una única máquina virtual y un único API (inicialmente diseñados para Palm OS), hecho que puso de manifiesto la insuficiencia de esta solución para la gran variedad de dispositivos diferentes que existían. De esta forma, en el 2000 nació la primera versión de una configuración, el *connected limited device configuration* (J2ME CLDC 1.0). Una configuración ofrece el API básico para programar dispositivos, aunque no aporta todas las clases necesarias para desarrollar una aplicación completa. Por lo tanto, la primera configuración no tenía las herramientas necesarias para permitir a los desarrolladores escribir programas para el dispositivo Palm. En julio del 2000 nació la primera implementación de un perfil, concretamente el llamado *mobile information device profile* (MIDP), que no estaba destinado a PDA, sino a teléfonos móviles y a paginadores. A partir de este primer per-

fil, J2ME fue ampliamente aceptado por la comunidad de desarrolladores de dispositivos móviles y se ha ido expandiendo a una gran velocidad hasta la actualidad.

Java ME[3] (anteriormente conocida como J2ME[4]) es, por lo tanto, una edición de Java orientada a dispositivos pequeños. Es una versión recortada del Java SE con ciertas extensiones enfocadas a las necesidades particulares de este tipo de dispositivos. Esta tecnología consiste en una máquina virtual y en un conjunto de API[5] adecuados para estos dispositivos.

(3)Java ME (Java Micro Edition)

(4)J2ME (Java 2 Platform, Micro Edition)

(5)API (*application programming interface*)

Esta plataforma produce normalmente aplicaciones portables, aunque algunas veces existen librerías específicas de cada dispositivo (comúnmente usadas para juegos), que las hacen no portables. A pesar de ello, Java ME se ha convertido en una buena opción para crear aplicaciones para teléfonos móviles, ya que se puede emular en un PC durante la fase de desarrollo y luego se pueden cargar fácilmente las aplicaciones en el móvil. Aunque el proceso no sea directo, resulta bastante económico portarlas a otros dispositivos al utilizar tecnologías Java para el desarrollo.

Se usa muchas veces para proporcionar aplicaciones simples en teléfonos móviles de gama baja. Por lo tanto, las aplicaciones (incluyendo sus datos) no pueden ocupar demasiada memoria si se tienen que ejecutar en la mayoría de estos teléfonos. Además, tienen que estar firmadas criptográficamente para poder usar APIs como la de acceso al sistema de ficheros. Esto es relativamente caro y raramente se hace, incluso para aplicaciones comerciales. Java ME se ejecuta sobre una máquina virtual que permite un acceso razonable, pero no completo, a las funcionalidades del dispositivo sobre el que se ejecuta la aplicación. El proceso JSR[6] sirve para incrementar gradualmente la funcionalidad disponible para JavaME, mientras proporciona a los operadores y a los fabricantes la capacidad de prevenir o limitar el acceso al *software* disponible.

(6)JSR (*Java specification requests*)

3.1.2. Symbian

La historia de Symbian comienza en el año 1981. En la siguiente cronología podéis ver la evolución del sistema operativo Symbian:

- **1981**. Psion lanza su primer producto, Flight simulator.
- **1984**. Psion Organiser ve la luz.
- **1990**. SIBO SO (16 bits).
- **1997**. EPOC SO (32 bits).
- **1998**. El nombre de Symbian aparece por primera vez.

- **1999**. EPOC versión 5.
- **2000**. Symbian 6.0.
- **2001**. Symbian 6.1.
- **2003**. Symbian 7.0.
- **2004**. Symbian 8.0.
- **2005**. Symbian 9.0.
- **2008**. Nokia compra Symbian Ltd., la empresa que hay detrás de Symbian OS.
- **2009**. Creación de la *Symbian Foundation*.
- **2010**. Se publica el código fuente de Symbian bajo licencia EPL[7].
- **2011**. Nokia realiza una importante alianza con Microsoft y deja de lado el sistema operativo Symbian, que sería reemplazado por el Windows Phone 7.

(7)EPL (Eclipse Public License)

Symbian es un sistema operativo fruto de la alianza de varias empresas de telefonía móvil, entre las que se encuentran Psion, Nokia, Ericsson y Motorola, con el que se pretendía desarrollar y estandarizar un sistema operativo que permitiera a teléfonos móviles de diferentes fabricantes intercambiar información.

El Symbian OS fue durante unos años el sistema operativo estándar para los *smartphones* de la época, ya que más del ochenta y cinco por ciento de los fabricantes de estos dispositivos tenían licencias para usarlo. El Symbian OS estaba diseñado para los requisitos específicos de los teléfonos móviles 2.5G y 3G.

Diseñada desde el inicio para dispositivos móviles, la plataforma Symbian es un sistema operativo de tiempo real, multitarea, específicamente pensada para funcionar bien en sistemas con recursos limitados, así como para maximizar la eficiencia y la vida de la batería y minimizar, de esta manera, el uso de memoria. La Symbian Foundation mantiene el código para la plataforma de *software* libre basada en Symbian OS y las aportaciones de *software* de Nokia, NTT DOCOMO y Sony Ericsson, e incluye las interfaces de usuario S60 y MOAP(S). La plataforma es de código abierto en su totalidad, y la mayoría se proporciona bajo la Licencia Pública de Eclipse.

Popularidad de Symbian OS

Se han vendido más de trescientos millones de unidades basadas en Symbian OS. Durante años, Symbian OS ha gozado de más del 50% de cuota de mercado.

El sistema operativo Symbian incorporó el soporte a pantallas táctiles gracias a UIQ[8]. UIQ es una interfaz de usuario gráfica basada en el uso de un lápiz, que se puede encontrar en teléfonos 2.5G y 3G de las siguientes marcas: Motorola, Sony Ericsson, BenQ y ARIMA. Los teléfonos UIQ utilizan pantallas táctiles con una resolución de 208 a 240 x 320 píxeles y una profundidad de color de 12, 16, 18 o 24 bits, dependiendo de la versión de UIQ o del terminal. Las últimas versiones de UIQ fueron las 3.x.

(8)UIQ (*user interface Quartz*): interfaz de usuario Quartz

3.1.3. Android

Android es una plataforma basada en Linux de la Open Handset Alliance, entre cuyos treinta y cuatro miembros se encuentran Google, HTC, Motorola, Qualcomm y T-Mobile. Por lo tanto, treinta y cuatro de las principales compañías de *software*, *hardware* y telecomunicaciones dan soporte a esta plataforma. El *kernel* de Linux se usa como HAL[9]. La programación de aplicaciones se hace básicamente en Java. Es necesario el SDK[10] específico de Android para desarrollar, aunque se puede usar cualquier IDE[11] Java. El código que sea crítico en cuanto a rendimiento se puede escribir en C, C++ u otros lenguajes de código nativo usando el NDK[12] de Android.

(9)HAL (*hardware abstraction layer*)

(10)SDK (*software development kit*)

(11)IDE (*integrated development environment*)

(12)NDK (*native development kit*)

3.1.4. Windows Mobile

La plataforma Windows Mobile estuvo disponible en una gran variedad de dispositivos de diferentes operadores inalámbricos. Se podía encontrar el *software* Windows Mobile en productos de Dell, HP, Motorola, Palm e i-mate. Los dispositivos con Windows Mobile estaban disponibles para redes GSM o CDMA.

Windows Mobile es una variante de Windows CE para teléfonos móviles. Originalmente, Windows CE se desarrolló para ordenadores de bolsillo y PDA con pantallas táctiles que funcionaban con un *stylus*, y se adaptó posteriormente para su uso en *smartphones* equipados con un teclado. Por lo tanto, los teléfonos se convirtieron en la mayor base de dispositivos instalados con CE, aunque la cuota de mercado ha caído dramáticamente desde la aparición de Android y iPhone. Windows Mobile soporta un subconjunto de la interfaz de programación de Win32 y una GUI[13] simplificada con una ventana en la pantalla a la vez. Las aplicaciones se pueden usar en .NET Compact Framework. Windows Mobile 6.5 introdujo las interfaces estilo iPhone basadas en el contacto con los dedos, mientras que Windows Phone 7 es un rediseño sustancial que usa Silverlight y XNA para interfaces de usuario más ricas.

(13)GUI (*graphical user interface)*

3.1.5. Qt framework

Qt usa estándar C++, pero hace un uso extensivo de un pre-procesador especial llamado MOC[14] para enriquecer el lenguaje. También se puede usar Qt en otros lenguajes de programación utilizando enlaces entre lenguajes. Funciona sobre las principales plataformas y tiene un soporte internacional extenso. Entre las

(14)MOC (*meta object compiler*)

características no relacionadas con la GUI, se encuentra el acceso a bases de datos SQL, el tratamiento de XML, la gestión de *threads*, el soporte de red y una API multiplataforma unificada para la gestión de ficheros.

3.1.6. BREW

BREW se usa para aplicaciones en dispositivos CDMA, aunque también soporta modelos GPRS/GSM. Las aplicaciones se distribuyen mediante una plataforma de contenido BREW y han tenido poca penetración en Europa. BREW puede proporcionar control completo del dispositivo y acceder a toda su funcionalidad. No obstante, el potencial que proporciona el código nativo con acceso directo a las APIs del dispositivo ha provocado que el proceso de desarrollo en BREW haya tenido que ser adaptado, en gran medida, para todos los vendedores de *software* reconocidos. Mientras que el SDK de BREW está disponible de forma libre, ejecutar *software* en *hardware* real de un dispositivo móvil (al contrario que el emulador proporcionado) requiere una firma digital que se pueda generar con herramientas publicadas por varios participantes, esencialmente proveedores de contenido para móviles y Qualcomm. Incluso entonces, el *software* solo funcionará en dispositivos habilitados para test. Para que se pueda descargar en teléfonos normales, el *software* tiene que ser comprobado, probado y recibir la aprobación de Qualcomm mediante su programa de testeo TRUE BREW.

3.1.7. Palm OS

Desde la aparición del primer Palm Pilot (en 1996), la plataforma Palm OS ha proporcionado a sus dispositivos móviles herramientas de negocio esenciales, así como la capacidad de acceder a Internet o a una base de datos central corporativa mediante una conexión inalámbrica.

El Palm OS tuvo una gran aceptación empresarial en el importante mercado de EE. UU. basada en las Palm PDA.

Palm webOS es el sistema operativo móvil propietario (evolución de Palm). Funciona sobre un *kernel* Linux que soporta multitarea. Se lanzó con Palm Pre y Pixi. Ahora es propiedad de Hewlett Packard.

3.1.8. Flash lite

Se usa en dispositivos que soportan el reproductor Flash lite.

3.1.9. *Microbrowser*

Los entornos basados en el concepto *microbrowser* proporcionan una funcionalidad limitada mediante una interfaz web.

3.2. Desarrollo multiplataforma

En este subapartado vamos a describir *frameworks* que permiten desarrollar aplicaciones que funcionen tanto en iPhone OS como en Android.

3.2.1. Titanium Mobile

Titanium es un *framework* de código abierto que permite desarrollo multiplataforma. Se puede desarrollar una aplicación que funcione en dispositivos móviles (iOS, Android, RIM) o plataformas de escritorios (OSX, Windows).

Todo el código fuente de la aplicación se escribe en Javascript, CSS y HTML5. Esto es positivo, ya que no es necesario aprender lenguajes complejos como Objective-C o C++.

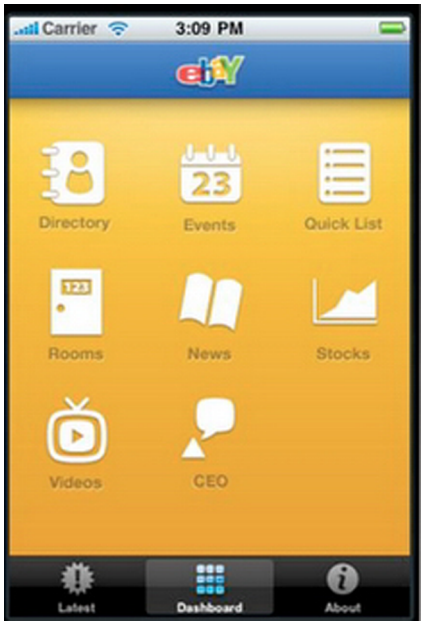
Titanium es extensible, se puede extender el *framework* añadiendo módulos propios en Objective-C o en Java para el caso de Android.

Nota

Existe una gran cantidad de documentación para Titanium.

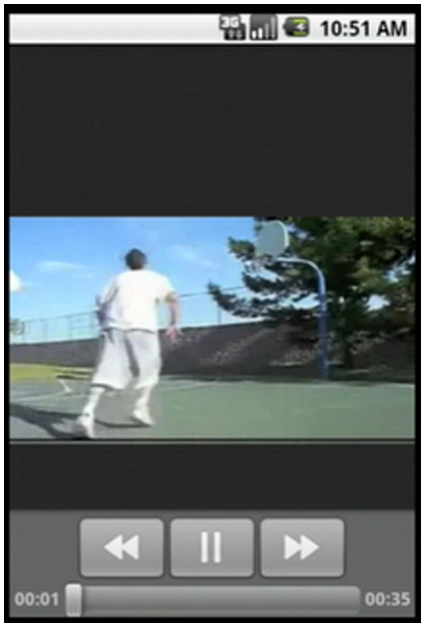
Con Titanium, un desarrollador se puede beneficiar del uso de:

1) Interfaces nativas:



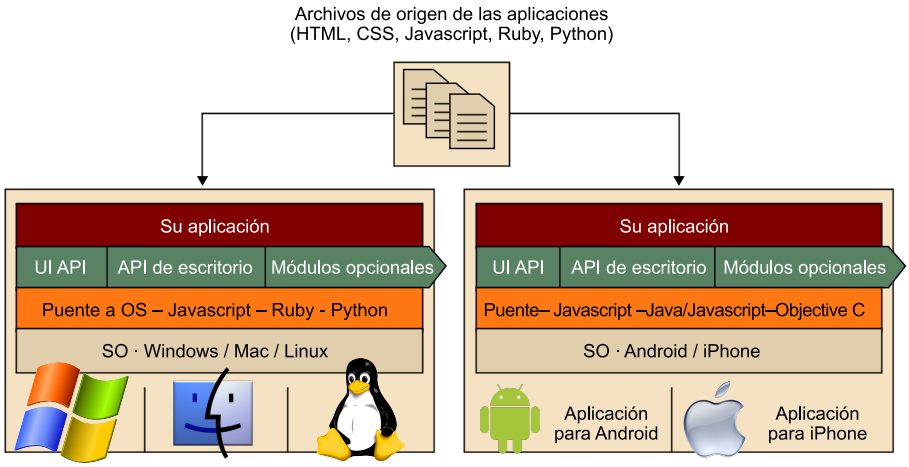
Interfaz nativa

2) Aplicaciones multimedia:



Aplicación multimedia

3) Entorno móvil y de escritorio:



Titanium permite desarrollar tanto aplicaciones móviles como de escritorio

4) Lenguaje Javascript:

> Todo esto es posible gracias a que Titanium tiene un puente que traduce el código Javascript al código equivalente Objective-C o Java en tiempo de ejecución.

3.2.2. PhoneGap

PhoneGap permite desarrollar aplicaciones para Android mediante tecnologías web como HTML, CSS y JavaScript, y puede convertir esas aplicaciones web en aplicaciones nativas Android. De hecho, PhoneGap soporta múltiples

(15)MIT (Massachusetts Institute of Technology)

plataformas (como Android, iPhone, Palm, Windows Mobile y Symbian), así que se puede usar el mismo código fuente para crear aplicaciones para múltiples plataformas. Pese a que se venden como "herramientas de tecnología web", lo que ofrecen PhoneGap u otros *frameworks* similares como Titanium es acceso al *hardware* de la máquina (se pueden hacer aplicaciones en HTML y JavaScript que usen la cámara, la brújula o el acelerómetro). PhoneGap es, además, libre bajo licencia MIT[15].

jQuery i jQTouch

PhoneGap permite desarrollar aplicaciones con librerías de JavaScript como jQuery/jQTouch

Por lo tanto, PhoneGap es una solución de código abierto diseñada para dar acceso JavaScript a los desarrolladores web a características populares de los dispositivos móviles como la cámara, el GPS, el acelerómetro o las bases de datos SQLite locales sin necesidad de tener que escribir aplicaciones completas. El objetivo es hacer más fácil el desarrollo de aplicaciones móviles.

Para conseguir esto, el *framework* PhoneGap actúa como un puente entre las aplicaciones web y los dispositivos móviles. Permite a los desarrolladores envolver aplicaciones web dentro de una aplicación nativa, lo que hace el desarrollo más fácil para aquellos que no están familiarizados con Objective-C y Cocoa.

3.3. Entornos para dispositivos de un vendedor único

Las siguientes plataformas *software* solo funcionan en plataformas *hardware* de un fabricante específico:

1) BlackBerry. Blackberry tiene soporte para correo electrónico, teléfono móvil, mensajes de texto, envío de faxes, navegación por Internet y otros servicios de información inalámbricos, así como una interfaz táctil. Los dispositivos Blackberry disponen de serie de un teclado QWERTY optimizado para utilizarlo tecleando con los pulgares. Cuando aparecieron, los dispositivos Blackberry cogieron pronto una posición dominante en el mercado norteamericano de los *smartphones*. Para Blackberry son importantes el BES[16] y el BlackBerry MDS[17].

(16)BES (*BlackBerry enterprise server*)

(17)MDS (*mobile data system*)

2) iOS de Apple. El SDK para iPhone y iPod usa Objective-C, que está basado en el lenguaje de programación C. En su momento, solo estaba disponible en Mac OS 10.5+ y era la única forma de escribir una aplicación para iPhone. Además, Apple tiene que verificar todas las aplicaciones antes de que se puedan alojar en el *app store*, el único canal de distribución para las aplicaciones para iPhone y iPod *touch*. No obstante, se pueden lanzar aplicaciones para iPhones pirateados no aprobadas por Apple mediante el instalador Cydia. Este sistema se usa también para el *tablet* iPad.

4. Lenguajes de programación

Como ya hemos visto, hay dos clases principales de aplicaciones para dispositivos móviles: las aplicaciones nativas y las aplicaciones web. Un tercer caso de paradigma sería el marcado por Java. Para este caso, muchos de los nuevos móviles soportan alguna versión de MIDP[18], y el desarrollo en este entorno es bastante sencillo. La instalación, no obstante, es algo más complicado. En general, se instalan aplicaciones mediante enlaces en Internet, pero algunos operadores o fabricantes ponen límites a las aplicaciones que se pueden instalar en el móvil.

[18]MIDP (*mobile information device profile*)

En cuanto a las aplicaciones nativas, depende del sistema operativo del móvil. Para muchos, desarrollar aplicaciones nativas puede costar dinero (para herramientas y SDKs), y también hay problemas en lo que respecta a la manera en que se distribuyen las aplicaciones. La instalación y la depuración de errores varían en función del sistema operativo.

Por lo tanto, el lenguaje de programación que se use vendrá probablemente dictado por el dispositivo y la plataforma para la que se desea desarrollar una aplicación, además de por la aplicación que se desea crear.

A continuación enumeramos los diferentes lenguajes con los que se pueden desarrollar aplicaciones nativas para diferentes plataformas:

- Si se quiere hacer una aplicación para iPhone o iPod touch, se usará **Objective-C**.
- Si se quiere hacer una aplicación para Android, se usará **Java**.
- Si se quiere hacer una aplicación para BlackBerry, se usará **Java Micro Edition**.
- Si se quiere hacer una aplicación para Symbian OS, se puede usar **C++**, **Java** o **.NET *Compact Framework***.
- Si se quiere hacer una aplicación parar Windows Mobile, las opciones son **Visual C++**, **Visual C#**, **Visual Basic**, **JScript** y **ASP.NET**.

Así, la plataforma o el dispositivo dictarán qué lenguajes de programación se pueden utilizar. Si se quiere desarrollar para una plataforma que permite tanto C++ como Java, entonces el tipo de aplicación que se planea desarrollar podría dictar qué lenguaje es la mejor opción.

A continuación se muestra una compilación de los lenguajes de programación más populares para dispositivos móviles.

4.1. Lenguajes de programación para Windows Mobile

Se puede escoger entre varias opciones de lenguajes de programación a la hora de desarrollar aplicaciones para dispositivos con Windows Mobile. En este subapartado describiremos brevemente cada una de estas opciones.

4.1.1. Visual C++

Se conoce a C++ como un lenguaje de desarrollo nativo, debido a que interactúa directamente con el *hardware* de un dispositivo Windows Mobile sin que intervenga ninguna otra capa (al contrario que Visual C#, por ejemplo). Programar usando C++ puede ser un desafío, ya que no es un lenguaje fácil de aprender. Algunos errores en un programa C++ pueden, potencialmente, bloquear todo el dispositivo.

Errores en un programa C ++

Acceder a memoria que ha sido liberada u olvidarse de liberar memoria en un programa C++ puede bloquear un dispositivo Windows Mobile

Las ventajas de usar Visual C++ son las siguientes: la velocidad de ejecución, el tamaño de la aplicación y la flexibilidad. Las aplicaciones escritas en C++ se ejecutan muy rápido y consumen los recursos mínimos.

Juegos de acción

Los juegos de acción en tiempo real son buenos ejemplos de programas que se benefician de la velocidad de ejecución de C++

Lectura recomendada

Una buena forma de aprender Visual C++ es investigar la *Visual C++ Express Edition* de Visual Studio (gratuita), ver el vídeo de entrenamiento, los *webcasts*, y leer la documentación. Aunque la *Express Edition* de Visual Studio no permite desarrollar aplicaciones para Windows Mobile, casi todo lo que aprendáis sobre desarrollo de aplicaciones se puede aplicar directamente a dispositivos móviles.

Las aplicaciones Visual C++ pueden interaccionar con el dispositivo Windows Mobile llamando a las APIs Win32. Estas APIs son funciones que realizan acciones concretas, como hacer que se oiga un sonido o dibujar un botón en la pantalla. Hay miles de APIs (Windows Mobile soporta un subconjunto del conjunto completo de APIs Win32 para escritorio) y están documentadas en la sección titulada *Windows mobile features (native)* del MSDN[19] de Microsoft. Cuando se navega por esta sección, hay que tener cuidado con el hecho de que algunas APIs solo están disponibles para Windows Embedded CE, una plataforma que está relacionada (pero separada) de Windows Mobile.

[19]MSDN (*Microsoft developer network*)

Si se tiene experiencia desarrollando para Windows usando Visual C++, la transición a Windows Mobile no es especialmente complicada. Sería necesario aprender a instalar y usar las herramientas específicas y luego aprender a utilizar las características específicas del dispositivo, lo que permitirá explotar las capacidades de los dispositivos.

Para empezar una aplicación Visual C++, hay que arrancar Visual Studio, seleccionar Archivo > Nuevo > Proyecto y seleccionar Smart device en el nodo Visual C++.

Si se es nuevo tanto programando como con Windows Mobile, sería una buena idea empezar con Visual C# y hacer la transición a Visual C++ entonces.

4.1.2. Visual C# y Visual Basic

Visual C# y Visual Basic .Net son lenguajes de desarrollo más sencillos que Visual C++. No solo son relativamente fáciles de aprender, sino que además tienen soporte para el .NET Compact Framework.

.NET Compact Framework

.NET Compact Framework es una librería de clases que realizan gran cantidad de tareas usadas frecuentemente en programación para simplificar el desarrollo de aplicaciones

Las herramientas de desarrollo para C# y Visual Basic .NET incluyen un diseñador completo de interfaz de usuario, WYSIWYG[20]. Podéis arrastrar y colocar botones y otros controles directamente en la ventana de la aplicación, y entonces hacer doble clic para acceder al código que hay debajo. Este sistema hace que crear interfaces de usuario para las aplicaciones sea extremadamente rápido y fácil.

[20]WYSIWYG (*what you see is what you get*)

Como parte de la librería Compact Framework, hay disponibles clases extra que cubren desde las estructuras de datos hasta la intercepción de mensajes de texto. Para hacer uso de las características específicas de Windows Mobile, se proporcionan un conjunto de clases extra. Estas clases proporcionan acceso a las características del dispositivo como, por ejemplo, la lista de contactos o la cámara.

Si se tiene experiencia desarrollando aplicaciones para Windows usando Visual C#, la transición debería ser relativamente sencilla. El Compact Framework es un subconjunto del .NET Framework, por lo que el código puede necesitar de ligeras modificaciones para algunas funcionalidades.

Visual C# es una buena forma de aprender programación. Para aprender todo lo necesario tanto de Visual C# como de Visual Basic, podéis acudir al MSDN de Microsoft.

Para empezar una aplicación Visual C# o Visual Basic .NET, hay que arrancar Visual Studio, seleccionar Archivo > Nuevo > Proyecto y seleccionar Smart device en el nodo correspondiente al lenguaje que nos interese.

4.1.3. JScript

El navegador web incluido en los dispositivos Windows Mobile (Internet Explorer Mobile) soporta JScript. JScript es un superconjunto del lenguaje conocido como JavaScript. Los programas JScript son ficheros de texto plano que ejecuta el navegador web. Pueden estar incrustados en una página HTML o almacenados en ficheros separados.

Una aplicación JScript se ejecuta dentro del navegador web y usa la ventana del navegador web para la entrada y salida de información. Es posible hacer uso de técnicas de programación AJAX[21] para proporcionar un grado de interacción con el usuario y comunicarse con un servidor remoto. Debido a la naturaleza de JScript, las aplicaciones no pueden acceder a datos locales que no sean simplemente *cookies*, lo que introduce algunas limitaciones.

[21] AJAX (*asynchronous JavaScript and XML*): JavaScript asíncrono y XML

Herramienta de desarrollo

No se necesita ninguna herramienta de desarrollo especial: un editor de texto es suficiente para crear una aplicación JScript. Podemos guardar el programa de forma local o acceder a él desde un servidor web.

4.1.4. ASP.NET

Mientras que JScript es una solución del lado del cliente para programas aplicaciones de Internet, ASP.NET es una solución del lado del servidor. Con ASP.NET se pueden escribir aplicaciones en C# o Visual Basic .NET que residan en un servidor web y realicen tareas complejas, como crear controles de interfaz de usuario y acceder a bases de datos. ASP.NET aísla las características del dispositivo de la aplicación y facilita la ejecución de una aplicación en varios tipos de dispositivos diferentes.

5. Ejemplos de entornos

En este apartado describiremos dos entornos de forma más detallada: iPhone / iOS y Android.

5.1. iPhone / iOS

iOS comprende el sistema operativo y las tecnologías que se usan para ejecutar aplicaciones de forma nativa en dispositivos como iPad, iPhone y iPod touch. Aunque comparte una herencia común y muchas tecnologías de base con el Mac OS X, iOS se diseñó para satisfacer las necesidades de un entorno móvil, donde las necesidades de los usuarios son ligeramente diferentes. Si se han desarrollado previamente aplicaciones para Mac OS X, se encontrarán muchas tecnologías familiares, pero también tecnologías que solo están disponibles en iOS, como el soporte a interfaz táctil o al acelerómetro.

El SDK de iOS contiene el código, la información y las herramientas necesarias para desarrollar, probar, ejecutar, depurar errores y adaptar aplicaciones para iOS. Las herramientas Xcode proporcionan el entorno básico para editar, compilar y depurar errores en el código. Xcode también proporciona el punto de lanzamiento para probar las aplicaciones en un dispositivo iOS y en un simulador iOS (una plataforma que imita el entorno básico iOS, pero se ejecuta en un ordenador Macintosh local).

Este subapartado proporciona una descripción detallada de las características básicas que se pueden encontrar en iOS.

5.1.1. Visión general del sistema iOS

iOS (conocido como iPhone OS antes del 2010) es el sistema operativo para dispositivos móvil de Apple. Originalmente desarrollado para el iPhone, se ha ido extendiendo para dar soporte a otros dispositivos Apple (como el iPod touch, el iPad y Apple TV). Apple no da licencias para la instalación de iOS en *hardware* de terceras partes. En enero del 2011, el *app store* de Apple contenía más de trescientas mil aplicaciones iOS, que se habían descargado colectivamente más de diez billones de veces. En el último trimestre del 2010, tenía el 16% de cuota del mercado de los sistemas operativos para *smartphones* (en unidades vendidas), y era el tercero por detrás del Android de Google y Symbian. En el 2010 se llevó el 59% del consumo web móvil (sin incluir el iPad) en Norteamérica.

La interfaz de usuario del iOS se basa en el concepto de manipulación directa mediante la utilización de gestos multicontacto. Los elementos de control de la interfaz consisten en deslizadores, interruptores y botones. La respuesta a las peticiones del usuario es inmediata y proporciona una interfaz fluida. La interacción con iOS incluye gestos como "tocar fuerte", "tocar de forma más débil", "sujetar" y "soltar", que tienen definiciones específicas en el contexto del sistema operativo iOS y su interfaz multicontacto.

Algunas aplicaciones usan los acelerómetros internos para responder cuando se sacude el dispositivo (un resultado común es el comando *deshacer*) o rotar en tres dimensiones (un resultado común es cambiar de orientación vertical a horizontal y viceversa).

iOS está derivado del Mac OS X, con el que comparte la fundación Darwin y es, por lo tanto, un sistema operativo parecido a Unix (por naturaleza).

En iOS hay cuatro capas de abstracción:

- la capa Core OS
- la capa Core Services
- la capa Media
- la capa Cocoa Touch

Espacop d'iOS

El sistema operativo iOS usa escasos quinientos megabytes de la capacidad de almacenamiento del dispositivo, dependiendo del modelo.

5.1.2. Historia del sistema iOS

El sistema operativo apareció con el iPhone en el *Macworld Conference & Expo* en enero del 2007 y fue lanzado en junio de ese año. Al principio, los mensajes de marketing de Apple no especificaban un nombre diferente para el sistema operativo; simplemente decían que el "iPhone ejecuta OS X". Inicialmente, no soportaba aplicaciones de terceras partes. Steve Jobs argumentaba que los desarrolladores podían programar aplicaciones que "se comportarían como aplicaciones nativas en el iPhone". En octubre del 2007, Apple anunció que se estaba desarrollando un SDK nativo y que planeaban ponerlo "en las manos de los desarrolladores en febrero". En marzo del 2008, Apple lanzó la primera versión beta junto con un nuevo nombre para el sistema operativo: iPhone OS.

Las grandes ventas de los dispositivos móviles de Apple encendieron el interés en el SDK. El mes de septiembre anterior, Apple había lanzado el iPod touch, que tenía la mayoría de las capacidades del iPhone no relacionadas con la telefonía. Además, Apple vendió más de un millón de iPhones durante las vacaciones del 2007. En enero del 2010, Apple anunció el iPad, un dispositivo con una pantalla más grande que el iPhone y el iPod touch, diseñado para navegar por Internet, por contenidos multimedia y para lectura de iBooks.

En junio del 2010, Apple renombró iPhone Os como iOS. El nombre IOS había sido usado por Cisco durante una década para su IOS[22], usado en *routers* Cisco. Para evitar pleitos potenciales, Apple pagó la licencia para usar la marca IOS de Cisco.

(22)IOS (*internetwork operating system*): sistema operativo de interconexión de redes

5.1.3. Historia de las versiones del sistema iOS

La versión 4, anunciada en abril del 2010, presentaba multitarea, correo electrónico organizado en hilos y varias características orientadas a los negocios. En el WWDC[23] 2010, Apple anunció que iPhone OS se había renombrado a iOS. Apple pagó la licencia para la marca iOS a Cisco Systems (que posee IOS), la misma compañía con la que Apple había tenido una disputa sobre la marca iPhone. Apple lanzó el iOS el 4 en junio del 2010, tres días antes que el iPhone 4, para reducir la carga en los servidores de Apple. iOS 4 fue la primera versión del sistema operativo que era una actualización gratuita para el iPod touch; Apple cobraba 9,99$ para actualizaciones anteriores. Apple anunció previamente que los usuarios de iPad con *software* 3.x recibirían una actualización gratuita de la siguiente versión importante (4.x).

(23)WWDC (Conferencia Mundial de Desarrolladores de Apple)

iOS 4.0.1 incluía un arreglo para el indicador de la fuerza de la señal recibida. Se lanzó en julio de 2010, el día antes de que Apple realizara una conferencia de prensa para explicar su respuesta a los muy publicitados problemas de la antena del iPhone. Apple también lanzó iOS 3.2.1 para el iPad, que mejoraba la conectividad WiFi, la reproducción de vídeo y el copiar y pegar de archivos PDF[24] adjuntos del *tablet*, además de otras actualizaciones.

(24)PDF (*portable document format*)

En agosto del 2010 se lanzó el iOS 4.0.2 para iPhone y iPod touch, y el iOS 3.2.2 para el iPad, para arreglar algunas vulnerabilidades de seguridad.

En septiembre del 2010 se lanzó el iOS 4.1 para el iPhone y iPod touch; la actualización arreglaba algunos errores detectados por los usuarios, mejoraba la duración de la batería y añadía una nueva característica llamada Game Center, que permitía a los jugadores jugar partidas con otros jugadores, subir puntuaciones altas y desbloquear logros, y añadía soporte inicial para el iPod touch 4th Generation y la Apple TV 2G. iOS 4.1 también añadía fotografía HDR[25], una característica que solo el iPhone 4 era capaz de usar. El iOS 4.1 también añadía una nueva característica, llamada Ping, una herramienta de descubrimiento y red social de música.

(25)HDR (*high dinamic range*): alto rango dinámico

Game Center

Apple acabó quitando el Game Center del iPhone 3G debido a informes de bajo rendimiento.

En noviembre del 2010 se lanzó el iOS 4.2 para los desarrolladores. Nunca se lanzó al público, ya que se encontró un *bug* en la parte de WiFi en la edición limitada. Finalmente, lo que Apple hizo fue lanzar el iOS 4.2.1 al público.

El iOS 4.2.1 se lanzó en noviembre de 2010 con soporte para todos los dispositivos Apple A4, tercera y segunda generación de dispositivos, con la exclusión del Apple TV. Proporcionaba soporte inicial de iOS 4.x al iPad, además de AirPlay y AirPrint a todos los dispositivos compatibles. Además, contiene cambios menores en la aplicación YouTube y modifica la animación de multitarea.

El iOS 4.2.5 se lanzó como versión demo para la versión CDMA del iPhone 4. Esta variante iPhone 4 estaba disponible para los clientes de Verizon Wireless en USA. Esta versión tenía ligeros cambios específicos para la versión CDMA del móvil en la interfaz de usuario.

La versión beta del iOS 4.3 se lanzó a los desarrolladores en enero de 2011.

5.1.4. Características del sistema iOS

Siempre que se enciende el dispositivo o se presiona el botón *Home*, se presenta la pantalla principal con iconos de aplicaciones y un receptáculo en la parte inferior donde los usuarios pueden colocar las aplicaciones usadas con más frecuencia. La pantalla tiene una barra de estado a lo largo de la parte superior para mostrar datos como la hora, el nivel de batería y la potencia de señal. El resto de la pantalla se dedica a la aplicación actual.

Con el iOS 4 llegó la introducción de un sistema de carpetas simple. Se puede arrastrar cualquier aplicación y soltarla encima de otra para crear una carpeta. Una vez hecho esto, se pueden añadir otras diez aplicaciones a la carpeta mediante el mismo procedimiento (una carpeta puede gestionar hasta doce aplicaciones en iPhone y iPod touch y hasta veinte en iPad). Se selecciona un título para la carpeta de forma automática en función del tipo de aplicaciones que hay dentro, pero el usuario puede también editar el nombre.

La pantalla de inicio del iOS contiene aplicaciones por defecto. Algunas de estas aplicaciones no son visibles por defecto. El usuario puede acceder a ellas mediante la aplicación Settings o mediante otro método.

Activación de Nike+iPod

Nike+iPod se activa mediante la aplicación Settings, mientras que AirPrint se activa cuando el usuario imprime un fichero

Todas las utilidades (como notas de voz, calculadora y brújula) están en una carpeta llamada Utilidades en 4.0. Muchas de las aplicaciones incluidas están diseñadas para compartir datos.

Compartición de datos

En el sistema iOS se puede seleccionar un número de teléfono de un correo electrónico y guardarlo como un contacto o marcarlo para hacer una llamada telefónica

El iPod touch mantiene las mismas aplicaciones que están presentes, por defecto, en el iPhone, a excepción de las aplicaciones (anteriores a la cuarta generación) de teléfono, mensajes, brújula y cámara. La aplicación iPod (presente en el iPhone) se divide en dos aplicaciones en el iPod touch: música y vídeos. La fila inferior de aplicaciones se usa para delinear los propósitos principales del iPod touch: música, vídeos, Safari y *app store* (esta configuración se cambió

en la actualización 3.1). Para la cuarta generación de iPod touch, incluye FaceTime y cámara, y la configuración del receptáculo inferior cambia a música, *mail*, Safari, vídeo.

El iPad viene con las mismas aplicaciones que el iPod touch, excluyendo Stocks, Tiempo, Reloj, Calculadora, y la aplicación Nike+iPod. Se proporcionan aplicaciones de música y vídeo por separado, como en el iPod touch, aunque (como en el iPhone) la aplicación de música se llama iPod. La mayoría de las aplicaciones están completamente rehechas para beneficiarse de la pantalla más grande del iPad. La configuración por defecto del receptáculo inferior incluye Safari, *mail*, fotos y iPod.

Multitarea

Antes del iOS 4, la multitarea estaba limitada a una selección de las aplicaciones que Apple incluía en los dispositivos. A Apple le preocupaba que al ejecutar múltiples aplicaciones de terceras partes de forma simultánea se descargara la batería demasiado rápido. A partir del iOS 4, en dispositivos iOS de tercera generación en adelante, hay soporte para multitarea mediante siete APIs:

- audio en segundo plano
- voz sobre IP
- localización en segundo plano
- notificaciones *push*
- notificaciones locales
- finalización de tareas
- cambio rápido de aplicación

Presionar dos veces el botón *Home* activa el intercambiador de aplicación. Entonces aparece una interfaz deslizable desde la parte inferior. Si escogemos el icono correspondiente, se cambia a esa aplicación. A la izquierda hay iconos que funcionan como controles de música. El usuario también puede finalizar aplicaciones.

Game Center

Game Center es una red social de juego multijugador online lanzada por Apple. Permite a los usuarios "invitar a amigos a jugar a un juego, empezar un juego multijugador, controlar los logros y comparar las puntuaciones más altas en un tablón de líderes".

Game Center se anunció durante una presentación de iOS4 en abril del 2010. Se lanzó una versión previa para los desarrolladores registrados de Apple en agosto. Finalmente se lanzó en septiembre del 2010 con al iOS 4.1 en iPhone, iPhone 3GS y iPod touch de la segunda a la cuarta generación. Game Center

hizo su debut público en el iPad con el iOS 4.2.1. No hay soporte para el iPhone 3G y el iPhone original. No obstante, Game Center está disponible de forma no oficial para el iPhone 3G.

5.1.5. Desarrollo de aplicaciones para iOS

Las aplicaciones tienen que estar escritas y compiladas específicamente para iOS y la arquitectura ARM. El navegador web Safari soporta aplicaciones web como otros navegadores. Hay disponibles aplicaciones nativas autorizadas de terceras partes para dispositivos con iOS 2.0 o posterior en el *app store* de Apple.

SDK

En octubre del 2007, en una carta abierta, Steve Jobs anunció que en febrero del 2008 se pondría a disposición de los desarrolladores externos de Apple un SDK. El SDK se lanzó en marzo del 2008 y permite a los desarrolladores hacer aplicaciones para el iPhone y el iPod touch, así como probarlas en un simulador iPhone. No obstante, cargar una aplicación en los dispositivos es solo posible después de pagar al iPhone Developer Program. Desde el lanzamiento de Xcode 3.1, Xcode es el entorno de desarrollo para el iOS SDK. Las aplicaciones iPhone, como el iOS y el Mac OS X, están escritas en Objective-C.

Los desarrolladores pueden poner cualquier precio a sus aplicaciones (por encima de un mínimo) para que se distribuyan en el *app store*, del que recibirán el 70% por cada venta. También pueden optar por lanzar su aplicación de forma gratuita y, de esta manera, no pagan ningún coste para lanzar o distribuir la aplicación, excepto el coste miembro.

5.1.6. *Jailbreaking*

iOS ha estado sujeto a una variedad de diferentes manipulaciones centradas en añadir funcionalidad sin el apoyo de Apple. Antes del debut del *app store* en el 2008, la razón principal para el *jailbreaking* era instalar aplicaciones nativas de terceras partes. Apple dijo que no diseñaría actualizaciones de *software* específicamente para que estas aplicaciones dejaran de funcionar (siempre que no fueran aplicaciones que hicieran desbloqueo de SIM[26]), pero el caso es que con cada actualización de iOS el *jailbreak* parecía dejar de funcionar.

[26] SIM (*subscriber identity module*): módulo de identificación de suscriptor

Desde la llegada del *app store* y las aplicaciones de terceras partes, el objetivo de la comunidad de *jailbreaking* ha cambiado. El principal objetivo del *jailbreaking* es permitir la personalización del dispositivo, usar emuladores y mejoras hechas por la comunidad como multitarea, Adobe Flash Player o acceder al sistema de ficheros del iPhone. La multitarea solo está disponible en dispositivos iOS de tercera generación en adelante, y las aplicaciones en el *app store* no tienen permiso para modificar la apariencia del sistema operativo.

Algunos *jailbreakers* también intentan compartir de forma ilegal aplicaciones de pago del *app store*. Este objetivo ha causado alguna distensión dentro de la comunidad de *jailbreaking*, debido a que no era el objetivo original del *jailbreaking* y es ilegal. Hay también algunos usuarios que se oponen a la censura de contenidos de Apple.

5.1.7. Gestión de derechos digitales

La naturaleza cerrada y propietaria del iOS ha generado críticas, particularmente de abogados de derechos digitales como la Electronic Frontier Foundation, el ingeniero informático y activista Brewster Kahle, el especialista en leyes de Internet Jonathan Zittraion y la Free Software Foundation, que protestó en el evento de presentación del iPad y ha hecho del iPad su objetivo con su campaña *Defective by design*. El competidor Microsoft también ha criticado el control de Apple sobre su plataforma.

En el conflicto están las restricciones impuestas por el diseño del iOS, conocidas como DRM[27], destinadas a bloquear los contenidos que se compran a la plataforma Apple, el modelo de desarrollo (que requiere una suscripción anual para distribuir aplicaciones desarrolladas para el iOS), el proceso de aprobación de aplicaciones centralizado, así como el control general de Apple sobre la plataforma en sí misma. Particularmente en disputa está la capacidad de Apple para inhabilitar o borrar aplicaciones de forma remota.

[27] DRM (*digital rights management*) gestión de derechos digitales

Algunas voces dentro de la comunidad tecnológica han expresado su preocupación por el hecho de que el iOS cerrado represente una tendencia cada vez mayor hacia la visión de Apple de la informática, y hacen notar el potencial de estas restricciones para reducir la innovación en *software*.

No obstante, también hay voces fuera de Apple que han mostrado su apoyo al modelo cerrado del iOS. El desarrollador de Facebook Joe Hewitt, que protestó contra el control de Apple sobre su *hardware* como un "precedente horrible", ha argumentado después que las aplicaciones cerradas en el iPad están relacionadas con las aplicaciones web y proporcionan mayor seguridad.

5.2. Android

Android es una pila de *software* de código abierto para dispositivos móviles que incluye sistema operativo, *middleware* y aplicaciones básicas. Google Inc., compró la empresa desarrolladora inicial del *software*, Android Inc., en el 2005. El sistema operativo de Android está basado en una versión modificada del *kernel* de Linux. Google y otros miembros de la Open Handset Alliance colaboran en el desarrollo y lanzamiento de Android. El AOSP[28] está encargado del mantenimiento y desarrollo de Android.

(28)AOSP (*Android open source project*)

En el cuarto trimestre del 2010, el sistema operativo Android fue la plataforma de *smartphone* más vendida del mundo, destronando al Symbian de Nokia de la primera posición por primera vez en 10 años. Otras fuentes indican que Symbian estaba aún ligeramente por delante en ventas, si se tenían en cuenta algunos teléfonos de modelos antiguos Symbian no Nokia.

Android tiene una gran comunidad de desarrolladores programando *apps*[29], que extienden la funcionalidad de los dispositivos. En el 2010 había alrededor de doscientas mil *apps* disponibles para Android. El *Android Market* es la tienda "en línea" gestionada por Google mediante la que también se pueden descargar *apps* de sitios de terceras partes. Los desarrolladores programan principalmente en el lenguaje Java y controlan el dispositivo mediante librerías Java desarrolladas por Google.

(29)apps: *application programs*

Lanzamiento de Android

La llegada de la distribución Android, en noviembre del 2007, se anunció con la fundación de la Open Handset Alliance, un consorcio de setenta y nueve compañías de *hardware*, *software* y telecomunicaciones, con el objetivo de desarrollar estándares abiertos para dispositivos móviles. Google lanzó la mayor parte del código Android bajo la licencia Apache, una licencia de *software* libre y código abierto.

La pila de *software* del sistema operativo Android consiste en aplicaciones Java que se ejecutan en un *framework* de aplicaciones basado en Java y orientado a objetos encima de librerías base Java, que se ejecutan en una máquina virtual Dalvik, la cual realiza compilación JIT[30]. También hay librerías escritas en C que incluyen el gestor de superficie, el OpenCore Media Framework, el sistema gestor de base de datos relacional SQLite, la API gráfica 3D OpenGL ES 2.0, el WebKit layout engine, el motor gráfico SGL[31], SSL, y Bionic libc. El sistema operativo Android consiste en doce millones de líneas de código que incluyen tres millones de líneas de XML, 2,8 millones de líneas de C, 2,1 millones de líneas de Java y 1,75 millones de líneas de C++.

(30)JIT (*just in time*)

(31)SGL (*scene graph library*)

5.2.1. Historia de Android

En octubre del 2003, Andy Rubin, Rich Miner y otros fundaron Android Inc. en Palo Alto, California (EE. UU).

En palabras de Rubin, el objetivo era desarrollar "dispositivos móviles más elegantes que tuvieran más en cuenta la localización y las preferencias de sus dueños".

Entre otros empleados iniciales importantes se incluyen Andy McFadden, que trabajó con Rubin en WebTV, y Chris White, que lideró el diseño y la interfaz de WebTV antes de ayudar a fundar Android.

Rubin, cofundador de Danger Inc., Miner, cofundador de Wildfire Communications Inc. y vicepresidente de tecnología e innovación en Orange, y los otros empleados iniciales llevaron una considerable experiencia en la industria inalámbrica a la compañía. A pesar de los logros obvios del pasado de los fundadores y de los primeros empleados, Android Inc. funcionó de forma reservada y simplemente admitió que estaba trabajando en *software* para teléfonos móviles.

En agosto del 2005, Google adquirió Android Inc. Los empleados principales de Android Inc., entre los que se encontraban Andy Rubin, Rich Miner y Chris White, permanecieron en la compañía después de la adquisición.

En el momento de la adquisición, debido al poco conocimiento que se tenía sobre el trabajo de Android Inc., se conjeturó que Google estaba planeando entrar en el mercado de los teléfonos móviles.

En Google, el equipo liderado por Rubin desarrolló una plataforma para dispositivos móviles basada en el *kernel* de Linux. Google puso en el mercado la plataforma para los fabricantes de dispositivos móviles y los operadores con la premisa de proporcionar un sistema flexible y actualizable. Google alineó a una serie de socios dedicados a componentes *hardware* y *software* e indicó a los operadores que estaba abierto a varios grados de cooperación por su parte.

Las especulaciones sobre la intención de Google de entrar en el mercado de las comunicaciones móviles continuaron durante diciembre de 2006. Informes de la BBC[32] y The Wall Stret Journal indicaron que Google quería su sistema de búsqueda y sus aplicaciones en teléfonos móviles y que estaba trabajando duro para conseguirlo. Algunos medios de comunicación escritos y "en línea" publicaron rápidamente rumores de que Google estaba desarrollando un dispositivo con la marca de fábrica Google. Algunos especularon que, mientras Google definía especificaciones técnicas, estaba mostrando prototipos a fabricantes de teléfonos móviles y operadores de red.

(32)BBC (British Broadcasting Corporation)

En septiembre del 2007, InformationWeek cubrió un estudio de Evalueserve que indicaba que Google había registrado varias patentes de aplicaciones en el área de la telefonía móvil.

En noviembre del 2007, se anunció a sí misma Open Handset Alliance, un consorcio de varias compañías, entre las que se encuentran Texas Instruments, Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group,

Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel y T-Mobile. El objetivo de la Open Handset Alliance es desarrollar estándares abiertos para dispositivos móviles. El mismo día, la Open Handset Alliance también anunció su primer producto, Android, una plataforma para dispositivos móviles construida sobre la versión 2.6 del *kernel* de Linux.

En diciembre del 2008 se unieron catorce nuevos miembros, entre los que se encontraban PacketVideo, ARM Holdings, Atheros Communications, Asustek Computer Inc., Garmin Ltd., Softbank, Sony Ericsson, Toshiba Corp. y Vodafone Group Plc.

Excepto durante breves periodos de actualización, Android ha estado disponible bajo una licencia de *software* libre de código abierto desde octubre de 2008. Google publicó todo el código fuente (incluyendo las pilas de red y telefonía) bajo una licencia Apache. Google también mantiene pública la lista de problemas revisados para que cualquiera pueda verla y comentarla.

5.2.2. Historia de las versiones de Android

Android ha visto varias actualizaciones desde su lanzamiento original. Estas actualizaciones para el sistema operativo base normalmente arreglan fallos y añaden nuevas funcionalidades. Generalmente, cada nueva versión del sistema operativo Android se desarrolla bajo un nombre código basado en un artículo de postre.

Las versiones más recientes de Android son:

- 2.0/2.1 (**Eclair**), que mejoró la interfaz de usuario e introdujo soporte a HTML5 y Exchange ActiveSync 2.5.

- 2.2 (**Froyo**), que introdujo mejoras de velocidad con la optimización del JIT y el motor Chrome V8 JavaScript, y añadió soporte para Adobe Flash.

- 2.3 (**Gingerbread**), que refinaba la interfaz de usuario, mejoraba el teclado *software* y las características del copiar y pegar, y añadía soporte para NFC.

Glosario

AJAX *f* AJAX (*asynchronous JavaScript and XML*), JavaScript asíncrono y XML, es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (*rich Internet applications*). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios, mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma, es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo cual aumenta la interactividad, la velocidad y la usabilidad en las aplicaciones.

Dalvik *f* Dalvik es la máquina virtual que utiliza la plataforma para dispositivos móviles Android. Dalvik ha sido diseñada por Dan Bornstein con contribuciones de otros ingenieros de Google.

jailbreak *m* *Jailbreak* (en español, escaparse de la cárcel o, más literalmente, rompecárcel) es un proceso que permite a los usuarios de los dispositivos iPhone, iPod touch, iPad y Apple TV de todas las generaciones ejecutar aplicaciones distintas a las alojadas en *app store*, el sitio oficial de descarga de programas para estos dispositivos.

licencia Apache *f* La licencia Apache (Apache *license* o Apache *software license* para versiones anteriores a 2.0) es una licencia de *software* libre creada por la Apache Software Foundation (ASF). La licencia Apache (con versiones 1.0, 1.1 y 2.0) requiere la conservación del aviso de *copyright* y el *disclaimer*, pero no es una licencia *copyleft*, ya que no requiere la redistribución del código fuente cuando se distribuyen versiones modificadas.

licencia MIT *f* La licencia MIT es una de tantas licencias de *software* que ha empleado el Instituto Tecnológico de Massachusetts (MIT, Massachusetts Institute of Technology) a lo largo de su historia, y quizás sería más acertado llamarla licencia X11, ya que es la licencia que llevaba este *software* de muestra de la información de manera gráfica. En cualquier caso, tanto MIT como X11 tienen el mismo texto.

MIDP *m* El *mobile information device profile* (MIDP) permite escribir aplicaciones descargables y servicios para dispositivos móviles que se conecten a la red.

webcast *m* Un *webcast* es un fichero multimedia distribuido por Internet mediante tecnologías de *streaming* para distribuir una única fuente de contenido a varios observadores simultáneos. Un *webcast* se puede distribuir en directo o bajo demanda.

Bibliografía

Marco, M. J.; Marco, J. M.; Prieto, J. y otros (eds.) (2010). *Escaneando la informática*. Barcelona: Editorial UOC. ISBN: 978-84-9788-110-4.

Stark, J. (2010). *Building iPhone apps with HTML, CSS, and JavaScript*. Editorial O'Reilly. ISBN: 978-0-596-80578-4.

Enlaces de Internet

http://en.wikipedia.org/

http://www.visionmobile.com/

http://blog.abrahambarrera.me/

http://developer.appcelerator.com/

http://www.phonegap.com/

http://softlibre.barrapunto.com/

http://www.nitobi.com/

http://msdn.microsoft.com/

http://forums.techarena.in/

http://stackoverflow.com/

http://developer.apple.com/

Métodos para el desarrollo de aplicaciones móviles

Robert Ramírez Vique

PID_00176755

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

En este módulo nos centraremos en los problemas que surgen en el desarrollo de aplicaciones y servicios móviles. En él os mostraremos, desde un punto de vista general, las oportunidades y dificultades propias del entorno.

El desarrollo de una aplicación o servicio conlleva una gran incertidumbre. Sin embargo, existen sistemas para paliar los riesgos asociados. En el caso del las aplicaciones móviles, las dificultades son mayores, si cabe (algunos problemas ya existían con los primeros desarrollos móviles, como la fragmentación o la calidad del servicio de las redes de telefonía). Con el tiempo, han ido apareciendo nuevas dificultades, como el acceso a la información del entorno o el control de las diferentes capacidades de los dispositivos. Al mismo tiempo, las oportunidades de negocio aparecen constantemente, lo que permite crear desde juegos de gran complejidad (reservados hasta ahora a consolas de gran potencia) hasta aplicaciones que nos ayuden a amueblar nuestro hogar.

Debido a esta situación, resulta muy difícil ofrecer una receta mágica para el desarrollo de aplicaciones móviles y, por tanto, se hace imprescindible aprender y adaptar los métodos y los conocimientos adquiridos. En este módulo os explicaremos las situaciones, los métodos y las estrategias oportunas para minimizar estos riesgos e implementar las soluciones móviles, así como para conseguir el mejor rendimiento de las capacidades de los dispositivos.

En el pasado se ha hablado de las aplicaciones móviles y, a pesar de que los móviles ya tenían una gran penetración en el mercado y de que su uso como herramienta de trabajo o elemento de la vida diaria era bastante común, las aplicaciones móviles no habían acabado de despegar. Las razones son varias, desde el intento infructuoso de conseguir aplicaciones ejecutables en todos los dispositivos, hasta el coste asociado a las mismas, lo que ha hecho que solo algunas aplicaciones hayan sido ampliamente usadas (como, por ejemplo, el SMS y el MMS).

SMS

SMS (*short message service)*

MMS

MMS (*multimedia message system*)

Actualmente, más del 70% de la población dispone de dispositivos móviles. El número de *smartphones* no para de crecer (el 90% de los nuevos dispositivos son *smartphones*, según los estudios de Gartner). Es, sin lugar a dudas, el sector que mayor innovación y expectación está generando y generará. Actualmente se dan muchos factores que hacen que casi nadie quede fuera del ecosistema móvil, por lo que es un momento perfecto para conocer mejor sus entresijos. Algunos de estos factores son los que explicamos a continuación:

- Las mejoras en las características *hardware* de los dispositivos móviles gracias a la inclusión de los fabricantes de la electrónica de consumo, que han visto un nicho de negocio y no quieren perder la oportunidad.

- La diversidad en las plataformas y dispositivos, de manera que se puede cubrir un gran abanico de posibles consumidores. Además, aparecen novedades a un gran ritmo, que no parece decaer. Sin duda, hay un papel especial para algunas apariciones, como son las de iOs (iPhone, iTouch y iPad) y Android, que han dado una perspectiva diferente.

- El uso generalizado de los dispositivos móviles (*smartphones*, *tablets pc*, televisores, etc.) en muchos aspectos de la vida cotidiana, que ha permitido que entren en muchos mercados. Lo que antes parecía reservado a las escenas de ciencia ficción, hoy está al alcance de la mano.

- La popularización (en aumento) de las tarifas de Internet móvil para conseguir una mayor cuota de mercado.

- La aparición de una gran cantidad de nuevas aplicaciones a diario, disponibles para el gran público gracias a las tiendas de aplicaciones o *market places*.

- Las nuevas formas o facilidades de venta de las aplicaciones, que hacen más atractivo para las empresas el desarrollo de aplicaciones para este tipo de dispositivos.

- La aparición de las redes sociales, cuyo propósito se ve complementado y potenciado con las aplicaciones móviles.

Sin duda, esto nos obliga, como profesionales del sector, a conocer los retos y posibilidades de este entorno.

En este módulo veremos, para empezar, una introducción a la situación del desarrollo de aplicaciones móviles. En ella, veremos por qué es peculiar y qué lo diferencia de otros procesos de construcción de aplicaciones.

Después, explicaremos detalladamente un método de desarrollo de aplicaciones móviles y expondremos las mejores prácticas en cada una de las fases del desarrollo.

Finalmente, repasaremos las opciones de negocio posibles en mundo de los móviles.

Objetivos

Con este módulo queremos proporcionaros un conocimiento amplio y variado de las alternativas para el desarrollo de aplicaciones móviles. En concreto, con el estudio de este módulo, pretendemos que consigáis los siguientes objetivos:

1. Que conozcáis los problemas de los desarrollos de aplicaciones para móviles.

2. Que veáis las restricciones y posibilidades de dichas aplicaciones.

3. Que conozcáis un método de desarrollo (pondremos especial atención a los problemas de las aplicaciones para dispositivos móviles).

4. Que conozcáis las herramientas necesarias para aplicar dicho método a las nuevas tecnologías emergentes.

5. Que seáis capaces de afrontar todas las fases de un proyecto relacionado con el desarrollo de aplicaciones móviles y dispongáis de herramientas para afrontarlo con garantías.

1. Ecosistema de aplicaciones móviles

Por ecosistema móvil nos referimos al conjunto de actores necesarios para poder tener los dispositivos móviles y a las aplicaciones para los mismos. En concreto, en el ecosistema móvil se incluyen las operadoras de telecomunicaciones, los fabricantes de *hardware* y todos los elementos de *software* que intervienen en la ejecución de la aplicación.

Todas las aplicaciones se ejecutan dentro de un ecosistema. Por lo tanto, para conseguir un desarrollo satisfactorio, es ideal conocerlo. Existen varios factores que afectan al ecosistema, como la infraestructura de la aplicación, el sistema operativo, los métodos de entrada de información, los propios usuarios, los canales de distribución de la aplicación, etc.

Por ejemplo, en el caso de las aplicaciones web, un punto característico es que debemos acceder a ellas mediante un navegador; esto condiciona muchas otras cosas, y para poder hacer una buena aplicación web, se debe conocer, sin duda, esta información. En el caso de las aplicaciones de sobremesa, tenemos un mayor control, pero también tenemos mayor diversidad, debido a los diferentes sistemas operativos disponibles. Lo mismo sucede con los servidores y con las diferentes redes o protocolos que tienen que soportar.

En el caso de las aplicaciones móviles, el ecosistema es aun más heterogéneo que en el resto de desarrollos. Pueden ejecutarse en diferentes tipos de dispositivo, ya sea en un móvil antiguo o bien en uno nuevo, un *smartphone* o un *table PC*, o incluso en aparatos menos evidentes, como un televisor o una *smart card*. Estos dispositivos suelen estar conectados a Internet mediante una conexión que se contrata con una operadora. Todo esto compone, como podéis ver en la siguiente tabla, un ecosistema con muchos actores a tener en cuenta para el desarrollo de aplicaciones móviles.

Smart cards

Las smart cards o tarjetas inteligentes son tarjetas que tienen un circuito integrado de tamaño de bolsillo en el que se puede programar algún tipo de lógica. Un ejemplo son las tarjetas de crédito con microchip.

Ecosistema de los dispositivos móviles

Las diferentes capas de actores que influyen hasta conseguir un servicio (como, por ejemplo, SMS o Internet móvil).

Servicios
Aplicaciones
Framework de aplicaciones
Sistemas operativos
Plataformas
Dispositivos
Redes (GPRS, 3G, etc.)
Operadoras

Sin duda, el ecosistema de una aplicación para dispositivos móviles es algo que no nos planteamos en un primer momento. Este ecosistema constituye una de las mayores dificultades en lo que respecta al desarrollo de aplicaciones, ya que acaba causando, entre otras cosas, una mayor fragmentación de la aplicación. Esto se implica que el desarrollador debe tener en cuenta muchos factores para que la aplicación funcione como desea.

Dentro de este ecosistema, disponemos de información que puede ser muy útil para nuestras aplicaciones, desde la información de la red de datos actual para adaptar los contenidos hasta la información del propio dispositivo (como, por ejemplo, su posición geográfica). También encontramos capacidades que en otros entornos no encontraríamos, como la de localizar otros dispositivos en movimiento, la de dar información sobre nuestro entorno (localización, orientación, presión atmosférica, etc.), la de conseguir información sobre el usuario (contactos, calendario, etc.) o, incluso, la de disponer de medios de pago mucho más directos (mediante la operadora o mediante el propio dispositivo).

En este apartado os mostraremos los retos y las oportunidades que nos ofrece este ecosistema móvil.

1.1. Fragmentación

Uno de los principios básicos para desarrollar aplicaciones consiste en intentar tener el código más simple posible, de manera que se reduzca la complejidad, se eviten los posibles errores y se facilite el mantenimiento. Esto ha resultado muy difícil debido a la fragmentación, que existe en los entornos de aplicaciones más conocidos.

> La fragmentación es una situación, o el conjunto de condicionantes de una situación, en la que no es posible compartir una misma aplicación entre diferentes ecosistemas. Es decir, la fragmentación impide que se pueda compartir la aplicación sin adaptar los ecosistemas.

Esta fragmentación puede ocurrir por muchos factores, los cuales provocan diversidad y entropía en las aplicaciones que queremos crear. La fragmentación puede originarse por los siguientes motivos:

- ***Hardware* diferente**: Como, por ejemplo, dispositivos con componentes distintos: tamaño o densidad de la pantalla, teclado, sensores, capacidad de proceso, etc.

- ***Software* diferente**:

  - **Plataforma diferente**. Una plataforma, un *framework*, un sistema operativo (o las versiones de cualquiera de ellos) puede generar la fragmentación de las aplicaciones.

  - **Diferencias en las implementaciones**. Por ejemplo, diferencias en la implementación del estándar, o bien errores conocidos de versiones concretas.

- **Variaciones de las funcionalidades**. Por ser una versión con menos privilegios (versión de pago y versión gratuita) o según los roles de los propios usuarios de la aplicación.

- **Preferencias de usuario**. Las más habituales son las localizaciones de la aplicación (idioma, orientación del texto, etc.).

- **Diversidad del entorno**. Derivado de la infraestructura, como pueden ser los operadores y sus API, los problemas de cortafuegos, las limitaciones de las redes, el *roaming*, etc.

Roaming

El roaming o itinerancia es un concepto relacionado con la capacidad de un dispositivo de moverse de una zona de cobertura a otra.

Esta fragmentación puede afectar a todo el proyecto de desarrollo, desde el modelo de negocio hasta el despliegue, además de la implementación y las pruebas. Es imprescindible tratarla muy seriamente.

Si no se trata correctamente, esta fragmentación puede causar muchos problemas. A continuación, os mostramos algunos:

- Reducir la calidad del producto. Debido a la mayor complejidad de las soluciones fragmentadas, se pueden generar más errores.

- Limitar el número de dispositivos soportados. Para evitar este problema, se puede decidir soportar un número menor de dispositivos (con posibilidades de ampliaciones en un futuro).

- Alargar cualquier fase del proyecto, desde las fases iniciales a la implementación, además del mantenimiento. Esta dilatación en el tiempo supondrá un sobreprecio, así como posible fracaso de dicho proyecto.

- Grandes costes asociados a las pruebas sobre dispositivos reales.

Sin duda, la fragmentación ha sido, y seguramente será, la mayor dificultad y el mayor riesgo en el desarrollo de aplicaciones móviles.

La fragmentación puede tener diferentes grados. No es lo mismo atacar la fragmentación de una aplicación que debe ejecutarse sobre un televisor y sobre un teléfono móvil, que la de una aplicación que debe ejecutarse sobre dos versiones de la misma plataforma. Por esta razón, existen diferentes estrategias para combatirla, y cada una tiene un sentido según el caso concreto.

1.1.1. Un desarrollo para cada escenario

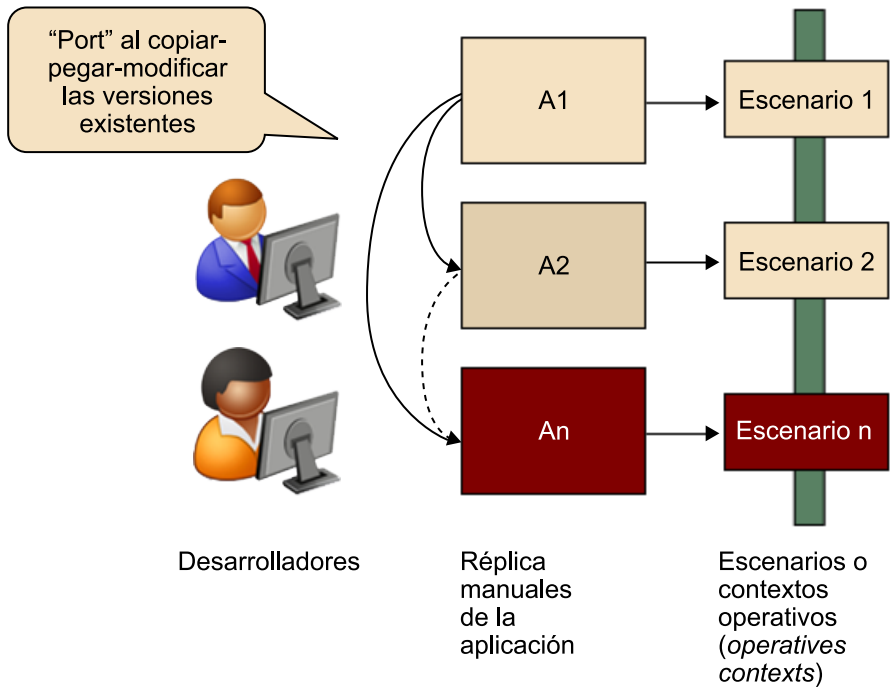
Un escenario es un caso de fragmentación que presenta cualquiera de las posibles causas de fragmentación o varias de ellas.

Es decir, se realiza todo un desarrollo para cada fragmentación que nos podamos encontrar, sin compartir nada. Esto suele ser útil en los casos en que los escenarios son muy diferentes.

El proceso de adaptar la aplicación a un nuevo escenario se llama portar la aplicación.

Esta estrategia es la más costosa de todas, pues no se puede aprovechar nada (o casi nada) sin adaptaciones del código realizado en otros escenarios. Sin embargo, podemos aprovechar al máximo la capacidad del dispositivo y del lenguaje, así como las últimas novedades.

Fragmentación de aplicaciones debido a factores de diversidad



1.1.2. Parte común y derivaciones

La **derivación** es la estrategia más habitual. Según esta estrategia, una parte de nuestra aplicación es común a todos nuestros escenarios, y para cada uno de ellos podemos definir la parte específica correspondiente.

Existen diversas variantes de esta estrategia, en función de cómo se realice la derivación a los diferentes escenarios. Estas derivaciones realizan los cambios específicos de cada escenario para que todos funcionen de la misma manera. Tenemos, pues, las siguientes opciones:

- **Derivación selectiva**. Las modificaciones necesarias están localizadas en unos elementos concretos (ya sean clases del código, ficheros de marcado, estilos u otros recursos), y existe un sistema o herramienta que genera las diferentes versiones mediante la captura y el empaquetado de estos elementos para cada escenario.

- **Derivación usando meta programación**. Se trata de programar algo que se va a ejecutar en varios escenarios. Para conseguir distintos comportamientos hay varias opciones:
  - Mediante la inyección de objetos o recursos (imágenes, ficheros XMLS, etc.) en el código, de manera que nuestra aplicación deje estos objetos vacíos y se rellenen, en tiempo de ejecución, los objetos específicos de cada escenario. Esta estrategia se baja en el patrón de diseño *Inversion of control*.

  - Utilizando preprocesadores, que se encargan de cambiar o ampliar nuestro código para adaptarlo a los diferentes escenarios antes de ejecutar.

- **Generación automática**. El *software* se debe escribir de una manera específica y solo una vez. A posteriori, existe un proceso que genera automáticamente las aplicaciones correctas para cada escenario, normalmente transformando nuestra aplicación al código particular de cada escenario. En este punto existen más variantes.

Esta estrategia puede ser menos costosa que la anterior, ya que se puede aprovechar parte del desarrollo de las alternativas y reducir así los costes de implementación. Sin embargo, a menudo requiere conocer tanto los lenguajes o entornos de los diferentes dispositivos como las herramientas (o, incluso, los lenguajes) necesarias para llevar a cabo las adaptaciones. En este caso, se suelen conseguir aplicaciones aprovechando, en gran medida, el potencial de los dispositivos, aunque según la estrategia escogida, puede ocurrir que perdamos el control sobre el código generado y, por lo tanto, potencia de desarrollo.

1.1.3. Adaptación única

Mediante la adaptación única, podemos conseguir una versión que funcione en todos los casos sin necesidad de realizar más cambios. Existen varias maneras de afrontar esta estrategia:

- **Mínimo común denominador**. Se trata de conseguir una aplicación mediante la reducción de los puntos de fragmentación, de manera que no exista la necesidad de adaptar la aplicación.

- **Todos en uno**.
  - La aplicación es capaz de conocer la información necesaria para poder adaptarse a todos los dispositivos. Por ejemplo, para evitar el problema de diferentes pantallas, se genera una aplicación con ventanas autoescalables.

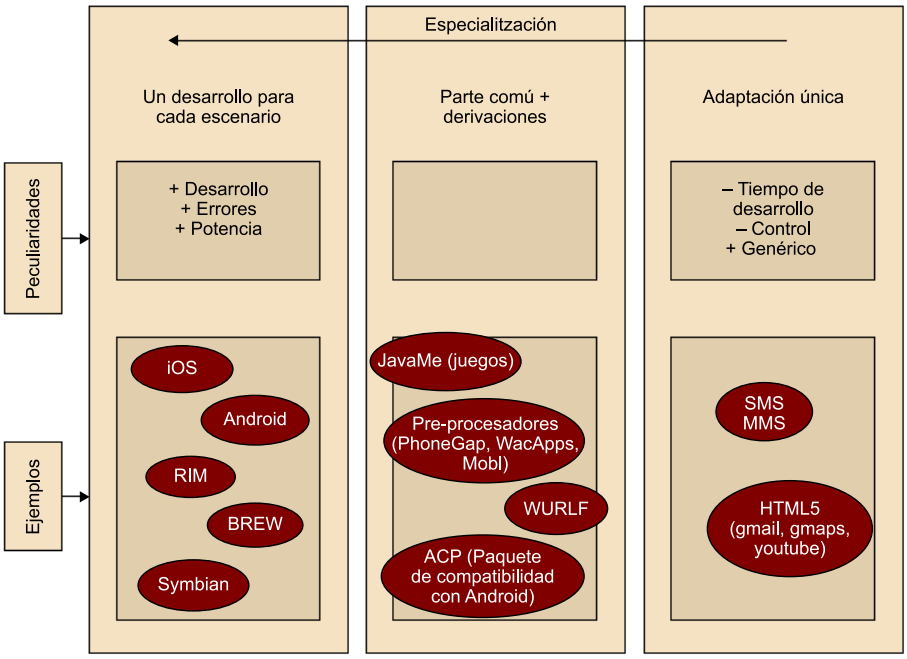
  - Son los dispositivos los que se adaptan; es decir, el *software* se escribe de manera abstracta y, cuando se llega al problema de la fragmentación, se pasa el testigo al dispositivo que sabe cómo tratarlo. Un ejemplo puede ser el acceso a los contactos o a las llamadas de teléfonos mediante API[1] abstractas.

(1) *Application program interface*

Esta estrategia es la más económica en lo que respecta a desarrollos distintos y conocimientos requeridos. Como podéis intuir, también es la que se queda en la capa más superficial del potencial de los dispositivos.

A pesar de esto, se consiguen soluciones potentes, ya que con este tipo de aplicaciones se pueden realizar una gran variedad de aplicaciones, desde aplicaciones muy simples y generales (como puede ser una aplicación basada en SMS) hasta aplicaciones como Google Places, que conoce nuestra localización, u otras que pueden trabajar con datos sin conexión (modo "fuera de línea").

Categorización de tipos de adaptaciones a los problemas de fragmentación



1.2. Contexto

El contexto se define como las informaciones conjuntas de la situación actual, el usuario, la información del dispositivo y la información de otras aplicaciones en un momento dado del tiempo.

Las aplicaciones móviles pueden aprovechar mucho más el contexto en el que están ejecutando, sobre todo si las comparamos con aplicaciones tradicionales. Esto se debe a diferentes factores, entre los que se encuentran las nuevas capacidades de los dispositivos, la capacidad de acceder a la información que el propio dispositivo tiene del usuario (incorporando, de esta manera, las capacidades o relaciones sociales del mismo) y las capacidades que puede aportar el entorno en el que estamos o el momento en que usamos la aplicación.

1.2.1. Capacidades de los dispositivos

Los nuevos dispositivos nos aportan mucha información sobre nuestro entorno. Por ejemplo, la más clara y conocida es la posición geográfica actual, que nos permite realizar aplicaciones basadas en la localización (LBS). Además, existen otras informaciones, (como, por ejemplo, la orientación, la presión, la luz, etc.). La posibilidad de grabar imágenes, vídeos y audio también nos aporta más información sobre el contexto (las aplicaciones que reaccionan al habla o las de realidad aumentada son ejemplos de aplicaciones que aprovechan este tipo de contexto). Existen capacidades más obvias (como la de saber la hora actual, el idioma, o la zona horaria) que ayudan a realizar aplicaciones

Aplicaciones basadas en la localización

Las aplicaciones basadas en la localización, en inglés *location based services* (LBS), son servicios que intentan dar un valor añadido gracias al conocimiento de la ubicación geográfica del usuario.

más contextualizadas. Otro aspecto que ha hecho mejorar la situación es, sin duda, la mejora de las comunicaciones inalámbricas a partir de la aparición del 3G o del UMTS, o la mejoras de Bluetooth o nuevos estándares de WIFI.

Realidad aumentada

La realidad aumentada (RA) es el término que se usa para definir una visión directa o indirecta de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta a tiempo real.

Muchas de las aplicaciones aprovechan varias capacidades. Por ejemplo, una aplicación de realidad aumentada aprovecha varias de ellas:

- GPS, para conocer la posición geográfica y saber qué se debe mostrar.
- Brújula, para saber la orientación actual
- Acelerómetro, para saber cuál es la orientación exacta de nuestro dispositivo y superponer las capas.
- Cámara, para poder captar nuestro alrededor y así ampliar la información (en ocasiones, incluso, varias cámaras).
- Conexión a Internet, para poder obtener la información para ampliar nuestra realidad. Esta conectividad puede venir mediante Internet móvil o WiFi, entre otros.
- Capacidad de procesamiento gráfico muy mejorada, con chips de aceleración gráfica potentes.

Además, están apareciendo nuevos protocolos o capacidades que ayudarán a mejorar las aplicaciones móviles, como es el caso de *NFC*. NFC, además de permitir realizar pagos con el móvil, permitirá comunicar información de entorno, como puede hacerse ahora con RFID, para llegar a aplicaciones móviles (por ejemplo, una visita guiada en un museo con información constante sobre lo que estamos viendo).

Sin duda, el salto en los últimos años ha sido muy grande con respecto a los móviles que "únicamente" eran capaces de llamar y enviar mensajes. Ahora tenemos al alcance de la mano muchas funciones que aportan posibles aplicaciones para conocer mucho mejor nuestro entorno.

1.2.2. Ubicuidad

La ubiquidad (o la omnipresencia) se define como la capacidad de acceder a toda la información o a todos los servicios que necesita el usuario en cualquier momento y circunstancia mediante el dispositivo que tengamos actualmente.

Con la situación actual de las aplicaciones móviles, estamos muy cerca de conseguir esta ubicuidad, pues las aplicaciones móviles pueden ir con el usuario continuamente. De hecho, se han convertido en accesorio diario, tanto en el plano profesional como en el personal. Gracias a las nuevas capacidades, tanto de conectividad como de proceso, podemos acceder a casi todos los servicios disponibles en Internet. El acceso a estos servicios puede llegar a ser más práctico que desde un ordenador personal, y en eso se centra el aprovechamiento de la ubicuidad.

En este sentido, en cualquier momento podemos recibir información útil para nosotros en el momento actual, como por ejemplo un correo electrónico o un mensaje instantáneo, pero también puede ser información sobre el lugar donde estamos en ese momento o recordatorios relacionados con el sitio o la persona con quien estamos.

También podemos conocer el estado del tiempo, del tráfico y muchos otros parámetros en el momento y lugar que nos interesa. En el caso de aplicaciones para otro tipo de dispositivos, como un televisor, también podemos tener información contextualizada, como juegos, concursos o simplemente comentarios relacionados con el programa que estemos viendo en ese momento.

Por lo tanto, tenemos una serie de beneficios relacionados con la ubiquidad en lo que respecta a las aplicaciones para dispositivos móviles:

- El dispositivo móvil es el primer medio de comunicación masivo real, dado que es capaz de llegar a casi todos los usuarios y en todo momento.
- El primer medio de comunicación permanentemente encendido, pues es capaz de captar y enviar información aun cuando está apagado (apagado en el sentido del usuario, es decir, cerrado, pero no totalmente apagado).
- Primer medio que está siempre con el usuario.
- Medio masivo que tiene incorporado un sistema de pago, que en este caso es mediante la operadora.

Media masivo

Media masivo o medio de comunicación de masas son los medios de comunicación recibidos simultáneamente por una gran audiencia.

1.2.3. Contexto social

Otro punto importante a tener en cuenta y que, sin duda, ayuda mucho a que una aplicación triunfe actualmente en el sector es su componente social. Esto significa la posibilidad de interaccionar mediante nuestras aplicaciones con nuestros amigos, nuestra familia y otros conocidos (o incluso desconocidos).

Ejemplos de interacción

A continuación exponemos algunos ejemplos de interacción:

- Compartir nuestra puntuación en un juego.
- Ver usuarios que están actualmente interaccionando con la aplicación, o bien con el tema que nos interesa en ese momento (por ejemplo, poder comentar una serie de televisión mientras la vemos).
- Publicar en cualquier momento lo que se estamos haciendo y ver lo que hacen nuestros amigos.
- Recibir avisos de cuando nuestros amigos están cerca, llegan a un lugar concreto o hacer alguna acción destacable.

Obviamente, el gran crecimiento de las redes sociales actuales es el caldo de cultivo ideal para este tipo de aplicaciones sociales móviles (MoSoSo[2]), pues actualmente hay una gran penetración de las redes en Internet a nivel mundial. Incluso hay casos, como Android, donde para sacarle el mayor partido a nuestro dispositivo es necesario acceder con unas credenciales de Google y su capa social, cosa que automáticamente nos conecta con todos nuestros amigos de dicho servicio y permite sacarle el máximo partido a muchas de nuestras aplicaciones.

(2)MoSoSo (*mobile social software*)

Las aplicaciones sociales gozan de gran proyección en el móvil debido a la capacidad de omnipresencia (o ubicuidad) de dichas aplicaciones; es decir, que nos acompañan en el momento adecuado. Por ejemplo, podemos subir las fotos que acabamos de hacer a nuestra red social o acceder a la información de nuestros contactos.

1.2.4. Costes

Sin duda, que esté contextualizada es un gran punto a favor para nuestra aplicación, pero como toda aportación, tiene su desventaja, que en este caso tiene que ver con los costes. Así, para poder tener este contexto, necesitamos lo siguiente:

- Acceso a Internet mediante redes WiFi o MWWAN.
- Proximidad de otros dispositivos para compartir información.
- Estar constantemente conectado a las redes sociales, lo que requiere que dispongamos de cuentas en esas redes sociales y que tengamos activado el acceso (con los posibles problemas de privacidad).
- Conexiones a otras redes inalámbricas, Bluetooth, GPS, NFC, etc.
- Grandes capacidades de proceso en nuestros teléfonos para realizar las acciones, ya sea la capacidad de proceso interno o bien mediante accesorios.
- Debido a innovaciones y cambios, tenemos que pagar el coste de actualizar nuestra aplicación. Es decir, durante o después del lanzamiento de nuestra aplicación, pueden aparecer nuevas fuentes de fragmentación *hardware* o

software que provoquen que nuestra aplicación se tenga que actualizar. Esto sucede en todos los desarrollos, pero especialmente en el desarrollo de aplicaciones para dispositivos móviles, debido a la gran velocidad del cambio en este sector.

Los costes finales se traducen en:

- **Limitación de la vida de las baterías**. Esto significa que, al utilizar muchas de estas capacidades (por ejemplo, el GPS), hacemos que nuestro terminal pierda autonomía, pues se necesita destinar energía a estos periféricos, y lo mismo sucede con el resto de capacidades. En la materia que nos incumbe, el desarrollo de aplicaciones, tenemos que tener esto muy presente a la hora de diseñar y escribir nuestras aplicaciones, ya que puede afectar mucho a su rendimiento.

- **Vulneración de la privacidad**. Nuestros dispositivos móviles contienen cada vez más información personal y confidencial, y requieren acceso a esta información para poder sacarle el mayor partido y conseguir integrarse en el contexto. Por eso, siempre que realicéis una aplicación que requiera acceso a información o acceda a datos de otras fuentes (como las redes sociales), debéis tener el permiso explícito del usuario, y este se debe poder revocar.

- **Necesidades de *hardware***. Por ejemplo, la necesidad de mayor velocidad de transmisión. Si nos encontramos en una zona con poca cobertura para nuestra red de transmisión de datos, puede ocurrir que nuestras aplicaciones no funcionen correctamente.

- **Necesidades de inversión no previstas debido a novedades del mercado**. Si aparece una fuente de fragmentación nueva (por ejemplo, un nuevo dispositivo), debemos invertir en dar soporte a ese nuevo dispositivo. Esta inversión puede suponer no tener que hacer nada o realizar un desarrollo nuevo.

1.2.5. Conclusiones

Sin duda, un aspecto diferenciador de cualquier aplicación móvil debe ser el uso del contexto, de uno o varios modos, pues eso constituye un valor diferenciador con respecto a las aplicaciones de otros soportes.

2. Características de un proyecto de desarrollo para dispositivos móviles

Como hemos visto anteriormente, los desarrollos de aplicaciones sobre dispositivos móviles tienen grandes oportunidades y posibilidades, pero también algunas dificultades añadidas que pueden llegar a ser un riesgo para conseguir que los proyectos sean un éxito.

Por lo tanto, al afrontar un proyecto de desarrollo de *software* para dispositivos móviles, o bien proyectos en los que una parte esté orienta a dispositivos móviles, tendréis que tener un método que, además de soportar la problemática habitual del desarrollo de *software*, se encargue de dar soluciones y de minimizar riesgos, para el caso concreto del desarrollo de *software* móvil.

En este apartado veréis una visión general del tipo de aplicaciones para dispositivos móviles que os podéis encontrar y las compararemos para que podáis elegir la mejor alternativa cuando os enfrentéis a un proyecto. Este punto es muy importante porque condiciona todas las fases del desarrollo. De hecho, podríamos decir que, según el tipo de aplicación, los desarrollos son muy diferentes entre sí.

Después de ver los tipos de aplicaciones que existen, es necesario que conozcáis las opciones disponibles para desarrollar dicha aplicación. En este punto daremos un repaso a las diferentes estrategias, lo cual os ayudará a entender mejor el mundo del desarrollo para dispositivos móviles.

Lo siguiente que veréis en este apartado serán los métodos de desarrollo existentes aplicados a desarrollo de aplicaciones móviles. Veréis las peculiaridades generales del desarrollo móvil y cómo se pueden utilizar estos métodos para solucionar las problemáticas.

A continuación veréis las fases del desarrollo. Haremos hincapié en las diferencias entre las fases de un desarrollo de aplicación normal y las de un desarrollo para dispositivos móviles.

Para que podáis abordar los proyectos, en este apartado veréis una introducción a las estrategias que existen en el desarrollo de aplicaciones móviles. Así conoceréis las diferentes alternativas.

Después os explicaremos detalladamente un método de desarrollo y sus fases, y podremos especial atención a las peculiaridades del desarrollo de aplicaciones móviles.

2.1. Tipos de aplicaciones

Existen muchos tipos de aplicaciones, ya que el tipo de dispositivo que tenemos en mente puede ser muy versátil. Además, hay aplicaciones en las que, seguramente, no pensamos, como las aplicaciones para dispositivos especiales (por ejemplo, un televisor o una consola), pues a pesar de que no son dispositivos móviles, suelen estar programados con las mismas tecnologías y limitaciones.

Las aplicaciones se pueden clasificar en función de la utilidad que queramos darles, o bien según las necesidades del dispositivo y de la complejidad de la propia aplicación.

Cuando os enfrentéis a un proyecto para dispositivos móviles, es importante que conozcáis las opciones que tenéis, sus puntos fuertes y débiles. Con esta información, podréis elegir mejor la aplicación a realizar.

A continuación os mostramos una división según el tipo de desarrollo.

2.1.1. Aplicaciones básicas

La **aplicaciones básicas** son aplicaciones de interacción básica con el dispositivo que únicamente envían o reciben información puntual del usuario.

La aplicaciones básicas se pueden gestionar simplemente con el envío de mensajes de texto (SMS o MMS). Estas aplicaciones existen desde hace mucho tiempo y, aunque han tenido gran aceptación y uso, actualmente están comenzando a dejar paso a aplicaciones más complejas.

Las aplicaciones básicas tienen las siguientes ventajas:

- simplicidad
- facilidad de venta
- gran cantidad de usuarios potenciales

También presentan algunas desventajas:

- poca o casi nula capacidad de procesamiento del contexto
- muy baja complejidad de las aplicaciones realizadas
- limitaciones impuestas por la tecnología sobre los diseños de las aplicaciones (ciento sesenta caracteres de texto)

Ejemplo de aplicación básica

Una ejemplo de aplicación es una aplicación para conocer cuál es el tiempo de llegada de un autobús (se envía un SMS a un número concreto y se recibe una respuesta).

2.1.2. Webs móviles

Las **webs móviles** son aquellas webs que ya existen actualmente y que son adaptadas específicamente para ser visualizadas en los dispositivos móviles. Adaptan la estructura de la información a las capacidades del dispositivo, de manera que no saturan a los usuarios y se pueden usar correctamente desde estos dispositivos.

Lenguaje de marcas

Lenguaje de marcado o lenguaje de marcas (*markup language*) es el lenguaje que estructura la información mediante marcas o etiquetas (*tags*).

Dependiendo de los dispositivos a los que queramos llegar, deberemos adoptar un lenguaje de marcado u otro, pues hay dispositivos que solo soportan un tipo de marcas. En los dispositivos de tipo *smartphone* actuales, podemos visualizar una web que no esté adaptada a dispositivos móviles, pero en ese caso no estaríamos hablando de una aplicación para dispositivos móviles, sino de una capacidad del dispositivo.

Este tipo de aplicaciones son aplicaciones básicas que, por lo general, no usan objetos dinámicos como Javascript. Por tanto, no tienen todo el potencial de un navegador web de sobremesa. Se utilizan estándares web como, por ejemplo, XHTML, WML, XHTML-MP, c-html, etc. y, en general, versiones previas a la nueva versión del estándar HTML: HTML 5. Están pensadas para dar soporte a dispositivos de media y baja gama.

Las ventajas de las webs móviles son las siguientes:

- **Fácil implementación, testeo y actualización**. Incluso se puede realizar gran parte del desarrollo sin necesidad de utilizar dispositivos móviles ni emuladores, hasta llegar a las fases finales del desarrollo.
- **Lenguaje conocido y estándar**. Los lenguajes de marcas son muy conocidos hoy en día por la mayoría de los desarrolladores, y en la mayoría de los casos se trata de subconjuntos de lenguajes conocidos.
- Pueden soportar múltiples dispositivos con un único código fuente. Para soportar fragmentación entre dispositivos, es necesario utilizar técnicas especiales (como el WURLF).

WML

Casi todos los dispositivos móviles actuales tienen soporte para algún tipo de lenguaje de marcado. Inicialmente era WML (*wireless markup language*), pero también se utilizan otros.

WURLF

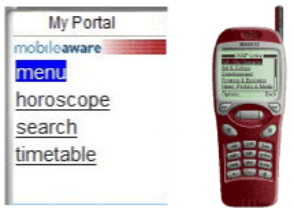
WURLF es un repositorio de información que identifica, a partir de la metainformación de una petición web de un dispositivo móvil, sus capacidades y limitaciones. Para saber más sobre WURLF, podéis visitar el siguiente sitio web:
http://wurfl.sourceforge.net/

Los inconvenientes de las webs móviles son los siguientes:

- Es difícil soportar múltiples dispositivos, así como conseguir la misma experiencia de usuario con varios tipos de navegadores.
- Ofrecen grandes limitaciones a la hora de realizar programas, tanto de proceso como de acceso a la información del dispositivo y del usuario. Por tanto, es difícil conseguir aplicaciones contextualizadas.
- En muchos casos está pensado para ser visualizado con conexiones lentas, pero dichas conexiones pueden ser demasiado lentas y provocar una experiencia de usuario deficiente.
- En la actualidad, la mayoría de los dispositivos nuevos están incorporando estándares más nuevos (como HTML 5), por lo que no se está trabajando en mejorar estos estándares.
- El número de dispositivos que solo pueden ver una página web con este tipo de lenguajes de marcas está disminuyendo.

Ejemplos de web móviles

Cualquier web pública con información para el contribuyente es un buen ejemplo de web móvil. Por ejemplo, http://wap.bcn.cat.



2.1.3. Aplicaciones web sobre móviles

Las aplicaciones web sobre móviles son aplicaciones que no necesitan ser instaladas en el dispositivo para poder ejecutase. Están basadas en tecnologías HTML, CSS y Javascript, y que se ejecutan en un navegador. A diferencia de las web móviles, cuyo objetivo básico es mostrar información, estas aplicaciones tienen como objetivo interaccionar con el dispositivo y con el usuario. De esta manera, se le saca un mayor partido a la contextualización.

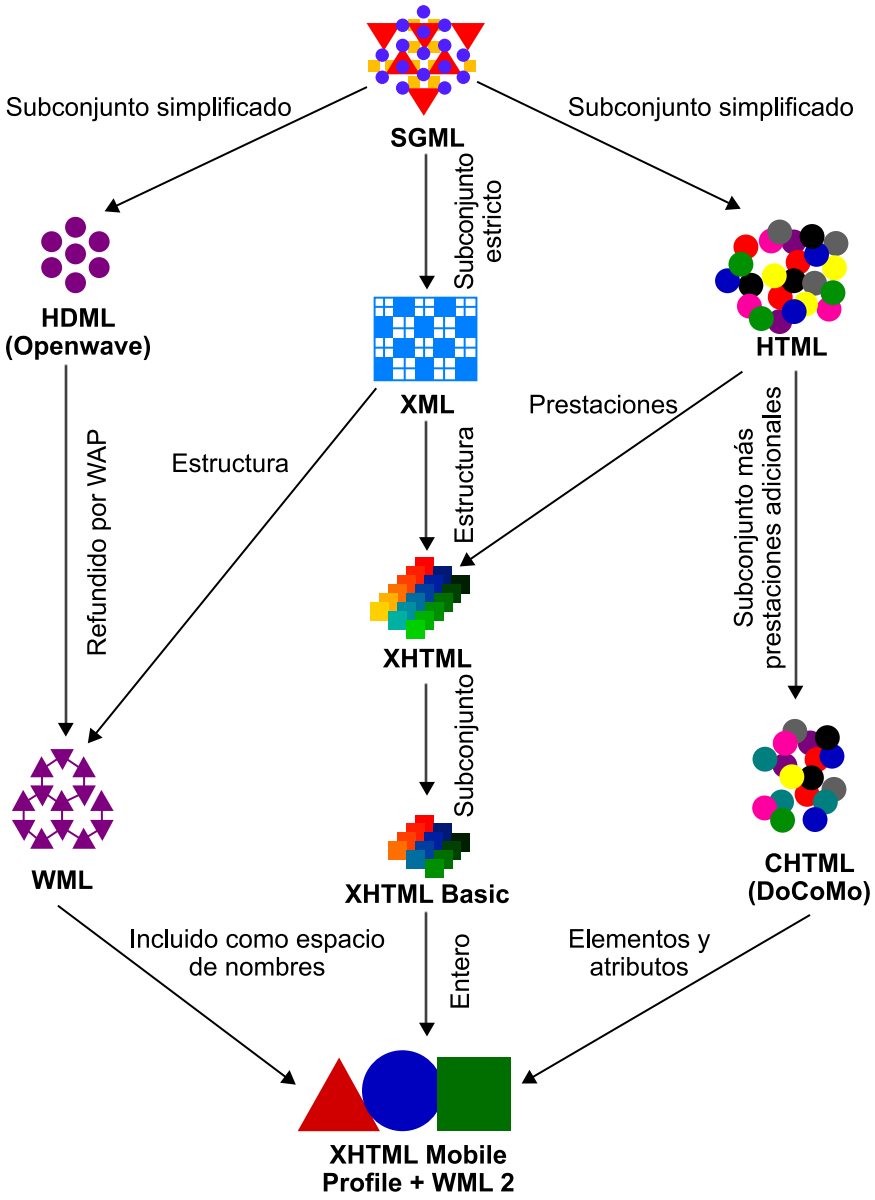
Son aplicaciones especialmente diseñadas para trabajar en el móvil y que intentan aprovechar al máximo sus posibilidades (en ocasiones lo hacen accediendo a datos del contexto: posición geográfica, datos guardados, etc.).

Este tipo de aplicaciones han existido desde hace tiempo, pero con la aparición de navegadores más potentes (por ejemplo, los basados en *WebKit*, como los navegadores de Android, iOs, Nokia S60 series, etc.) se ha pasado básicamente del XHTML a soportar HTML y, sobre todo, HTML 5. Gran parte de este avance se consigue con HTML 5 y CSS3, que nos ofrece, entre otras cosas, la posibilidad de tener aplicaciones muy parecidas y con mucho potencial con respecto a las aplicaciones nativas.

WebKit

WebKit es una plataforma de aplicaciones que funciona como base de varios navegadores (como Google Chrome, Safari, etc.). Están en el motor de renderizado XHTML del navegador Konqueror de KDE.

A continuación presentamos la evolución de los lenguajes de marcado, partiendo del original SGML. Veremos que, finalmente, se llega al que hasta ahora era el lenguaje más usado en el desarrollo de webs para dispositivos móviles: XHTML-MP (Mobile profile). Como se puede observar han existido muchas alternativas, y algunas que no hemos visto, lo cual provocaba una gran dificultad para desarrollar dichas webs:



Las ventajas de las aplicaciones web sobre móviles son las siguientes:

- Posibilidad de acceso a mucha información del dispositivo para realizar aplicaciones relativamente complejas.
- Desarrollo, distribución y pruebas sencillas.
- Convergencia entre aplicaciones de sobremesa y de dispositivos móviles, lo cual tiene muchas implicaciones, como, por ejemplo, que los desarrolladores solo tienen que conocer una tecnología.
- Uso de estándares de la web (claramente definidos).
- Ampliamente soportado por la industria, de manera que la mayoría de los nuevos dispositivos tienen soporte para este tipo de aplicaciones.

Por el contrario, también tienen los siguientes inconvenientes:

- Se necesita un navegador que pueda dar soporte a este tipo de tecnología. Aunque la mayoría de los nuevos dispositivos lo pueden dar, los antiguos no.
- Su rendimiento es menor con respecto a las aplicaciones nativas, pues se ejecuta todo mediante el Javascript del navegador, cuya potencia es limitada.
- Imposibilidad de acceder a todas las posibilidades del dispositivo. No se puede acceder al *hardware* ni a muchos periféricos (como sensores o cámaras). No se puede acceder a mucha de la información del usuario (contactos, sus citas, etc.).

Ejemplo de aplicación móvil

Actualmente existen muchos ejemplos de aplicaciones web realizadas para ser ejecutadas en sobremesa, que han sido rápidamente portadas a los dispositivos móviles, como son mobile.twitter.com, facebook.com, maps.google.com, etc.

Introducción a HTML 5

HTML5 es un gran paso adelante para mejorar los estándares web y conseguir desarrollos para dispositivos móviles más sostenibles. Es por ello que vamos a ver una pequeña introducción a dicho estándar.

Antes de empezar a definirse HTML 5, el consorcio W3C[3] ya estaba trabajando en un estándar para sustituir la gran variedad de estándares web que existían: XHTML 2.0.

(3)World Wide Web Consorcium (W3C) es un consorcio internacional que produce recomendaciones, que no estándares, para la World Wide Web (WWW). El creador, y actual líder, de este consorcio es Tim Bernes-Lee, considerado el creador de la web y concretamente creador de la URL, HTTP y HTML.

Logotipo oficial del estándar HTML 5

XHTML 2.0, al igual que su predecesor XHTML 1.0, contenía elementos XML por lo que era incompatible con HTML 4. Debido a la lentitud del W3C para generar nuevos estándares (HTML 4.01 fue aprobado en 1999), el WHATWG[4] decide tomar como referencia HTML 5 y abandonar la estructura HTML en favor de XML. Finalmente, en 2006, W3C decide participar en la elaboración de HTML 5. Se desarrolla en paralelo XHTML 2.0 y HTML 5. De esta manera, se ha conseguido el soporte de los principales navegadores y se sigue siendo compatible con las versiones anteriores.

(4)Web Hypertex Application Techonolgoy Working Group. Este grupo lo forman las empresas responsables de los principales navegadores web: Apple, Opera, Mozilla, Microsoft y Google.

Con XHTML 5 se quiere conseguir:

- Añadir novedades al HTML y CSS actuales.
- Dar soporte a todos los navegadores, incluyendo los navegadores móviles, evitando una ruptura total con las versiones actuales del estándar.
- Aprovechar las características de los dispositivos actuales, como pueden ser aceleración gráfica o múltiples procesadores.
- Tener mayor libertad de desarrollo, evitando al máximo la necesidad de extensiones propietarias que existen actualmente.

HTML 5 se engloba dentro del estándar Web Applications 1.0, que incorpora algunos estándares más. Este estándar da soporte, por ejemplo, a las versiones XHTML y HTML del estándar y define las API para poder realizar las acciones dinámicas (con ECMAScript[5]).

(5)Estándar sobre el que está basado el popular lenguaje Javascript.

- Visualización de información e interacción con dicha información más potente, a través de los lienzos (*canvas*) y los formularios (*web forms*). En los formularios se tiene validación nativa del navegador.

- Modo sin conexión. Guardado de la información del usuario en el dispositivo físico, mediante el almacenamiento local para por ejemplo soportar falta de conectividad.

- Almacenamiento de datos en el navegador como una base de datos.

- Acceso a datos como la posición geográfica.

- Renderización de objetos gráficos aprovechando la potencia de las tarjetas gráficas de los dispositivos, para así mejorar su rendimiento (gracias en parte al soporte de SVG[6]).

(6)SVG (*scalable vector graphics*), se trata de especificación para definir gráficos bidimensionales, tanto estáticos como animados. Desde 2001 es una recomendación de W3C. SVG está integrado en los estándares web de manera que se puede definir con etiquetas estándar HTML los objetos gráficos.

- Soporte para video y audio (etiquetas <video> y <audio>) sin necesidad de extensiones propietarias.

- Nuevas etiquetas semánticas, como *section, article, header, nav*, etc.

- API para coger y arrastrar (*Drag & drop*).

- Trabajos pesados en segundo plano (*Web Workers*).

Enlaces de interés

En la siguiente dirección web encontraréis diversos ejemplos de las posibilidades de HTML 5:

http://html5rocks.com

Y en esta dirección encontraréis el soporte actual de los navegadores para los diferentes estándares:

http://html5readiness.com/

Paralelamente a la definición de los estándares HTML, se han ido actualizando los estándares correspondientes al estilo de las páginas, en concreto la nueva versión es CSS3. Esta versión está dividida en módulos, que han ido siendo aprobados como especificaciones en momentos diferentes. Algunos de los

más importantes son CSS Selectors, CSS Colors, backgrounds & borders, CSS Multi-column layout, CSS Namespace, Media Queries, CSS Speech, CSS Animations, CSS 3D Transformations, etc. Todos estos estándares también contribuyen a conseguir que la experiencia de los usuarios de webs para dispositivos móviles sea más completa y su programación pueda ser más estructurada.

2.1.4. Aplicaciones web móviles nativas

Existe un tipo de aplicaciones, llamadas aplicaciones web móviles nativas, que no son aplicaciones web propiamente ni tampoco nativas. Se ejecutan con un navegador o, mejor dicho, con un componente nativo que delega en un navegador, y tienen algunas de las ventajas de las aplicaciones nativas.

Este tipo de aplicaciones pueden ser instaladas en el dispositivo, con lo que pueden utilizar los canales estándares de distribución de aplicaciones nativas, o bien incorporarse como accesos directos (como el resto de aplicaciones). Sin embargo, estas aplicaciones no tienen la potencia de las aplicaciones nativas, sino que simplemente ejecutan código en un navegador embebido (generalmente con HTML 5).

Las ventajas de las aplicaciones web móviles nativas son las siguientes:

- Todos los puntos a favor de las aplicaciones web móviles.
- Se pueden considerar, en lo que respecta a la instalación y la distribución, como aplicaciones nativas.

En cambio, sus inconvenientes son los siguientes:

- La mayoría de los inconvenientes de las aplicaciones web móviles, a excepción de la instalación en el cliente.

- La experiencia del usuario es, en ocasiones, contradictoria, pues a pesar de tratarse de una aplicación nativa, requiere de conexión a Internet para poder trabajar y funciona según los tiempos de respuesta del navegador.

Ejemplo de aplicaciones móviles nativas

Aplicaciones de acceso a información confidencial, como son las entidades financieras "en línea", que distribuyen la aplicación, pero por su naturaleza no es recomendable que haya datos en el dispositivo físico.

Existen también los llamados *mobile widgets*, que se pueden englobar como un subtipo de aplicación web móvil. Ejecutan aplicaciones web como aplicaciones nativas, pero a modo de *widget*, es decir, que se pueden ejecutar continuamente o están integradas en la plataforma destino. Un ejemplo de este tipo de aplicaciones sería un buscador de resultados en Wikipedia, o un traductor. Tienen la mayoría de las ventajas de las aplicaciones móviles, pero con limitaciones en lo que respecta a la dimensión, ya que suelen ser ejecutadas en una parte limitada de la pantalla.

Ejemplos de *mobile widgets*

Algunos ejemplos de los *mobile widgets* son WRT (Web Runtime Widgets de Sybiuan), Blackberry Widget SDK, etc.

2.1.5. Aplicaciones nativas

Las aplicaciones nativas son las aplicaciones propias de cada plataforma. Deben ser desarrolladas pensando en la plataforma concreta. No existe ningún tipo de estandarización, ni en las capacidades ni en los entornos de desarrollo, por lo que los desarrollos que pretenden soportar plataformas diferentes suelen necesitar un esfuerzo extra.

Estas aplicaciones son las que mayor potencial tienen, pues aprovechan al máximo los dispositivos y consiguen, de esa manera, una mejor experiencia de usuario.

Existen muchas plataformas, una gran parte de ella ligadas al tipo de dispositivo, aunque también hay plataformas, como Android, que existen para diferentes tipos de dispositivos.

Algunas de las más conocidas son iOs, Android, Blackberry, bada, Java Me, Windows Phone (antes Windows Mobile o Windows Ce), Symbian, Web OS, Brew, etc. Todas ellas tienen diferentes tipos de dispositivos con una base común entre ellos.

Las ventajas de las aplicaciones nativas son las siguientes:

- Acceso total al contexto, con todas las posibilidades que eso conlleva. Consigue las mejores experiencias de usuario.
- Posibilidad de gestión de interrupciones en la aplicación o en las capacidades del dispositivo. Desde saber si tenemos conexión de datos o conexión de localización hasta tener información sobre la batería.
- Son relativamente fáciles de desarrollar si solo se contempla una plataforma.
- Se pueden distribuir por los canales conocidos de aplicaciones que permita la plataforma, con lo que se pueden vender más fácilmente.
- Todas las novedades llegan primero a este tipo de aplicaciones, pues es en este tipo de aplicaciones donde se prueban.

En cambio, tienen sus inconvenientes:

- Portar aplicaciones es costoso. En el caso de querer realizar una aplicación para más de una plataforma, se complica el desarrollo, debido a los problemas de la fragmentación.
- Dependiendo de la plataforma elegida, puede haber fragmentación dentro de cada plataforma, debido a los diferentes tipos de dispositivos o versiones de la plataforma.
- No existe un estándar, por lo que cada plataforma ofrecerá sus peculiaridades.
- Normalmente, para desarrollar, distribuir o probar estas aplicaciones en dispositivos reales, es necesario tener una licencia de pago, dependiendo de la plataforma.
- Las ganancias por estas aplicaciones suelen repartirse entre el creador de la aplicación y la plataforma de distribución.

Ejemplos de aplicaciones nativas

Algunos ejemplos de aplicaciones nativas son los siguientes:

- Aplicaciones para la gestión de la agenda o para encontrar amigos de la agenda con su posición geográfica.

- Alarmas, aplicaciones correo electrónico, etc.

- Aplicaciones sociales, como las aplicaciones de las redes sociales y otras aplicaciones que permiten utilizar el acceso a estas aplicaciones, como los agregadores de redes sociales.

Las siguientes imágenes representan aplicaciones nativas que aprovechan las funcionalidades del dispositivo, como, por ejemplo, aplicaciones de realidad aumentada o juegos.

2.2. Estrategias de desarrollo de aplicaciones móviles

A la hora de emprender un proyecto móvil, es importante que conozcáis las alternativas. En ocasiones es inviable conocerlas todas y, normalmente, resulta muy difícil conocerlas todas en detalle, pero sí es muy recomendable que tengáis una visión general de las opciones y alternativas del mercado. En este módulo os daremos una visión general y ampliaremos algunos aspectos.

Dada la gran fragmentación de plataformas y tipos de aplicaciones que existen, lo primero que tenéis que hacer es intentar minimizar al máximo el abanico de posibilidades. Para ello, veréis los tipos de aplicaciones a los que os vais a enfrentar.

Una vez tengáis claro el tipo de aplicación que queréis hacer, deberéis elegir el tipo de estrategia a utilizar para llevarla al dispositivo. Hay diferentes tipos de estrategias, y dentro de estas estrategias existen muchas alternativas concretas. Veréis también los puntos fuertes y débiles de cada alternativa.

Así, podremos ver las diferentes estrategias para desarrollar en nuestro móvil.

2.2.1. Desarrollos web

Dentro de este subapartado englobamos todas las aplicaciones que están basadas en lenguajes de marcas, lo cual añade la facilidad de poder programar y probar sin necesidad de un emulador o un dispositivo real. A estas aplicaciones se accede directamente mediante la red. Quedan excluidas las aplicaciones que requieren de un preproceso para poder ser distribuidas.

En este punto también se incluyen aplicaciones web basadas en aplicaciones propietarias, como, por ejemplo, Flash y Flash lite. Estas aplicaciones tienen el mismo modelo de desarrollo.

1) Prerequisitos. En general, se puede utilizar cualquier entorno de desarrollo conocido. En el caso de las aplicaciones de tipo *widget* o entornos de ejecución propietarios, los fabricantes suelen dar soporte a estos desarrollos mediante entornos de desarrollo específicos.

2) Fragmentación. La fragmentación en este tipo de aplicaciones existe, aunque suele ser menor que en el resto. Para poder adaptar nuestra aplicación a las capacidades del dispositivo, y dado que estamos en una arquitectura de navegador web, la mejor opción es intentar reconocer el dispositivo cuando se recibe la primera petición. También se puede intentar mostrar de manera diferente la información en el navegador, pero normalmente las capacidades del navegador hacen inviable esta opción.

Una vez que sabemos el dispositivo, tenemos que conocer sus capacidades. Existen varias maneras de adaptar nuestra aplicación a dicho dispositivo:

a) Teniendo el conocimiento (por ejemplo, el proyecto WURLF, que consiste en una base de datos de todos los dispositivos y sus capacidades).

b) Utilizando servidores que incorporen ese conocimiento y lo automaticen, de manera que lleguen a realizar transformaciones automatizadas. Esta opción tiene el inconveniente de que las últimas versiones no suelen estar soportadas.

Ejemplo

A continuación exponemos ejemplo de código de validación de las capacidades de un dispositivo mediante la utilización de la librería de WURLF para reconocer el dispositivo y sus capacidades.

```
public class HelloWorld extends HttpServlet {
  ...
  protected void processRequest(
     HttpServletRequest request,
     HttpServletResponse response)
     throws ServletException, IOException {

        WURFLHolder wurflHolder = (WURFLHolder) getServletContext()
                    .getAttribute("net.sourceforge.wurfl.core.WURFLHolder");

        WURFLManager wurfl = wurflHolder.getWURFLManager();

        Device device = wurfl.getDeviceForRequest(request);

        MarkUp markUp = device.getMarkUp();
        request.setAttribute("markUp", markUp);
        request.setAttribute("device", device);
     ...
  }
}
```

3) Pruebas. Las pruebas se pueden empezar con navegadores de escritorio que soporten HTML 5 o el correspondiente lenguaje de marcado, pero después se deberán hacer pruebas reales, o bien con emuladores. Es muy recomendable que se realicen las pruebas con dispositivos reales. Hay que prestar especial atención a aquellas partes de los estándares que no son soportadas por todos los dispositivos.

En el mundo del desarrollo web existen muchas herramientas para probar las aplicaciones web. La gran mayoría de ellas se puede aplicar a este tipo de aplicaciones.

4) Distribución. La distribución es lo más sencillo, ya que es igual que la distribución de una aplicación web cualquiera. Solo cuando haya cambios incompatibles con los datos guardados "fuera de línea" de una aplicación, habrá que realizar acciones informativas para que el usuario actualice dichos datos.

2.2.2. Entornos de desarrollo nativos

Como ya hemos visto, las aplicaciones nativas son las que ofrecen una mejor experiencia de usuario. Son aquellas que están especialmente diseñadas e implementadas para el contexto de ejecución (plataforma o dispositivo) en el que van a ejecutarse, y pueden sacarle partido a todas las capacidades de dichos dispositivos. En ocasiones, también están sujetas a normas específicas de los fabricantes de dispositivos o responsables de las plataformas.

1) Prerrequisitos. Para poder desarrollar una aplicación nativa, generalmente se necesita el entorno de desarrollo o IDE de cada plataforma. Por ejemplo, para Android necesitamos su SDK, y es recomendable usar Eclipse y añadirle algunos *plugins*. En cambio, en el caso de iOS, necesitamos xCode; para Blackberry *apps* necesitamos también su SDK; para Windows Phone necesitamos Miscrosoft Visual Studio, etc. Estos IDE pueden tener una licencia de pago, la cual dependerá de cada plataforma.

IDE

Integrated development environment (IDE) es un entorno de desarrollo que incorpora todas o casi todas las herramientas de desarrollo necesarias, desde herramientas de modelado o diseño hasta herramientas de *debugging*.

Algunos de estos IDE no son multiplataforma, por lo que requieren de un equipo de desarrollo específico (como es el caso de Windows Phone o iOs).

Estos IDE suelen proporcionar todo lo necesario para cubrir el desarrollo completo de la aplicación, de manera que incluyen los emuladores necesarios para probar nuestra aplicación mientras la desarrollamos.

2) Implementación. Todas las implementaciones son distintas. Cada sistema utiliza su propio método y sus propios patrones, pero hay algunos puntos comunes:

a) Existe un emulador con el que podemos probar nuestras aplicaciones. Sin embargo, en ocasiones el emulador no permite emular todas las acciones de usuarios o la emulación no es lo suficientemente ágil, por lo que necesitamos un dispositivo real.
b) Separación de presentación y lógica, de manera que aprovechemos al máximo los componentes.
c) Posibilidad de "debugar" nuestra aplicación para poder tener mayor control.
d) Generalmente existen herramientas que facilitan la construcción de las interfaces gráficas o UI (*user interface*).

3) Pruebas. Para poder hacer pruebas, cada IDE tiene sus herramientas, desde las típicas tecnologías de pruebas unitarias hasta sistemas más complejos, como el *monkeyrunner* de Android. Para realizar pruebas de estrés de las aplicaciones, también existen herramientas para hacer pruebas de aceptación contra la UI, que utilizan lenguajes de alto nivel, como es el caso de UIAutomation sobre iOs.

Monkeyrunner

Las herramientas *monkeyrunner* y *monkey* sirven para lanzar muchos eventos de usuario sobre el emulador. Son eventos aletorios (por eso se empela la metáfora de un mono).

UIAutomation

UIAutomation sirve para definir en Javascript las acciones de usuario que finalmente se ejecutan en el dispositivo o en el emulador.

Sin duda, las posibles pruebas que se pueden realizar sobre las aplicaciones nativas son mucho más extensas y están más controladas que aquellas que se puedan realizar en otro tipo de aplicación, ya que se tienen las herramientas propias de la plataforma.

4) Firma y distribución. Para poder distribuir la aplicación o, incluso, ejecutarla en un terminal para hacer pruebas, puede ser necesario firmar dicha aplicación con un certificado digital que nos identifique como desarrolladores.

Si la distribución se realiza mediante terceros, como sucede en los mercados de aplicaciones (*market places*), es incluso más necesario para poder acreditar que tenemos el derecho de publicar aplicaciones, así como para hacernos responsables de dichas aplicaciones.

Según la plataforma, los modelos de distribución son simplemente sistemas de descarga, bien mediante sistemas *OTA* (*over the air*), bien mediante los mercados de aplicaciones. Este último sistema está teniendo una gran acogida, ya que permite vender fácilmente la aplicación y llegar a un gran número de usuarios potenciales de forma rápida, aunque podemos encontrarnos con unas fuertes restricciones a la hora de conseguir publicar nuestra aplicación.

2.2.3. Entorno de desarrollo multiplataforma

Como habéis visto, las aplicaciones nativas son muy potentes, pero a la vez requieren de un esfuerzo de desarrollo para soportar solamente una plataforma, y así con cada una de las plataformas que queramos soportar. Para poder soportar todas las plataformas, necesitaríamos saber muchos lenguajes, ya que habría que portar dichas aplicaciones entre plataformas. En concreto, en la actualidad existen al menos cinco lenguajes diferentes, que son necesarios para poder realizar aplicaciones sobre las plataformas más actuales: C, C++, Java, C#, Javascript, Objective-C, además de los diferentes IDE necesarios y sus correspondientes librerías específicas.

En cambio, las aplicaciones web nos permiten llegar a muchas plataformas con un mismo código sin necesidad de portar el código, pero sin poder llevar al usuario la misma experiencia que consigue con las aplicaciones nativas.

Por lo tanto, si existiera la posibilidad de realizar aplicaciones nativas desde una misma línea de código, tendríamos lo mejor de ambas aproximaciones. Aquí es donde entran en juego las estrategias de aplicaciones multiplataforma o *cross-platform*, también conocidas como aplicaciones híbridas.

Desde hace tiempo se está persiguiendo esta "multiplataformidad" con intentos como JavaMe, que no han dado los resultados esperados. La llegada de HTML 5 y la explosión de nuevos *smartphones* han creado nuevas alternativas, que se deben tener en cuenta. Muchas de estas alternativas aprovechan HTML 5 como base y construyen a su alrededor maneras de acceder a las capacidades que HTML 5 no da de partida, prácticamente siempre mediante objetos Javascript. Usar elementos 100% estándar (como HTML 5, CCS y Javascript) ofrece un gran punto a favor, pues se trata de tecnologías muy conocidas.

Las hay que simplemente se quedan en un *wrapper* de la aplicación HTML 5 y añaden estos puntos de acceso. En cambio, otras realizan preprocesamiento para acabar generando aplicaciones 100% nativas. Hay otras alternativas que proporcionan su propia arquitectura y sus propios lenguajes, y también mediante un sistema de compilación o ejecución vía máquina virtual consiguen tener aplicaciones nativas.

Hay aspectos que estas aproximaciones no van a poder evitar fácilmente (a no ser que tengan código condicional específico para cada plataforma). Son los siguientes:

- **Pérdida de controles específicos de una plataforma**: Si tenemos un control de la UI o una funcionalidad concreta que solo existe en una plataforma, no podremos generar de manera única, por nuestro desarrollo multiplataforma.

- **Integración en el escritorio del dispositivo**: Según la plataforma, las posibilidades de añadir elementos en el escritorio de cada usuario varían. Por ejemplo, en Android o Symbian se pueden crear *widgets* potentes para mejorar la el uso de nuestra aplicación, mientras que en Windows Phone solo es posible añadir iconos de la misma.

- **Gestión de la multitarea**: Debido a que se trata de conceptos de bajo nivel de cada plataforma, cada una la trata de manera diferente, con restricciones diferentes, por lo que no será fácil hacer código común para todas sin perder mucha potencia.

- **Consumo de la batería**: Estas aproximaciones requieren de una capa de abstracción sobre nuestro dispositivo, que provoca problemas como la multitarea. De la misma forma, el control sobre el consumo de batería se hace más difícil cuando no se tienen las capacidades concretas de la plata-

forma. También afecta el hecho de no tener el control sobre la multitarea, ya que esta es una de las maneras de ahorrar las baterías.

- **Servicios de mensajería asíncrona o *push services***: Sirven para implementar elementos como la mensajería instantánea, pero debido a que cada plataforma los implementa de una manera, se hace complicado atacarlos conjuntamente.

A continuación exponemos algunos de los ejemplos que existen:

- **PhoneGap**: Son aplicaciones realizadas en HTML 5 con objetos Javascript que permiten el acceso, mediante enlaces, a las funciones nativas para las capacidades que HTML 5 no ofrece. Las aplicaciones se ejecutan sobre un componente que contiene un navegador.

- **Rhodes**: Son aplicaciones escritas en Ruby que utilizan el patrón de diseño MVC[7] y que se ejecutan en máquinas virtuales específicas de Ruby para cada plataforma. Por tanto, son aplicaciones nativas. Incluyen sistemas para sincronizar datos de manera sencilla y conseguir, de esa manera, el cambio de "en línea" a "fuera de línea" sin mucho coste.

[(7)]MVC (modelo vista controlador)

- **Appcelerator**: Son aplicaciones escritas en HTML 5 que son compiladas en aplicaciones nativas. También son capaces de generar aplicaciones de sobremesa clásicas, ejecutables sobre Mac, Linux o Windows.

- **Wacapps**: Son aplicaciones escritas en HTML 5 con soporte para la distribución por parte de las operadoras.

- **Flash**: Son aplicaciones que corren sobre un *player* propietario. Si existe el *player* correspondiente, no necesitan ser portadas. Se escriben de igual manera que las aplicaciones de sobremesa y tienen las mismas restricciones.

- **Java ME**: Son aplicaciones escritas en Java, con todas las peculiaridades de los perfiles J2ME, y se ejecutan mediante una máquina virtual, con sus correspondientes restricciones. Actualmente están en casi el ochenta por ciento de los dispositivos móviles del mercado.

- **Unity3D**: Entorno para desarrollar juegos nativos o web para cada plataforma, como Play Station, Wii, PC, Mac, etc. También da soporte para las Android e Iphone.

Estas son solo algunas de las existentes, pero hay muchas más y, actualmente, siguen saliendo alternativas.

Estos desarrollos nos ofrecen beneficios de los desarrollos previos:

- Solo una línea de desarrollo para varias plataformas.

- Aplicaciones que podemos instalar en nuestros dispositivos, que tienen la posibilidad de ser distribuidas en los mercados de aplicaciones.
- Dependiendo del caso, el resultado son aplicaciones nativas que aprovechan en gran medida el potencial del dispositivo y ofrecen un diseño y experiencia de usuario idéntica a las aplicaciones desarrolladas sólo para una plataforma.

Sin embargo, también existen aspectos negativos:

- En diferentes grados, tienen acceso parcial a los recursos del dispositivo. Esto significa que, según el grado, no pueden interaccionar con algunas capacidades y, en mayor o menor medida, serán aplicaciones menos eficientes que las aplicaciones nativas.

- Para poder soportar varios dispositivos, suelen tener que hacer el mínimo común denominador de las capacidades. Así, si una plataforma aporta una nueva funcionalidad (por ejemplo, un sistema de comunicación que el resto no tenga), no se podrá implementar sin añadir código específico.

- Soporte parcial. No tienen soporte absoluto para todas las plataformas y versiones.

- Las nuevas funcionalidades tardan en ser soportadas. Dado que las novedades en las plataformas aparecen en un ritmo muy elevado y en cortos periodos de tiempo, si se utilizan estos entornos para el desarrollo, es necesario esperar un tiempo para poder utilizar dichas herramientas. En el mejor de los casos, si se trata de código libre o ampliable, podemos realizar una implementación para tener acceso a esa funcionalidad.

- Se requiere pagar la licencia, si la tiene, para el entorno de desarrollo, además de la licencia propia de cada plataforma para poder distribuir la aplicación.

Como veis, esta es sin duda una alternativa a tener en cuenta para decidir qué estrategia de desarrollo queréis para vuestro proyecto web, pero no siempre será la mejor.

1) Prerrequisitos. En general, para conseguir las aplicaciones nativas, cada uno de los entornos proporciona su entorno de desarrollo completo.

En muchos casos, las aplicaciones se prueban en los emuladores de las plataformas nativas, de manera que hay que instalar el IDE propio del entorno de desarrollo y los emuladores o, en ocasiones, los SKD.

SDK

SDK o kit de desarrollo de *software* es el conjunto de herramientas necesarias para realizar el desarrollo de aplicaciones en una plataforma.

2) Implementación. Durante la implementación, variará mucho en función de cada plataforma, pues hay algunas que podrán aprovechar todas las herramientas de desarrollo, otras que no lo necesitarán en exceso y algunas en las que el desarrollo será más difícil.

En general, las herramientas facilitan el proceso de desarrollo, aunque sin llegar al nivel de las herramientas de desarrollo nativas.

3) Firma y distribución. A veces la distribución se puede realizar directamente mediante los IDE de las plataformas de desarrollo, pero en la mayoría de los casos es necesario hacerla mediante los canales habituales para las aplicaciones nativas.

2.3. Métodos aplicados al desarrollo de aplicaciones móviles

En el mundo del desarrollo de *software* existen muchos métodos de desarrollo, cada uno con sus puntos fuertes y sus puntos débiles. En el caso del desarrollo de aplicaciones móviles sucede lo mismo, y cuando os planteéis qué método elegir deberéis saber escoger en función de vuestras necesidades.

Algunos de los métodos más conocidos son los siguientes:

- modelo *waterfall*
- desarrollo rápido de aplicaciones
- desarrollo ágil (cualquiera de sus variantes)
- Mobile-D

Una de las características importantes de la gran mayoría de los desarrollos móviles es su corta duración. Esto se debe a factores como la gran competencia en el sector, los cambios en el mismo con la aparición, casi constante, de novedades tanto *software* como *hardware*, el hecho de que muchas aplicaciones nacen con un desarrollo precoz en forma de prototipo (y van evolucionado después) o incluso la simplicidad de las aplicaciones, que no requieren grandes desarrollos. Esta suele ser, salvo algunas excepciones, la norma de los desarrollos de aplicaciones para dispositivos móviles.

"It can take up to nine months to deploy an entertainment (mobile) application, but that's the duration of a cell phone in this market."

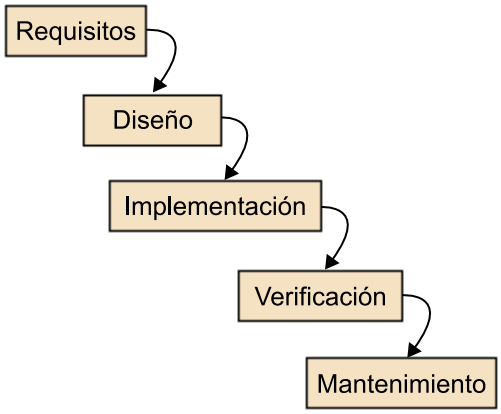
Craig Hayman (IBM)

2.3.1. Modelo *waterfall*

El modelo *waterfall* es el modelo más estático y predictivo. Es aplicable en proyectos en los que los requisitos están fijados y no van a cambiar durante el ciclo de vida del desarrollo. Esta aproximación divide el proyecto en fases

estancas totalmente secuenciales. En este modelo, el desarrollo se interpreta como el agua que va cayendo de un estanque al siguiente. Se le da mucho énfasis a la planificación, a los tiempos, a las fechas límite y al presupuesto.

Ejemplo de fases de un proyecto de *software* en un modelo *waterfall*



En el contexto del desarrollo de aplicaciones móviles, el modelo *waterfall* puede ser aplicable a proyectos realmente controlados y previsibles, en los que no hay mucha incertidumbre por lo que se desea hacer y para los que no son importantes los cambios constantes en la industria.

2.3.2. Desarrollo rápido de aplicaciones

El desarrollo rápido de aplicaciones es un método de desarrollo iterativo cuyo objetivo es conseguir prototipos lo antes posible para mejorarlos después, poco a poco. Se suele priorizar la implementación sobre la planificación, y se utilizan muchos patrones de diseño conocidos para poder adaptarse de la mejor manera a cambios en los requerimientos.

El desarrollo rápido de aplicaciones es un método muy útil para el desarrollo de proyectos realmente urgentes con tiempos de entrega muy cortos.

2.3.3. Desarrollo ágil

El desarrollo ágil es un modelo de desarrollo basado en iteraciones, donde en cada iteración se realizan todas las fases del ciclo de desarrollo.

Manifiesto ágil

El manifiesto ágil fue publicado en el 2001 por diecisiete desarrolladores de *software*, quienes representaban entonces los métodos de desarrollo más populares, que pasarían a conocerse como ágiles (Extreme Programming, Crystal Clear, DSDM o ASD, entre otros). El manifiesto define doce principios y cuatro valores éticos para los desarrolladores. Podéis consultarlo en http://agilemanifesto.org.

El desarrollo ágil se basa en los principios del manifiesto ágil y sus valores éticos, que tratan de dar más valor a algunos conceptos, pero sin dejar de lado los demás. Son los siguientes:

1) Dar más valor a los individuos y a sus interacciones que a los procesos y herramientas.
2) Dar más valor al *software* que funciona que a la documentación exhaustiva.
3) Dar más valor a la colaboración con el cliente que a la negociación contractual.
4) Dar más valor a la respuesta al cambio que al seguimiento de un plan.

Métodos derivados

Existen varios métodos derivados de este modelo de desarrollo ágil, como Scrum, Kanban, Lean, AUP, Extreme Programming, etc.

Con estos valores se intenta conseguir, entre otras cosas, entregar algo lo más pronto posible y evitar problemas originados por cambios de requisitos. Esto es muy apropiado para proyectos cambiantes, ya sean grandes o pequeños, ya que mediante estos valores se pueden mitigar los riesgos. Para conseguir proyectos que puedan cambiar fácilmente, se pone especial atención en la calidad de los productos conseguidos, cosa que es realmente importante en proyectos de *software* para dispositivos móviles. Para conseguir esto, se basan en las pruebas de la aplicación y, a menudo, las automatizan.

Los métodos ágiles suelen ser muy adecuados para el desarrollo de aplicaciones móviles por las siguientes razones:

- **Alta volatilidad del entorno**: Con cambios en entornos de desarrollo, nuevos terminales y nuevas tecnologías a un ritmo mucho más elevado que en otros entornos de desarrollo.

- **Equipos de desarrollo pequeños**: Dado que los desarrollos móviles suelen ser proyectos relativamente pequeños, los equipos no suelen ser muy grandes. Generalmente son llevados a cabo por desarrolladores individuales o por PYME.

- ***Software* no crítico**: No suelen ser aplicaciones de alto nivel de criticidad, dado que suelen ser aplicaciones para entretenimiento o gestión empresarial no crítica.

- **Ciclos de desarrollo cortos**: Dada la evolución constante de la industria, se requieren ciclos de vida realmente cortos para poder dar salida a las aplicaciones a tiempo.

2.3.4. Mobile-D

El método Mobile-D se desarrolló junto con un proyecto finlandés en el 2004. Fue realizado, principalmente, por investigadores de la VTT (Instituto de Investigación Finlandés) y, a pesar de que es un método antiguo, sigue en vigor (se está utilizando en proyectos de éxito y está basado en técnicas que funcionan).

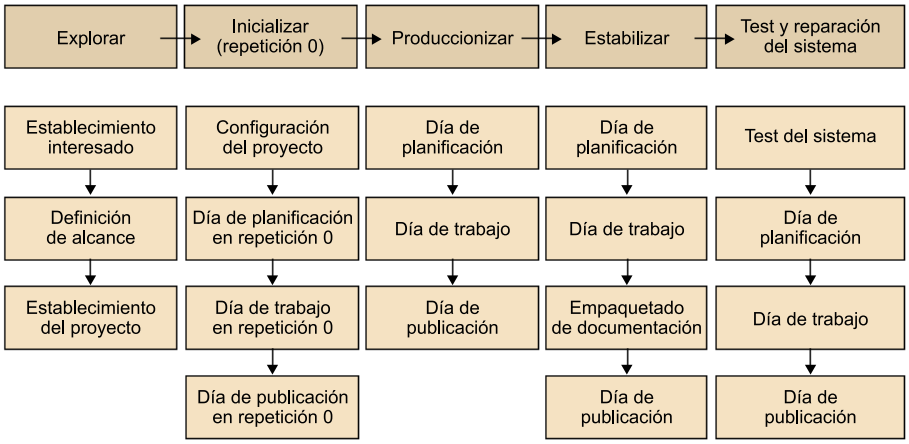
Enlace de interés

Para saber más sobre Mobile-D, podéis visitar la siguiente página web de la VTT: http://agile.vtt.fi/mobiled.html.

El objetivo es conseguir ciclos de desarrollos muy rápidos en equipos muy pequeños (de no más de diez desarrolladores) trabajando en un mismo espacio físico. Según este método, trabajando de esa manera se deben conseguir productos totalmente funcionales en menos de diez semanas.

Se trata de método basado en soluciones conocidas y consolidadas: Extreme Programming (XP), Crystal Methodologies y Rational Unified Process (RUP), XP para las prácticas de desarrollo, Crystal para escalar los métodos y RUP como base en el diseño del ciclo de vida.

Ciclo de desarrollo de Mobile-D



Cada fase (excepto la inicial) tiene siempre un día de planificación y otro de entrega. Las fases son:

- **Exploración**. Se dedica a la planificación y a los conceptos básicos del proyecto. Es diferente del resto de fases.

- **Inicialización**. Se preparan e identifican todos los recursos necesarios. Se establece el entorno técnico.

- **Productización o fase de producto**. Se repiten iterativamente las subfases, con un día de planificación, uno de trabajo y uno de entrega. Aquí se intentan utilizar técnicas como la del *test driven development* para conseguir la mayor calidad.

Test driven development

El *test driven development* (TDD) o desarrollo dirigido por las pruebas, indica que antes de realizar una funcionalidad debe existir una prueba que verifique su funcionamiento.

- **Fase de estabilización**. Se llevan a cabo las acciones de integración para asegurar que el sistema completo funciona correctamente.

- **Fase de pruebas y reparación**. Tiene como meta la disponibilidad de una versión estable y plenamente funcional del sistema según los requisitos del cliente.

2.4. Fases de los proyectos de desarrollo de aplicaciones móviles

Hemos visto que hay varios métodos existentes en el mercado para el desarrollo de aplicaciones móviles y que todos ellos se dividen en diferentes fases. Cada uno de estos métodos especifica lo que se debe hacer en cada fase, así como el nivel o los resultados que se requieren. Estas fases del desarrollo de aplicaciones móviles tendrán problemas comunes y soluciones comunes, de modo que vamos a repasarlas.

2.4.1. Planificación

En la fase de planificación se intenta distribuir el tiempo y los recursos necesarios para poder llevar a cabo el proyecto, ya sea en una única planificación completa o en planificaciones más divididas.

Durante las fases de planificación siempre se intenta conseguir la máxima precisión de las estimaciones, para minimizar los riesgos asociados al proyecto. En el caso de los proyectos de aplicaciones móviles, es necesario que tengáis cuenta los siguientes riesgos implícitos:

- **Dificultades por el desconocimiento de la tecnología**. Suele tratarse de tecnologías nuevas, en ocasiones desconocidas para los desarrolladores. Eso repercute en el tiempo de aprendizaje y puede ocasionar desviaciones por contratiempos no conocidos

- **Disponer de dispositivos reales**. Para poder realizar un buen proyecto de desarrollo de aplicaciones móviles, es necesario poder probarlo en dispositivos reales. Para ello, se debe planificar una fase de pruebas reales.

- ***Time to market***. A la hora de planificar hay que tener muy en cuenta el que suele tardar una idea en llegar al mercado e intentar reducirlo al máximo, pues la competencia es muy grande.

- **Prototipado**. Es importante planificar cuando se va a conseguir el primer prototipo (y, tal vez, la primera salida en beta), pues el prototipado rápido puede ser muy útil para este tipo de aplicaciones.

2.4.2. Toma de requisitos

Como en todo proyecto de *software*, es necesario que conozcáis los requisitos (tanto los funcionales como los no funcionales). En cuanto a los requerimientos no funcionales, deberéis tener muy presente aspectos relacionados con el uso como, por ejemplo:

- ¿Quien va a ser nuestro usuario? ¿En qué momento va a utilizar nuestra aplicación?
- ¿Qué requerimientos mínimos de *hardware* son necesarios?
- ¿Necesitamos gestionar el modo "en línea" y "fuera de línea"?
- ¿Deben existir datos de terceros? (desde un mapa, hasta datos del propio dispositivo).

Una vez tengáis todos los requisitos, es importante que dediquéis un tiempo a intentar ordenarlos, lo que os ayudará a tomar decisiones.

Un objetivo importante de esta fase es conseguir el plan de dispositivos o *device plan*.

Plan de dispositivos (*device plan*)

El plan de dispositivos o *device plan* es un listado ordenado de todos los dispositivos o grupos de dispositivos que se desean soportar.

Para elaborar el plan de dispositivos, agrupamos los dispositivos cuyo desarrollo se pueda atacar conjuntamente (por ejemplo, todos los que tengan Android 2.1 o superior o todos los que tengan GPS). A partir de ahí, se le da un valor de negocio a cada clase de dispositivo, así como el coste asociado al desarrollo de la aplicación para ese grupo. Finalmente, se consiguen los beneficios esperados de cada clase, tal como os mostramos a continuación:

	Coste estimado	Ingresos estimados	Beneficios previstos
Dispositivos de clase A	1	5	4
Dispositivos de clase B	2	4	2
Dispositivos de clase C	3	3	0
Dispositivos de clase D	4	2	-2
Dispositivos de clase E	5	1	-4

A la hora de decidir el coste, debéis añadir todos los posibles costes técnicos o no técnicos que puedan deberse a los requerimientos asociados.

Ejemplos

Si realizáis un juego 3D, debéis añadir el coste asociado al desarrollo en el caso de alguna plataforma no tenga librerías conocidas para ese tipo de modelado gráfico. Si queréis realizar una aplicación para gestión empresarial de inventario, por ejemplo, y realizáis la aplicación para Android y iOs, en caso de querer soportar los *tables PC* correspondientes, debéis tener en cuenta el coste que conlleva adaptar la interfaz de usuario.

Definición de la arquitectura

Siempre que se tiene un aplicación (móvil o no), existen varias opciones de arquitecturas. En el caso de los dispositivos móviles, hay aún más alternativas. Ya hemos visto que existe la posibilidad de tener sitios web móviles o aplicaciones web móviles o, cómo no, aplicaciones nativas, pero esto es solo una parte del problema, pues existen muchas arquitecturas de aplicación posibles en lo que respecta a las aplicaciones móviles.

En ocasiones, para poder tomar una buena decisión sobre la arquitectura, es necesario realizar pequeños prototipos. Esto dependerá del tamaño del proyecto y del conocimiento de la tecnología.

Sin duda, decidir la arquitectura de vuestra aplicación os ayudará a saber qué tipo de desarrollo vais a realizar (una aplicación web, una aplicación nativa o, tal vez, una híbrida). Por tanto, si no tenéis clara la arquitectura, muchas de las fases siguientes del proyecto pueden verse afectadas.

A continuación repasamos las arquitecturas más habituales en el desarrollo de aplicaciones para dispositivos móviles, como las aplicaciones "en línea", "fuera de línea" o las aplicaciones de sincronización. Finalmente repasaremos las aplicaciones que necesitan conexión entre dispositivos.

1) Aplicación "fuera de línea"

La aplicaciones "fuera de línea" son aplicaciones que, una vez descargadas, no requieren en absoluto de conexión (a excepción de las actualizaciones) para poder funcionar. Estas aplicaciones solo necesitan desarrollar la aplicación del dispositivo móvil (no son necesarios más componentes).

Tienen como ventaja que se pueden utilizar tanto con conexión como sin ella, pero suelen ser aplicaciones a las que, una vez instaladas, se les pierde el rastro.

Ejemplos de aplicaciones "fuera de línea"

Os mostramos algunos ejemplos de aplicaciones "fuera de línea":

- Muchos tipos de juegos que no comparten ningún tipo de información.
- Aplicaciones de productividad (como el gestor de tareas o las alarmas).

2) Aplicación totalmente "en línea"

Las aplicaciones totalmente "en línea" son aplicaciones que no pueden funcionar sin conexión a Internet. Estas arquitecturas requieren, sin lugar a dudas, de una parte de servidor, y están pensadas para mantener una comunicación constante con dicha parte servidora.

Tienen como desventaja que el usuario no puede utilizar la aplicación cuando no tiene conexión, pero disponen de información constante de las interacciones del usuario. Es necesario desarrollar, al menos, la parte servidora, tal vez una parte de desarrollo en el cliente y, en ocasiones, la comunicación entre ambos. Al necesitar estar siempre conectados, tienen un consumo extra de batería.

En ocasiones no es necesario realizar la parte servidora, ya que se trata de aplicaciones llamadas *mash-ups*, que son aplicaciones que aprovechan API existentes en la red para interaccionar con datos, (como, por ejemplo, la API de Twitter o datos públicos del estado).

Ejemplos de aplicaciones totalmente "en línea"

Os mostramos algunos ejemplos de aplicaciones totalmente "en línea":

- Todas las aplicaciones que se pueden utilizar en el móvil mediante el navegador web correspondiente (redes sociales, accesos a *webmails*, etc.).
- Aplicaciones web móviles nativas, que son aplicaciones web pero tienen apariencia y características de una aplicación nativa.
- Aplicaciones nativas que requieren de la conexión para poder estar autentificados o para obtener los datos.
- Aplicaciones de redes sociales o de tiempo real en las que la conexión es prácticamente imprescindible para tener la información actualizada.
- Cualquier tipo de *chat*, aplicación de llamadas o videoconferencias.

3) Aplicaciones de sincronización

Las aplicaciones de sincronización son aplicaciones que pueden funcionar en ambos modos, "en línea" y "fuera de línea", y permiten realizar las mismas acciones o acciones muy parecidas en ambos casos. La aplicación debe sincronizar los datos de la situación "fuera de línea" cuando se encuentre "en línea" y gestionar los posibles conflictos. Esto supone un beneficio para el usuario, ya que le permite trabajar en cualquier lugar y tener la información lo más actualizada posible.

La sincronización puede darse en un servidor propio, o bien con objetos o API para la sincronización en la nube, de modo que facilita su desarrollo y reduce costes. Hay diferentes tipos de sincronización:

- **Unidireccional**. Son aplicaciones que pueden sincronizar los cambios con algún servidor externo, bien porque en uno de los dos extremos solo es posible leer información, bien porque sirven como *backup* de información.

- **Bidireccional**. Son aplicaciones que pueden sufrir modificaciones tanto en el servidor como en el cliente, y la parte servidora y cliente se deben comunicar para realizar la sincronización.

Ejemplos de sincronización bidireccional

Os mostramos algunos ejemplos de sincronización bidireccional:

- Envío de estadísticas de juegos o de aplicaciones deportivas.
- Una información de visualización de mapas o de hojas de ruta para repartidores.
- Aplicaciones de envío y recepción de correo electrónico.
- Calendarios que se puedan modificar externamente a la aplicación móvil.
- Aplicaciones de suscripción y lectura de agregadores de contenidos (RSS)

4) Aplicaciones para la comunicación entre dispositivos

Las aplicaciones para la comunicación entre dispositivos son aplicaciones que interconectan dos (*unicast*) o más (*multicast*) dispositivos e intercambian información.

Este tipo de aplicaciones requieren desarrollar la parte del cliente además de la comunicación y la recepción de la información. También es necesario desarrollar la fase de búsqueda y enlace entre los dispositivos. Este tipo de aplicaciones suelen utilizar comunicaciones WPAN, como Bluetooth o NFC.

Ejemplos de aplicaciones para la comunicación entre dispositivos

Os mostramos algunos ejemplos de aplicaciones para comunicación entre dispositivos:

- aplicaciones de intercambio de contactos
- juegos entre dos o varios jugadores

2.4.3. Especificación y diseño

A diferencia de lo que sucede con otras fases del desarrollo de aplicaciones, la especificación no tiene grandes diferencias con respecto a las aplicaciones de sobremesa normales. Únicamente hay que tener en cuenta que, en muchas ocasiones, la fase de especificación se solapa con la del diseño.

En cuanto al diseño, existen algunos patrones de diseño ampliamente conocidos en el desarrollo de aplicaciones, que suelen ser implementados en las aplicaciones para dispositivos móviles:

- ***Model-View-Controler*** **(MVC)**. Se utiliza para poder separar al máximo la lógica de la visualización e interacción, y así poder dar soporte a más escenarios, como puede ser el caso de una aplicación para *smartphone* y la misma para *tablet PC*.

- ***Threading***. Se refiere al uso de hilos en segundo plano para realizar tareas largas que bloqueen al usuario.

- ***Delegation***. Se trata de delegar una parte del trabajo hacia otro objeto sin que este tenga que ser una subclase del primero. En el caso de los desarrollos móviles, es muy útil para delegar trabajos relacionados con la interfaz, de manera que pueda trabajar con eventos de manera muy sencilla y sostenible.

- **Modelo de memoria gestionada**. En general, las aplicaciones no se ejecutan directamente sobre la plataforma, sino que suele haber una capa intermedia o *middleware*, y esta suele gestionar la memoria. Sin embargo, para poder hacerlo de manera eficiente, es necesario realizar algunas acciones o convenciones.

Además de los patrones orientados a conseguir un *software* de mejor calidad, también existen patrones de diseño para maximizar el uso de nuestra aplicación. Estos patrones deben ser interpretados de manera diferente en función de la plataforma para la que desarrollemos, pues al final intentan utilizar los comportamientos comunes al resto de aplicaciones, y puede que estos comportamientos varíen según la plataforma.

Ejemplo

En Android se recomienda hacer que las navegaciones mediante pestañas se sitúen en la parte superior de la pantalla, mientras que en iOs se recomienda que estén en la parte inferior.

En general, existen unos principios básicos aplicables a todo diseño; por ejemplo, dar más valor a lo claro que a lo simple y al contenido que al continente. Para ello, existen algunos patrones de diseño de la UI (interfaz de usuario) que ayudan a resolver problemas conocidos y que han sido probados en numerosas ocasiones.

Entre los más conocidos se encuentran Dashboard, ActionBar, Dynamic List, Pager, *popups*, *alerts*, *etc*.

A continuación os explicamos algunos de ellos. Exponemos el problema que solucionan y cómo lo solucionan, así como algunas recomendaciones a tener en cuenta.

Cuadro de control (*dashboard*)

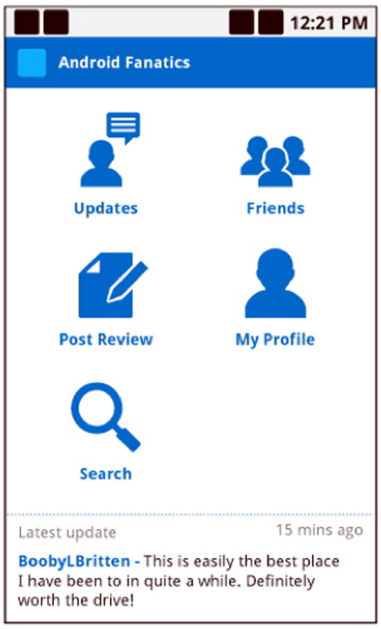
Problema: Acceder de manera rápida, clara y sencilla a las funcionalidades principales de la aplicación. En el caso del móvil, es importante por la responsabilidad que el usuario requiere.

Solución: Tener una página de llegada con la información clara de la última información de la aplicación y las acciones más importantes.

Se recomienda:

- Resaltar lo que es nuevo.
- Enfocar de tres a seis características importantes.

Ejemplo de patrón de interfaz de cuadro de control

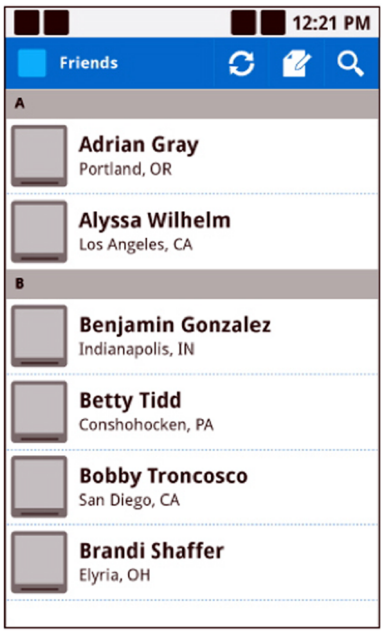


Barra de acción (*action bar*)

Problema: La limitación de espacio de las pantallas hace pueda ser muy complicado mostrar las acciones que el usuario puede hacer en una pantalla, y puede provocar mucha pérdida de espacio útil si se introduce un objeto visual botón por cada acción.

Solución: Se agrupan todas las acciones que se pueden hacer en la pantalla actual en una zona, en la parte superior o inferior (dependiendo de la plataforma), para aprovechar mejor el espacio y tener una mayor cohesión entre aplicaciones. Además, se puede aprovechar este espacio para indicar el lugar de navegación donde nos encontramos.

Ejemplo de patrón de barra de acción



Menús contextuales (*quick action menu*)

Problema: Las limitaciones de espacio para las acciones propias de un elemento de la pantalla ocuparían, potencialmente, mucho espacio.

Solución: Se muestra un menú contextual al pulsar el objeto, generalmente en la imagen (por ejemplo, en las imágenes de los contactos), y así se evita tener que añadir más ruido a la interfaz. Dentro de este menú se pueden agrupar todas las acciones correspondientes al objeto en cuestión.

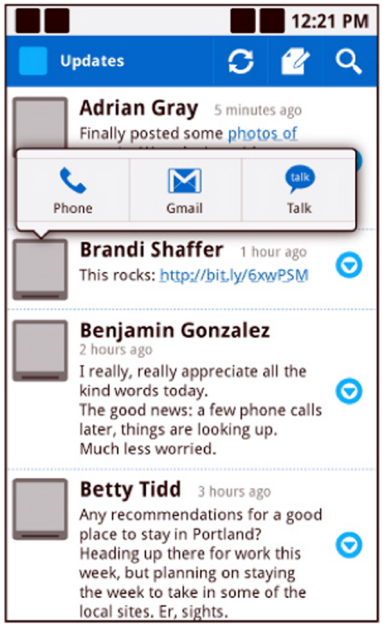
Se recomienda:

- Usarlo solo para las acciones más obvias e importantes

- Usarlo cuando el objeto no tenga una vista especialmente diseñada para él. En ese caso, hay que ir a esa vista y mostrar allí la información.

- No usar en contextos donde se soporten múltiples selecciones.

Ejemplo de uso de patrón de menú contextual

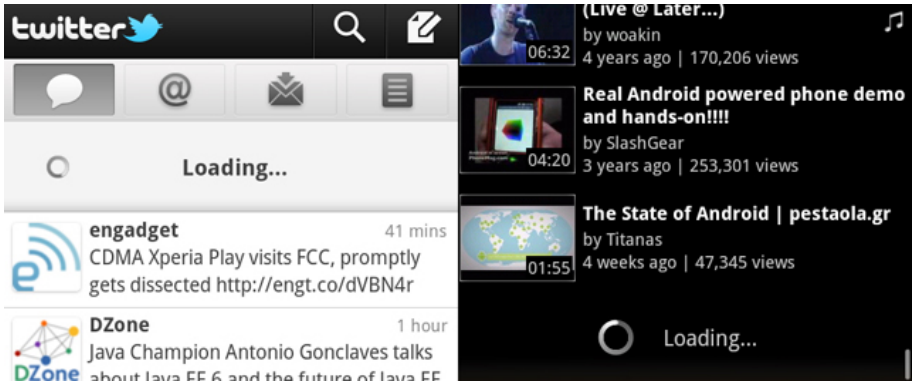


Lista dinámica (*dynamic list*)

Problema: Cargar muchos datos en una lista puede ser muy lento, especialmente si hay problemas de red. Para el usuario puede suponer una mala experiencia si tiene que esperar a que se cargue toda una lista (que en ocasiones no acaba o es muy grande).

Solución: En lugar de esperar a que la lista se cargue, se pueden mostrar los datos relevantes y cargar la lista inmediatamente. Los datos que faltan se pueden cargar cuando el usuario lo pida, o bien cuando la aplicación prevea que va a ser así (por ejemplo, cuando se desplace hacía el final de la lista).

Ejemplo de uso de patrón de lista dinámica



Mensajes de alerta

Problema: Sucede un evento en el sistema o la aplicación que es relevante para el usuario.

Solución: Mostrar un mensaje de alerta, que llama la atención del usuario, para advertirle de la situación. Por ejemplo, cuando hay problemas de batería o cuando nuestra aplicación ha detectado una perdida de conexión.

Ejemplo de patrón de mensaje de alerta

Se recomienda:

- Usarlo solo para mensajes importantes y necesarios. Este tipo de mensajes son muy invasivos, y el usuario, en lugar de sentirse agradecido por recibirlos, puede acabar ignorándolos.

- Realizar alguna acción cuando sea necesario; mostrar una de las opciones como atajo para dicha acción. Por ejemplo, si hay un problema de espacio en el disco, una de las acciones irá al gestor de aplicaciones instaladas para eliminar las que no necesitemos.

2.4.4. Implementación y pruebas

La implementación de aplicaciones para dispositivos móviles se asemeja mucho a la del resto de aplicaciones, aunque, generalmente, se trata de aplicaciones más pequeñas, o bien que tienen ciclos de desarrollo más cortos que las aplicaciones tradicionales. Esto se debe tanto a la propia naturaleza de la aplicación como a las necesidades del mercado que demanda conseguir prototipos o pruebas de concepto rápidas.

Una particularidad de las pruebas es la necesidad de tener un emulador o entorno de pruebas para poder probar aquello que estamos desarrollando. Esta necesidad provoca dificultades (no se puede realizar la prueba si no se dispone de un dispositivo) y ser lentas (por la lentitud de ejecución sobre los dispositivos o emuladores). Por este motivo, se acude a las pruebas unitarias que permiten dividir el desarrollo, con lo que se puede probar de manera desacoplada y desarrollar partes de nuestra aplicación sin la necesidad de pruebas en el emulador, como pueden ser los accesos a bases de datos o bien la lógica de negocio.

De cara a la implementación, hay una serie de factores que toman especial importancia en el caso de las aplicaciones para dispositivos móviles, y serán primordiales para el éxito del proyecto:

- **Usabilidad**. La usabilidad de la aplicación se debe tener muy en cuenta (debemos tener presente que la mayoría de usuarios no tendrán tiempo ni ganas de leer los manuales). Una buena práctica para mejorar la usabilidad es adaptar las aplicaciones a los comportamientos estándares de la plataforma, de manera que el usuario pueda aprovechar reglas mnemotécnicas o hábitos adquiridos.

- **Responsividad**. La aplicación debe responder a las acciones de usuario lo mejor posible y de manera ágil. Es un punto muy importante pues el usuario de dispositivos móviles suele ser más exigente que el de aplicaciones tradicionales en situaciones parecidas. Algunas plataformas, como Android, son muy estrictas a la hora conseguir que la aplicación sea suficientemente responsiva. Si la aplicación no es suficientemente ágil a la hora de responder al usuario, el sistema debe avisarle de esta circunstancia y permitir que la cierre inmediatamente.

Ejemplo de no responsividad

Una aplicación que, mientras lleva a cabo una tarea costosa, bloquea la interfaz de usuario, y además, no le informa del hecho.

- **Optimización de recursos**. Los dispositivos móviles, incluso los actuales, cuentan con unos recursos mucho más reducidos que las aplicaciones de sobremesa o las páginas web. Esto quiere decir que tenemos que hacer un buen uso de la memoria y del procesador del dispositivo, por lo que es conveniente cerrar los recursos que no se necesiten para evitar los problemas asociados.
  También es importante prestar mucha atención al consumo de batería de la aplicación. Por ello, es recomendable evitar cálculos excesivos (como los cálculos de coma flotante), el uso de las funciones de vibración o el uso de las conexiones inalámbricas. Cada una de estas recomendaciones se debe adaptar a la plataforma específica donde trabajamos, pues los fabricantes nos darán las recomendaciones concretas.
  Otra buena práctica, en este sentido, es perfilar la aplicación con las herramientas propias de la plataforma para encontrar problemas.

- **Accesibilidad de la aplicación**. Debemos tener en cuenta al diseñar la aplicación que los usuarios pueden necesitar acceder a ella de diferentes maneras. Por ejemplo, si tenemos una aplicación con formularios, es ideal que sea accesible tanto a través de las pantallas táctiles como de los *trackballs*.

Nota

Existen herramientas que traducen la aplicación a texto, haciéndola accesible a personas ciegas. Para ello, se debe desarrollar de manera correcta.

Importancia de las pruebas unitarias

Las pruebas unitarias sirven, como ya hemos comentado, para probar el correcto funcionamiento de una parte del código. Estas pruebas tienen como características más destacadas, que han de ser automatizables (no es obligatorio, pero sí muy recomendable), completas, reutilizables o repetibles a lo largo del tiempo, independientes entre si y tan profesionales como el propio código.

Si la prueba se centra en una parte de la aplicación que depende del dispositivo en qué se ejecuta, se la denomina prueba de integración.

Las pruebas unitarias agilizan el desarrollo porqué se centran en una parte del desarrollo y, por tanto, no será necesario probar dicha unidad dentro del emulador o emuladores, siempre que se trate de una parte realmente unitaria. Además, refuerzan la fiabilidad del desarrollo porqué se realiza al mismo tiempo que la prueba que lo verifica. Y las ventajas se multiplican cuando la prueba es automatizada pues evita la aparición de errores en un futuro al probar la aplicación de manera más exhaustiva.

Pruebas de integración contextualizadas

Las pruebas que realicéis deben estar contextualizadas; es decir, deben reproducir lo que realmente le está pasando al usuario cuando utiliza nuestra aplicación. Así, no es lo mismo probar un gestor de rutas para el coche que probar una aplicación para controlar los gastos de la empresa, pues se usan en momentos y con objetivos muy distintos.

Hay una serie de preguntas que se debe hacer la persona que pruebe la aplicación para poder realizar correctamente estas pruebas. Por ejemplo:

- ¿Cuál es la experiencia del usuario con la aplicación?
- ¿Se carga rápidamente la aplicación? ¿Se necesita alguna barra de progreso de la aplicación? ¿Y con conectividad reducida?
- ¿Cambia la aplicación al mover el dispositivo? Es decir, ¿se deben tener en cuenta los sensores de acelerómetros? ¿Reacciona con agilidad a estos cambios?
- ¿Acepta correctamente la aplicación las interrupciones como, por ejemplo, las llamadas telefónicas? ¿Acaba correctamente la aplicación? Es decir, ¿cierra correctamente todos los recursos utilizados?
- En caso de utilizar conectividad, geolocalización o cualquier otro servicio, ¿cuál es el comportamiento esperado de la aplicación ante la falta de dicha capacidad?

Para poder responder a este tipo de preguntas se deberán generar una serie de pruebas que llevaran a contextualizarla. Sin estas pruebas, la aplicación puede no funcionar bien en muchos casos. La mejor manera de llevar a cabo estas pruebas es, sin duda, sobre los propios dispositivos. El tipo, número y diversidad de las pruebas dependerá de los requisitos de la aplicación, pero se hace totalmente imprescindible haber realizado pruebas con dichos dispositivos antes de poder distribuir la aplicación.

Ejemplo

http://www.perfectomobile.com/ es un ejemplo de servicio que permite probar aplicaciones sobre dispositivos reales, a través de una granja de los mismos. Además, permite automatizarlas, lo que redunda en una mejora de la calidad.

Continuidad de las pruebas

Como en cualquier desarrollo, es importante que realicéis pruebas en cada nuevo desarrollo, pero en este caso lo es más que en otros entornos con mayor facilidad de cambio, pues generalmente (y en especial con aplicaciones nativas) los despliegues de la aplicación y de sus actualizaciones no suelen estar controlados por los desarrolladores. Esto quiere decir que cualquier cambio, *bug* o mal funcionamiento tiene mayor repercusión y se debe ir con mucho cuidado para evitar la reaparición de problemas ya solucionados. Una manera de combatir este problema es mantener un plan de pruebas y ejecutarlo en cada nueva versión.

Si se trata de una aplicación web para móviles, es mucho más fácil hacer cambios. En una aplicación nativa, las nuevas versiones pueden ser actualizadas mediante el propio sistema, dependiendo del entorno operativo (OC).

Es importante tener sistemas que aseguren la fiabilidad y la no reaparición de errores antes de lanzar una nueva versión, ya sean automatizados (como los sistemas de integración continua) o manuales (como planes de pruebas completos y reproducibles).

3. Negocio

Uno de los aspectos que están favoreciendo el crecimiento del desarrollo de aplicaciones móviles es la facilidad para poder hacer negocio con ellas. Rentabilizar una idea es, actualmente, más fácil que hace unos años.

Esto se ha conseguido gracias a la mejora del canal de distribución, principalmente con la aparición de los mercados de aplicaciones, que facilitan su distribución y compra.

Así, anteriormente los negocios en dispositivos móviles se basaban en la venta directa de aplicaciones a medida o en servicios de micropagos con SMS. En cambio, hoy en día han aparecido nuevos modelos de negocio que han sabido aprovechar la presencia de nuestro dispositivo móvil en nuestro día. Por ejemplo el pago en aplicaciones móvil por bienes virtuales o el pago por suscripción.

Con el fin de que entendáis las posibilidades existentes, en este apartado os mostramos una clasificación de los posibles negocios que se pueden realizar con las aplicaciones para dispositivos móviles. Después veremos con más detalle cómo son algunos de estos negocios, y pondremos especial atención en aquellos de reciente aparición. También veremos los pasos necesarios para conseguir publicar aplicaciones que se venden y distribuyen a través de mercados de aplicaciones, y las recomendaciones que son importantes para llegar a más usuarios.

3.1. Posibilidades de negocio

Existen muchas maneras de hacer negocio con las aplicaciones móviles. Entre los modelos clásicos de negocio con el móvil, destacaremos los siguientes, aunque hay que tener en cuenta que alguno de ellos ha sufrido una caída en las ventas, en parte por el abuso (principalmente mensajería) y la saturación del mercado:

- SMS premium. Mensajes de texto con un coste extra, también conocidos como servicios de tarificación adicional (STA).
- SMS para descargar aplicaciones o melodías.
- Desarrollo de aplicaciones a medida.

Nota

En el 2009 entra en vigor el código de conducta de los STA, que, entre otras medidas para regular el sector, obliga a todos los servicios basados en mensajes premium a avisar al usuario del coste (siempre que sea mayor de 1,2 euros). Este hecho se conoce como *opt-in*.

En contrapartida, han aparecido nuevos modelos de negocio, entre los cuales destacamos los siguientes:

- Vender la aplicación a través de una tienda de aplicaciones. Es el modelo más clásico, aunque no siempre es el más eficaz.

Terminología

A las aplicaciones gratuitas que tienen una versión de pago se las suele conocer como versión *lite*, y en ocasiones, a la versión de pago, como versión *pro*.

- Ofrecer versión gratuita, y con su distribución conseguir marketing de la aplicación o de los servicios relacionados, como puede ser una web inmobiliaria.

- Ofrecer una versión gratuita y una versión de pago. La versión gratuita no contiene todas las funcionalidades de la aplicación, de esta manera, el usuario puede probarla y, si le convence y quiere obtener las funcionalidades extras, deberá pagar por ellas.

- Vender publicidad. Diversas tecnologías permiten incluir en las aplicaciones, tanto en las basadas en tecnología web como en las nativas, anuncios contextualizados. En ocasiones la publicidad sólo aparece en la versión *lite*.

- Vender packs de contenidos extras. Se trata de vender la aplicación por partes, cada una con un coste. Esto es muy habitual en los juegos, por ejemplo, que ofrecen packs con distintos niveles.

- Llevar un negocio existente al mundo móvil:
  - Mejorar una línea de negocio, por ejemplo se hace añadiendo la localización geográfica a nuestro servicio.
  - Intentar conseguir nuevos clientes. Por ejemplo, conseguir clientes compulsivos; es decir, aquellos que si tuvieran que ir al ordenador no comprarían, pero teniéndolo en el bolsillo sí.

Ejemplo

Pensemos, por ejemplo, en una aplicación de información meteorológica o de tráfico, ofrecida vía web, que concrete la información en función de la localización del usuario.

- Construir una aplicación como un servicio (como, por ejemplo, Gootaxi, que permite acceder al servicio de taxis a través del móvil).

Ejemplo

Pensemos, por ejemplo, en información turística durante un viaje, o en información de descuentos del comercio donde está el usuario.

- Construir nuestra aplicación como una suscripción. Por ejemplo, el *streaming* de partidos de la NBA que ofrece el portal http://www.nba.com.

- Movilizar una tecnología existente. Por ejemplo, un CRM.

- Crear una aplicación que extienda un negocio "en línea", aprovechando principalmente las API públicas de Google, Twitter, Idealista, etc.

- Aplicaciones con sistemas *pay-in*. Esto significa que la aplicación puede pedirte pagos a cambio de opciones de la aplicación mientras está ejecutándose. Esto puede funcionar para aplicaciones como juegos para comprar elementos que ayuden al juego, o aplicaciones de modelado para comprar elementos especiales, o aplicaciones normales que piden pagos para servicios adicionales.

Según el público objetivo de la aplicación, la aceptación de que sea de pago será muy diferente: hay plataformas donde es probable que el usuario esté dispuesto a pagar y otras en que la publicidad es el mejor medio para conseguir ingresos por la aplicación.

A continuación veremos con más detalle dos de los modelos apuntados en este apartado:

- El modelo de aplicación gratuita
- El modelo de pago directo o indirecto

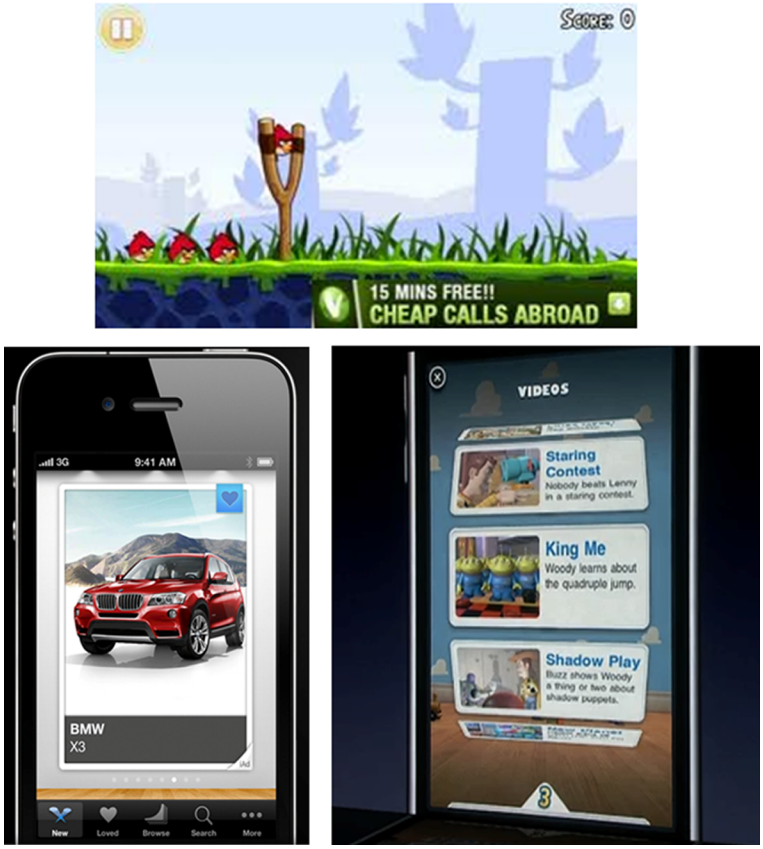
3.1.1. Modelo de aplicación gratuita

Hay muchos tipos de aplicaciones gratuitas, la mayoría sencillamente pretende atraer usuarios para que acaben comprando la versión de pago de la aplicación o productos relacionados con ellas.

Terminología

Este modelo de aplicaciones recibe el nombre de *freemium*.

Otra tipología de aplicación gratuita, a la que también nos hemos referido antes, es la que obtiene sus beneficios a partir de la publicidad, generalmente contextualizada. Este modelo de negocio es muy cercano al de las páginas web que obtienen beneficios por el número de visitas o por el número de clics en los anuncios que incorporan. Hay muchas empresas que ofrecen dicha publicidad y las propias plataformas incorporan soporte directo para ellas, como por ejemplo iAd para iOs, adMob para Android, Microsoft Advertising Exchange para Windows Phone 7 o Blackberry Advertising Service para Blackberry, entre otros. Y muchas de ellas se pueden usar en otras plataformas, lo cual hace algo más fácil portar las aplicaciones. También existen plataformas de publicidad específica para dispositivos móviles que ofrecen la publicidad en el formato correcto y con mayor contextualización.

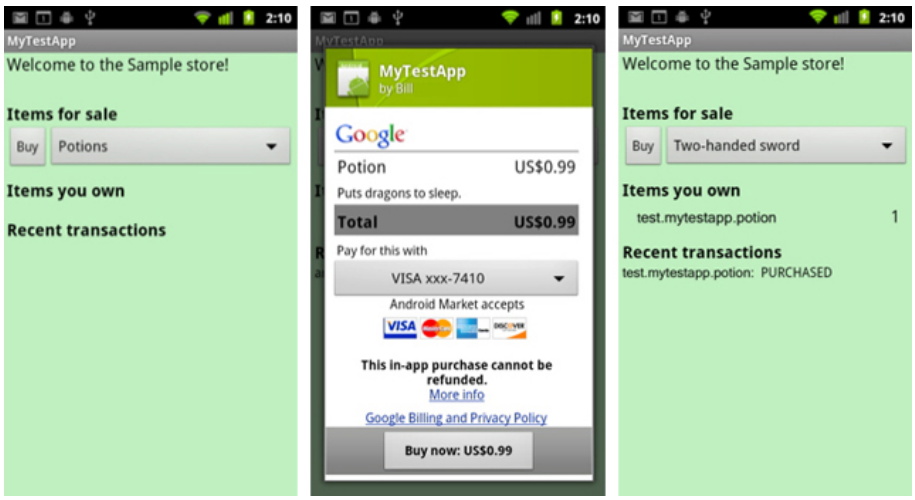


Ejemplo de publicidad en aplicaciones para dispositivos móviles

Este tipo de negocio tiene un beneficio oculto, que es el seguimiento (*tracking*) de los usuarios de la aplicación. Esta información variará según la plataforma de publicidad utilizada, pero siempre ayuda a definir el perfil de los usuarios, lo que redunda inmediatamente en posibilidades de mejora de la aplicación.

3.1.2. Pago directo o indirecto

Las aplicaciones que se deben pagar, ya sea en el momento de la descarga o bien en el momento de utilizar algún servicio concreto, conformarían este grupo, que también engloba las aplicaciones que tienen una versión gratuita y otra de pago.



Ejemplo de aplicación de tipo *pay-in*

El primer paso para conseguir que una aplicación pueda ser vendida una vez desarrollada es incluirla en una tienda de aplicaciones. Actualmente existen muchas tiendas oficiales, tanto de las plataformas como de los fabricantes (si son diferentes) y también tiendas alternativas o extraoficiales.

Las exigencias para poder incluir una aplicación en una tienda pueden variar mucho según la tienda de que se trate, lo que parece indudable es que el número de compradores potenciales suele ser mayor, cuanto mayores son estas exigencias: en una tienda oficial la aplicación deberá pasar unos controles de calidad que, seguramente, no existirán en una tienda alternativa. Evidentemente, el número de compradores potenciales de una tienda oficial es mucho mayor que el de una tienda alternativa.

En resumen, si pretendemos que nuestra aplicación tenga una perspectiva de descargas aceptable, deberemos incluirla en alguna de las tiendas oficiales antes mencionadas, para lo cual habrá que cumplir unos requisitos:

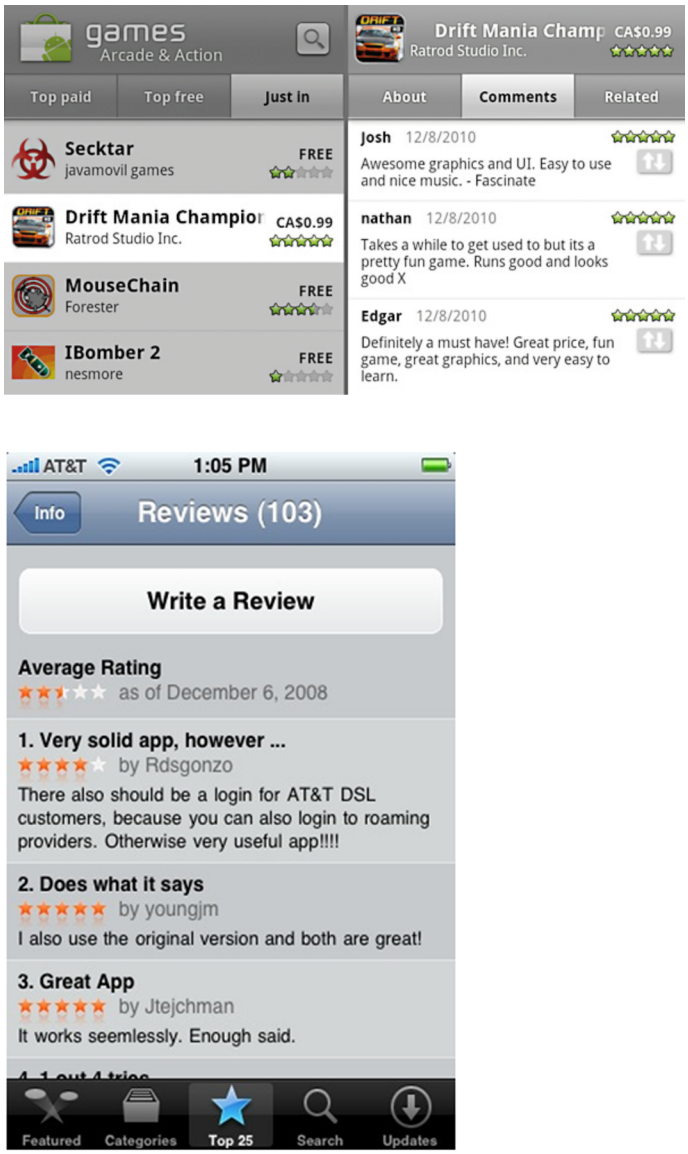
- Registrarnos y, generalmente, pagar para conseguir una cuenta de desarrollador que nos acredite como responsables de las aplicaciones.

- Subir la aplicación a la tienda a través de su zona de desarrollador.

- Superar el proceso de aprobación, lo que puede tardar días o semanas. En todas las tiendas es posible que nuestra aplicación sea eliminada en caso de causar problemas o bien de incumplir alguna de las normas de dicha tienda.

Una vez la aplicación está en la tienda, podremos controlar la información relativa a la misma, como el número de descargas, comentarios, valoraciones, etc.

Debemos tener presente que el éxito de las aplicaciones depende de que los usuarios las puedan encontrar fácilmente, por lo que es primordial situarse en la parte alta de la lista de aplicaciones de la tienda, ya que ocupar esas posiciones suele estar asociado a las descargas y a la valoración de los usuarios, por lo que es muy importante tener en cuenta sus opiniones.



Ejemplo de opiniones sobre las aplicaciones

Resumen

El objetivo principal de este módulo didáctico ha sido ver y aprender sobre las peculiaridades del desarrollo de aplicaciones para dispositivos móviles y, para ello, hemos presentado tanto las principales dificultades como los posibles beneficios derivados del tipo de dispositivos sobre el cual se van a ejecutar dichas aplicaciones y se ha profundizado en las estrategias de desarrollo posibles.

Por lo que respecta a las dificultades hemos destacado especialmente la fragmentación, y se ha estudiado su origen y cómo combatirla. En cuanto a los beneficios, hemos visto que la contextualización de la aplicación y su ubicuidad son los dos grandes puntos a tener en cuenta.

También hemos visto algunas pinceladas sobre los métodos utilizados en proyectos orientados a conseguir este tipo de aplicaciones, con sus puntos fuertes y puntos débiles. Prestando atención a fases que son especialmente delicadas en este tipo de proyectos, como el diseño de la interfaz y la selección del tipo de aplicación (web, nativa, etc.), o incluso únicas para este tipo de aplicaciones como son la distribución basada en *market places*.

Finalmente hemos presentado las posibilidades de negocio para este tipo de aplicaciones.

Actividades

1. Evaluad el estado del arte en las aplicaciones móviles en 3D, investigad las opciones de desarrollo y las API que existen para dos de las siguientes plataformas:

a) Android
b) IPhone
c) Java ME
d) BREW
e) Symbian

2. Instalad y desarrollad una aplicación cualquiera con, al menos, dos herramientas de desarrollo multiplataforma: Phonegap, Rhomobile, Wapps, Appcelerator, etc.

3. Planificad un proyecto de una aplicación sencilla para gestionar tareas compartidas. La planificación debe incluir temporalización, recursos humanos y materiales (dispositivos, por ejemplo).

4. Desarrollad un plan de implantación de dispositivos para los siguientes casos:

a) Una aplicación para leer firmas de documentos oficiales. Las firmas se realizan con el dedo o un bolígrafo sin punta sobre la pantalla del dispositivo.
b) Una aplicación para mostrar la situación de nuestros amigos de una red social y su último comentario.

5. Identificad entre las aplicaciones que usáis diariamente cuáles podrían ser móviles y cuáles no. Explicad los motivos.

6. Averiguad los costes asociados al pago de las aplicaciones.

7. Contabilizad el número total de aplicaciones vendidas y los beneficios generados para la plataforma y para los desarrolladores de tres de las principales plataformas del mercado con tiendas de aplicaciones.

8. Indicad cómo se puede evitar la fragmentación con iOs.

9. Explicad cómo funciona la gestión de la caché del navegador en HTML5.

10. Explicad la diferencia que hay en HTML5 entre base de datos local y gestión fuera de línea de la aplicación.

11. Especificad los componentes que se utilizan para conseguir aplicaciones web móviles nativas. Escribid un ejemplo para cargar la página principal de la UOC.

12. Indicad cuáles son las posibles fuentes de fragmentación actuales.

13. Enumerad las fuentes de fragmentación que se utilizan en una aplicación de visualización de fotografías para dispositivos Android.

14. Poned un ejemplo de aplicación LBS en el contexto de la restauración.

Glosario

accesibilidad *f* Grado en el que las personas pueden utilizar un objeto, visitar un lugar o acceder a un servicio, independientemente de sus capacidades técnicas, cognitivas o físicas.

API *f* Sigla de *application program interface*. Interfaz expuesta para ser utilizada dentro de una aplicación con el objetivo de dar acceso a librerías o funciones externas a la aplicación.

CRM *f* Sigla de *customer relationship management*. Software para gestionar la relación con los clientes.

CSS *m* Sigla de *cascade stylesheet*. Lenguaje usado para definir la presentación de un documento estructurado con XML o HTML.

derivación de software *m* Modificación sobre el software original para poder adaptarse a los cambios de la fragmentación.

device plan *m* Lista ordenada de todos los dispositivos o grupo de dispositivos que se desea soportar.

ecosistema móvil *m* Conjunto de actores necesarios para poder tener los dispositivos móviles y finalmente las aplicaciones para estos dispositivos. En concreto, en este ecosistema se incluyen las operadoras de telecomunicaciones, los fabricantes de hardware y los elementos de software que intervienen en la ejecución de la aplicación

escenario de una aplicación *m* Caso de fragmentación debido a una o varias de las posibles causas de fragmentación.

fragmentación *f* Situación, o condicionantes de la situación, que no permite compartir una misma aplicación entre diferentes ecosistemas sin hacerle las adaptaciones pertinentes.

***framework* de aplicaciones** *m* Marco de desarrollo que permite realizar aplicaciones de manera más sencilla, ordenada y mantenible.

fremium *m* Modelo de negocio gratuito en contraposición al modelo premium.

ide *m* Sigla de *integrated development environment*. Entorno de desarrollo que incorpora todas, o casi todas, las herramientas necesarias (herramientas de modelado o diseño, herramientas de *debugging*, etc.).

inverse of control *m* Patrón de diseño utilizado para definir las dependencias desde un contenedor externo.

iOs *m* Sistema operativo para dispositivos móviles de Iphone.

itinerancia *m* Capacidad de un dispositivo de moverse de una zona de cobertura a otra.

Javascript *m* Dialecto del estándar ECMA Script, utilizado para dar dinamismo a las aplicaciones web.

LBS *m* Sigla de *location based service*. Servicio que intenta ofrecer un valor añadido gracias al conocimiento de la ubicación geográfica del usuario.

lenguaje de marcado o lenguaje de marcas *m* Lenguaje que estructura la información a través de marcas o etiquetas (*tags*).

markup language *m* Lenguaje de marcado o lenguaje de marcas.

mash-up *f* Aplicación que aprovecha las API existentes en la red para interactuar con datos, como puede ser la API de Twitter, o datos públicos del Estado.

método *m* Definición de los pasos a seguir para conseguir un objetivo.

MMS *m* Sigla de *multimedia message system*

MoSoSo *m* Sigla de *mobile social software*. Software con una capa social añadida para aprovechar las conexiones sociales del usuario.

OTA *m* Sigla de *over-the-air*. Método para distribuir actualizaciones u otras funciones accesibles a través de Internet.

plugin *f* Aplicación añadida a una aplicación principal para ampliar y/o cambiar su funcionalidad.

preprocesador *m* Herramienta que permite realizar cambios sobre el código con el objetivo de adaptarlo a unas necesidades específicas.

prueba unitaria *f* Prueba que sirve para comprobar el correcto funcionamiento de una parte del código. Es recomendable que este tipo de pruebas sean automatizables, completas, reutilizables (repetibles a lo largo del tiempo), independientes entre sí y tan profesionales como el propio código.

responsividad *f* Capacidad de respuesta de una aplicación ante las acciones de usuario.

roaming *m* Itinerancia.

SDK *m* Sigla de *software development kit*. Herramientas necesarias para poder realizar el desarrollo de aplicaciones en una plataforma.

smart card *f* Tarjeta inteligente.

tarjeta inteligente *f* Tarjeta con un circuito integrado de tamaño bolsillo que se puede programar con algún tipo de lógica. Un ejemplo de ello son las tarjetas de crédito con microchip.

TDD *m* Sigla de *test driven development*. Desarrollo dirigido por las pruebas: antes de realizar una funcionalidad debe existir una prueba que verifique su funcionamiento.

ubicuidad *f* Capacidad de acceder a toda la información o servicios que necesita el usuario en cualquier momento y circunstancia, a través del dispositivo que tenga actualmente.

usabilidad *f* Facilidad con que las personas pueden utilizar una herramienta particular o cualquier otro objeto fabricado por humanos con el fin de alcanzar un objetivo concreto.

w3c *m* Sigla de World Wide Web Consorcium. Consorcio internacional que produce recomendaciones, que no estándares, para la World Wide Web (www).

WHATGW *m* Sigla de Web Hypertex Application Techonolgoy Working Group. Grupo formado por las empresas responsables de los principales navegadores web: Apple, Opera, Mozilla, Microsoft y Google.

WML *m* Sigla de *wireless markup languaje*

WURLF *m* Repositorio de información que permite identificar las capacidades y limitaciones de un dispositivo móvil a través de la metainformación de una petición.

Bibliografía

Ahonen, Tomi (2007). *Mobile the 7th Mass Media is to internet like TV is to radio*.

Allen, Sarah; Graupera, Vidal; Lundrigan, Lee (2010). *Pro Smartphone Cross-Platform Development*. Apress.

Blanc, Pablo; Camarero, Julio; Fumero, Antoni; Werterski, Adam; Rodriguez, Pedro (2009). *Metodología de desarrollo ágil para sistemas móviles*. Universidad de Madrid

Fling, Brian *et al.* (2009). *Mobile Design and Development*. O'Reilly.

Lehtimaki, Juhani. "Android UI Design Patterns".

Rajapakse, Damith C. (2008). *Fragmentation of mobile applications*

Spataru, Andrei Cristian (2010). *Agile Development Methods for Mobile Applications*. University of Edinbugh.

Virkus, R.; Gülle, R.; Rouffineau, T, y otros (2011) *Don't panic Mobile Developer's guide to Galaxy*. Enought Software Gmb H + Co. KG.

Wong, Richard (2010). *In Mobile, Fragmentation is Forever. Deal With*.

Woodbridge, Rob (2010). 9 Mobile Business Modesl taht yo ucan use right now to generate revenue.

Enlaces de Internet

http://www.comp.nus.edu.sg/~damithch/df/device-fragmentation.htm

http://techcrunch.com/2010/03/04/mobile-fragmentation-forever/

http://communities-dominate.blogs.com/brands/2007/02/mobile_the_7th_.html

http://www.versionone.com/pdf/mobiledevelopment.pdf

http://www.adamwesterski.com/wp-content/files/docsCursos/Agile_doc_TemasAnv.pdf

http://www.inf.ed.ac.uk/publications/thesis/online/IM100767.pdf

http://developer.smartface.biz/documents/Application_Development_Methodology.pdf

http://untether.tv/ellb/blog/8-mobile-business-models-that-you-can-use-right-now-to-generate-revenue/

http://en.wikipedia.org/

http://www.mit.jyu.fi/opetus/kurssit/jot/2005/kalvot/agile%20sw%20development.pdf

http://developer.android.com

http://developer.apple.com

Desarrollo de aplicaciones basadas en Android

Robert Ramírez Vique

PID_00178750

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

En este módulo os presentamos una introducción al desarrollo de aplicaciones móviles basadas en Android. Para estudiar la plataforma Android, nos centraremos en el tipo de aplicaciones nativas, de manera que podáis ver sus particularidades en detalle y tengáis una visión más clara del desarrollo de aplicaciones nativas.

Android nació como una novedad con mucho futuro, y ha pasado de ser una gran promesa a uno de los grandes de la industria. Con muchos dispositivos, y en constante crecimiento, es una tecnología que, sin duda, debéis tener en cuenta. Además, ha sido construida con unas bases muy sólidas y sin cerrar la puerta a ninguna opción de desarrollo ni arquitectura, lo cual os puede resultar muy útil para aprender los desarrollos móviles.

Haremos un recorrido por las diferentes fases de desarrollo de una aplicación móvil (para el caso de Android).

En primer lugar, veremos una pequeña introducción sobre la historia y las peculiaridades de la tecnología. Después expondremos las razones por las que Android es, actualmente, uno de los grandes candidatos a líder mundial.

También veremos en detalle las herramientas que conforman todo el ecosistema, la arquitectura de las aplicaciones y su ciclo de vida típico.

A continuación, nos centraremos en las diferentes funcionalidades que nos ofrece la tecnología para acceder al *hardware* de los dispositivos, en la construcción de interfaces de usuario y en el almacenamiento de datos o de las comunicaciones. Aprovecharemos este apartado para que veáis las opciones que nos ofrece la plataforma para probar y mejorar nuestra aplicación.

Finalmente, tocaremos todos los temas relacionados con las fases de distribución de la aplicación. Veréis cómo se puede llegar a publicar una aplicación para que los usuarios puedan comprarla y su seguimiento posterior, y también os explicaremos cómo podéis aprovechar las herramientas que ofrece Android para hacer negocio, las cuales no se reducen únicamente a la venta directa.

De esta manera, tendréis la suficiente visión global de la tecnología como para poder comenzar un nuevo proyecto Android, sabréis qué opciones hay en cada apartado o, en su defecto, dónde encontrar dicha información.

Hay algunos temas que no quedarán cubiertos en este módulo. Las razones son las siguientes:

- No veremos ningún tutorial, por lo que no existirá la descripción sobre cómo hacer un desarrollo paso a paso. Para eso, os proporcionarán recursos existentes fuera del módulo y específicamente diseñados para ello.

- No os presentaremos una guía de referencia exhaustiva del desarrollo en Android; para ello están las referencias oficiales, que son mucho más amplias y están actualizadas. Sin embargo, os expondremos casos concretos para ayudaros a comprender el desarrollo.

- Presuponemos que, gracias a otras asignaturas, tenéis bases de programación y de patrones de diseño de *software*.

Objetivos

Mediante el estudio de este módulo pretendemos que consigáis los objetivos siguientes:

1. Que sepáis qué es Android y cuáles son sus posibilidades.

2. Que sepáis cómo funcionan la aplicaciones en Android y dominéis las bases para poder desarrollarlas haciendo uso de las principales herramientas que existen.

3. Que tengáis una visión global de todas las posibilidades que ofrece la plataforma, tanto en lo que respecta a las arquitecturas como en lo referente a API.

4. Que conozcáis el proceso de comercialización de una aplicación Android, de modo que podáis llevarlo a cabo si es necesario, y su posterior seguimiento.

1. Introducción a Android

1.1. Pasado, presente y futuro de Android

Android fue fundada en el año 2003 por la empresa Android Inc. En el año 2005, Google la compró, e incorporó en sus filas a algunos de los principales valores de esta empresa, entre los que se encuentran Andy Rubin, Rich Miner y Chris White.

En noviembre del 2007, se crea la Open Handset Alliance, cuyo objetivo consiste en desarrollar estándares para dispositivos móviles. En esa misma fecha presentan su nuevo producto: Android, una plataforma construida sobre el *kernel* de Linux.

Open Handset Alliance

La Open Handset Alliance es un consorcio de varias empresas, entre las que se encuentran Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile y Texas Instruments.

En diciembre del 2008 se añaden a la *Open Handset Alliance* catorce nuevos miembros, entre los que se encuentran ARM Holdings, Atheros Communications, Asustek Computer Inc, Garmin Ltd, PacketVideo, Softbank, Sony Ericsson, Toshiba Corp y Vodafone Group Plc.

En mayo del 2008 conceden los premios del primer Android Developer Challange, un concurso para conseguir las mejores aplicaciones para Android. Gracias a este concurso y a sus siguientes ediciones, consiguen una gran repercusión, además de grandes aplicaciones.

En septiembre del 2008 lanzan la primera versión de la plataforma, la 1.0, pero aún no existía ningún dispositivo.

Finalmente, en octubre del 2008, aparece el primer dispositivo, llamado T-Mobile o HTC Dream, que se vendió inicialmente en Estados Unidos y, posteriormente, en el resto del mundo. Desde entonces, el número de dispositivos no ha dejado de crecer; actualmente ya hay centenares de dispositivos diferentes.

En febrero del 2011 se hace oficial la primera versión no orientada a *smarphones*, sino a *tablets PC*, con mejoras en la interfaz, soporte de *videochat* y capacidades para un *hardware* mucho más potente.

Actualmente se habla de más de cien mil activaciones de dispositivos diariamente con la plataforma Android y de casi doscientos mil desarrolladores, que ya han creado decenas de miles de aplicaciones; un número que no deja de crecer.

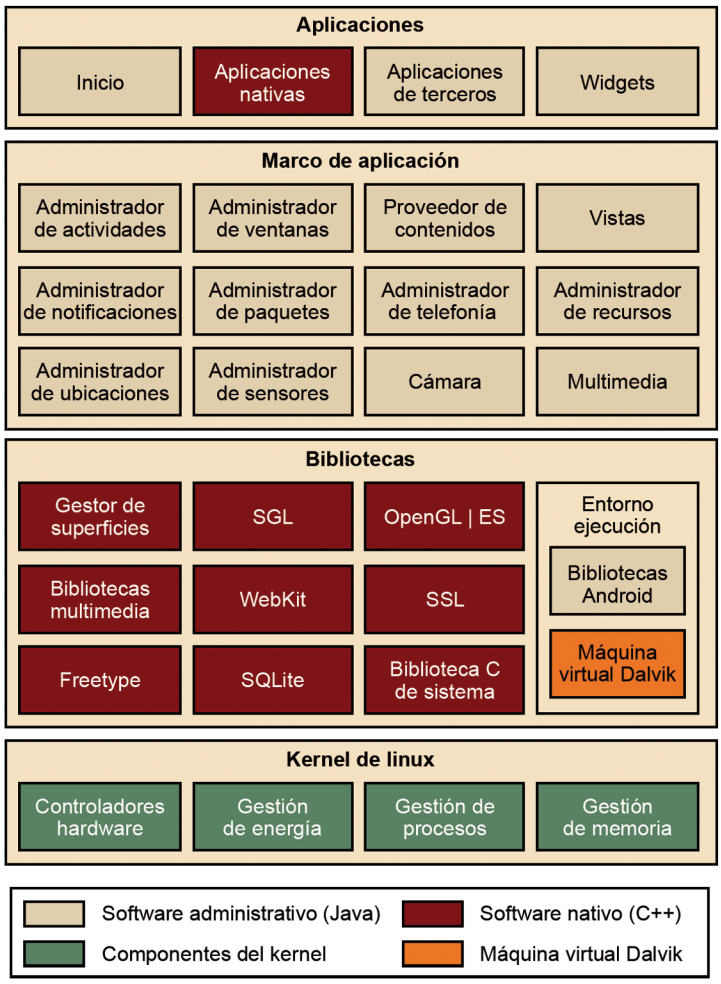
Esta gran cantidad de novedades y la velocidad a la que aparecen, provoca que existan muchas versiones disponibles en cada momento, y eso es algo que debéis tener en cuenta a la hora de desarrollar para Android.

1.2. ¿Qué es Android?

Android es una solución completa de *software* para dispositivos móviles. Incluye toda una pila de aplicaciones, como el sistema operativo, el *middleware* y las aplicaciones clave. También incluye herramientas para desarrollar en la plataforma, principalmente con el lenguaje de programación Java. Todo ello bajo licencia de código libre Apache, lo cual abre muchas posibilidades.

Android nos proporciona una serie de funcionalidades organizadas, tal como podéis ver en el siguiente gráfico.

Arquitectura de componentes de Android



1.2.1. Kernel o núcleo

El núcleo de Android está basado en el *kernel* de Linux, en el que se han realizado algunas modificaciones para adaptarlo a las necesidades de Android.

Este sirve como capa de abstracción del *hardware* al que tienen que acceder las aplicaciones. De esta manera, si se necesita un elemento *hardware*, la aplicación simplemente pedirá el recurso en genérico, sin tener que conocer el modelo exacto de *hardware*. Por ejemplo, si queremos acceder a la cámara, no importará el tipo, la definición ni el fabricante, simplemente accederemos a la cámara mediante el *driver* correspondiente.

1.2.2. Bibliotecas

La capa que se sitúa justo sobre el *kernel* la componen las **bibliotecas nativas de Android**. Estas bibliotecas están escritas en C o C++ y compiladas para la arquitectura *hardware* específica del teléfono, tarea que realiza normalmente el fabricante, que también se encarga de instalarlas en el terminal antes de ponerlo a la venta. Su cometido es proporcionar funcionalidad a las aplicaciones (para tareas que se repiten con frecuencia), evitar tener que codificarlas cada vez y garantizar que se llevan a cabo de la forma más eficiente.

Estas son algunas de las bibliotecas que se incluyen habitualmente:

- **Gestor de superficies**[1]: Se encarga de componer, a partir de capas gráficas 2D y 3D, las imágenes que se muestran en la pantalla. Cada vez que la aplicación pretende dibujar algo en la pantalla, la biblioteca no lo hace directamente sobre ella. En vez de eso, realiza los cambios en imágenes (mapas de bits) que almacena en memoria y que después combina para formar la imagen final que se envía a la pantalla. Esto permite realizar con facilidad diversos efectos gráficos.

(1) En inglés, *surface manager*.

- **Scalable Graphics Library (SGL)**: Desarrollada por *Skia* (empresa adquirida por Google en el 2005) y utilizada tanto en Android como en Chrome (navegador web de Google), se encarga de representar elementos en dos dimensiones.

- **OpenGL for Embedded Systems (OpenGL ES)**: Motor gráfico 3D basado en las API[2] de OpenGL ES 1.0, 1.1 y 2.0. Utiliza aceleración *hardware* (si el dispositivo la proporciona) o un motor *software* altamente optimizado cuando no la hay.

(2) API (*application program interface*)

- **Bibliotecas multimedia**: Basadas en OpenCORE, permiten visualizar, reproducir e incluso grabar numerosos formatos de imagen, vídeo y audio.

- **WebKit**: Motor web utilizado por el navegador (tanto como aplicación independiente como embebido en otras aplicaciones). Es el mismo motor que utilizan Google Chrome y Safari (el navegador de Apple, tanto en Mac como en el iPhone).

- **Secure Sockets Layer (SSL)**: Proporciona seguridad al acceder a Internet por medio de criptografía.

- **FreeType**: Permite mostrar fuentes tipográficas, tanto basadas en mapas de bits como vectoriales.

- **SQLite**: Motor de bases de datos relacionales, disponible para todas las aplicaciones.

- **Biblioteca C de sistema (libc)**: Está basada en la implementación de Berkeley Software Distribution (BSD), pero optimizada para sistemas Linux embebidos. Proporciona funcionalidad básica para la ejecución de las aplicaciones.

1.2.3. Entorno de ejecución

El entorno de ejecución de Android, aunque se apoya en las bibliotecas enumeradas anteriormente, no se considera una capa en sí mismo, dado que también está formado por bibliotecas. En concreto, por las bibliotecas esenciales de Android, que incluyen la mayor parte de la funcionalidad de las bibliotecas habituales de Java, así como otras específicas de Android. Se basa en funcionalidades del *kernel* (como el *threading* o la gestión de memoria a bajo nivel).

El componente principal del entorno de ejecución de Android es la máquina virtual **Dalvik**, componente que ejecuta todas y cada una de las aplicaciones no nativas de Android. Las aplicaciones se codifican normalmente en Java y son compiladas, pero no para generar un ejecutable binario compatible con la arquitectura *hardware* específica del dispositivo Android. En lugar de eso, se compilan en un formato específico para la máquina virtual Dalvik, que es la que las ejecuta. Esto permite compilar una única vez las aplicaciones y distribuirlas ya compiladas con la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera cada aplicación.

No se debe confundir Dalvik con la máquina virtual Java, pues son máquinas virtuales distintas. De hecho, Google creó esta máquina virtual, entre otras cosas, para evitar problemas de licencias. Java es únicamente el lenguaje de programación, y los programas generados no son compatibles en cuanto a *bytecode* Java.

Bytecode

Se entiende por *bytecode* al código formado por instrucciones ejecutables independientes del *hardware* final de la máquina.

Los archivos generados para esta máquina virtual, archivos .dex, son muchos más compactos (hasta un 50%) que los generados para la máquina virtual Java tradicional. Para conseguir esto, Dalvik se basa en registros, en lugar de una pila para almacenar los datos, lo que requiere menos instrucciones. Esto permite ejecuciones más rápidas en un entorno con menos recursos.

Las aplicaciones Android se ejecutan cada una en su propia instancia de la máquina virtual Dalvik, de manera que se evitan interferencias entre ellas, y tienen acceso a todas las bibliotecas mencionadas antes y, mediante estas bibliotecas, al *hardware* y al resto de recursos gestionados por el *kernel*.

1.2.4. Marco de aplicaciones

La siguiente capa la forman todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones y que, obviamente, se apoyan en las bibliotecas y en el entorno de ejecución que ya hemos detallado. La mayoría de los componentes de esta capa son bibliotecas Java que acceden a los recursos mediante la máquina virtual Dalvik. Entre las más importantes se encuentran las siguientes:

- **Administrador de actividades (*activity manager*)**: Se encarga de controlar el ciclo de vida de las actividades y la propia pila de actividades. Sobre estas se puede hacer una metáfora con las ventanas de las aplicaciones de escritorio.

- **Administrador de ventanas (*windows manager*)**: Se encarga de organizar lo que se muestra en pantalla y de crear, para ello, superficies que pueden ser "rellenadas" por las actividades.

- **Proveedor de contenidos (*content provider*)**: Permite encapsular un conjunto de datos que va a ser compartido entre aplicaciones y crear, de esa manera, una capa de abstracción que permita acceder a dichos datos sin perder el control sobre cómo se accede a la información. Por ejemplo, uno de los proveedores de contenido existentes permite a las aplicaciones acceder a los contactos almacenados en el teléfono. Esta biblioteca nos permite crear también nuestros propios proveedores para permitir que otras aplicaciones accedan a información que gestiona la nuestra.

- **Vistas (*views*)**: Si antes equiparábamos las actividades con las ventanas de un sistema operativo de PC, las vistas las podríamos equiparar con los controles que se suelen incluir dentro de esas ventanas. Android proporciona

numerosas vistas con las que construir las interfaces de usuario: botones, cuadros de texto, listas, etc. También proporciona otras más sofisticadas, como un navegador web o un visor de Google Maps.

- **Administrador de notificaciones (*notification manager*)**: Proporciona servicios para notificar al usuario cuando algo requiera su atención. Normalmente, las notificaciones se realizan mostrando alerta en la barra de estado, pero esta biblioteca también permite emitir sonidos, activar el vibrador o hacer pardear los LED del teléfono (si los tiene).

- **Administrador de paquetes (*package manager*)**: Las aplicaciones Android se distribuyen en paquetes (archivos .apk) que contienen tanto los archivos .dex como todos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación. Esta biblioteca permite obtener información sobre los paquetes actualmente instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes.

- **Administrador de telefonía (*telephony manager*)**: Proporciona acceso a la pila *hardware* de telefonía del dispositivo Android, si la tiene. Permite realizar llamadas o enviar y recibir SMS y MMS, aunque no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso (por motivos de seguridad).

- **Administrador de recursos (*resource manager*)**: Proporciona acceso a todos los elementos propios de una aplicación que se incluyen directamente en el código: cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos e, incluso, estructuraciones de las vistas dentro de una actividad (*layouts*). Permite gestionar esos elementos fuera del código de la aplicación y proporcionar diferentes versiones en función del idioma del dispositivo o la resolución de pantalla que tenga, para poder contrarrestar la fragmentación actual de dispositivos.

- **Administrador de ubicaciones (*location manager*)**: Permite determinar la posición geográfica del dispositivo Android (usando el GPS o las redes disponibles) y trabajar con mapas.

- **Administrador de sensores (*sensor manager*)**: Permite gestionar todos los sensores *hardware* disponibles en el dispositivo Android: acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, sensor de presión, sensor de proximidad, sensor de temperatura, etc.

- **Cámara**: Proporciona acceso a las cámaras del dispositivo Android, tanto para tomar fotografías como para grabar vídeo.

- **Multimedia**: Conjunto de bibliotecas que permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo.

1.2.5. Aplicaciones

La capa superior de esta pila *software* la forman, como no podría ser de otra forma, las aplicaciones. En este saco se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, tanto las nativas (programadas en C++) como las administradas (programadas en Java[3]), tanto las que vienen de serie con el dispositivo como las instaladas por el usuario.

(3)Se dispone de algunas de las librerías del JDK estándar de Java, pero no de todas ellas, y algunas de ellas están especialmente rediseñadas para ser ejecutadas en entornos móviles.

Aquí está también la aplicación principal del sistema: inicio (*home*), también llamada a veces lanzador (*launcher*), porque es la que permite ejecutar otras aplicaciones proporcionando la lista de aplicaciones instaladas y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o, incluso, pequeñas aplicaciones incrustadas o *widgets*, que son también aplicaciones de esta capa.

El aspecto principal a tener en cuenta de esta arquitectura es que todas las aplicaciones (las nativas de Android, las que proporciona Google, las que incluye de serie el fabricante del teléfono o las que instala después el usuario) utilizan el mismo marco de aplicación para acceder a los servicios que proporciona el sistema operativo. Esto implica dos cosas: que podemos crear aplicaciones que usen los mismos recursos que usan las aplicaciones nativas (nada está reservado o inaccesible) y que podemos reemplazar cualquiera de las aplicaciones del teléfono por otra de nuestra elección.

Este es el verdadero potencial de Android y lo que lo diferencia de su competencia: el control total por parte del usuario del *software* que se ejecuta en su teléfono.

Enlace de interés

Para saber más sobre qué es Android, podéis consultar la siguiente dirección de Internet:

http://developer.android.com/guide/basics/what-is-android.html

2. Fundamentos de las aplicaciones

Como hemos visto anteriormente, una aplicación Android suele ser una aplicación escrita en Java y compilada a un fichero de tipo .apk. Dicho fichero se considera una aplicación y es el elemento usado por los dispositivos Android para instalar la aplicación.

Ficheros .odex

Los ficheros .odex están pensados para aplicaciones de sistema optimizadas y cargadas en la máquina virtual en el momento de arrancar el dispositivo.

Una aplicación puede estar compuesta por uno o más componentes, los cuales realizan funciones diferentes para darle el comportamiento a la aplicación, y cada uno puede ser activado de manera individual.

Más adelante veréis por qué estados puede pasar nuestra aplicación, para qué sirve cada uno y cómo podéis prepararos para ello.

Manifiesto de una aplicación

Un elemento importante de las aplicaciones Android es su manifiesto, que declara la metainformación sobre la aplicación.

Es importante que tengáis presente que las aplicaciones Android se ejecutan sobre un único y separado proceso Linux que tiene su propia máquina virtual, de manera que tienen los recursos restringidos y controlados, cosa que es ideal para maximizar la seguridad. Siguiendo el principio de los mínimos privilegios, a cada proceso se le adjudican, por defecto, los permisos de acceso a los componentes que requiere y ninguno más.

Sin embargo, se pueden hacer excepciones:

- Se pueden tener varias aplicaciones compartiendo un recurso (para ello deben tener el mismo *user ID)*, para lo cual es necesario que estén firmadas con el mismo certificado.

- Si una aplicación sabe de antemano que quiere acceder a recursos como, por ejemplo, la cámara, Bluetooth, etc., debe pedírselo al usuario de forma explícita en el momento de instalarla. Esto se lleva a cabo gracias al manifiesto.

2.1. Componentes de una aplicación

Para facilitar la reutilización de código y agilizar el proceso de desarrollo, las aplicaciones Android se basan en componentes. Los componentes pueden ser de cuatro tipos:

Ejemplo de uso de componentes

Si una aplicación quiere utilizar una lista con barra de desplazamiento, puede utilizar una aplicación que sea simplemente esa lista para hacer la suya sin tener que duplicar el código.

1) Actividades (*activity*): Como hemos visto, son interfaces visuales que esperan alguna acción del usuario. Una aplicación puede tener una actividad o más, y desde una actividad se puede invocar a otras y volver nuevamente a la original. Todas las actividades extienden la clase *activity*. El contenido visual de cada actividad lo proporciona una serie de objetos derivados de la clase *view*. Android proporciona muchos de estos objetos prediseñados (como botones, selectores, menús, etc.).

2) Servicios (*services*): Los servicios no tienen interfaz gráfica, son procesos que se ejecutan en segundo plano para realizar acciones de larga duración.

Ejemplo de servicios

Un ejemplo sería la reproducción de una canción. Para una aplicación reproductora podríamos tener varias actividades para mostrar listas de canciones o un reproductor con botones, pero el usuario esperará que la canción siga sonando incluso al salir de la aplicación (terminar la actividad), por lo que esta aplicación deberá controlar un servicio para que se reproduzca la música. Cualquier servicio extiende la clase *service*.

3) Proveedores de contenidos (*content providers*): Permiten que una aplicación ponga ciertos datos a disposición de otras aplicaciones. Estos datos pueden estar guardados en ficheros, en una base de datos SQLite, en la web o en cualquier otro sitio que sea accesible. Mediante ellos, otras aplicaciones pueden preguntar o incluso modificar estos datos.

Por ejemplo, una grabadora de sonidos puede compartir esos datos con un reproductor de música. Estos datos pueden almacenarse en el sistema de ficheros o en una base de datos. Para proveer contenidos, se debe extender la clase ContentProvider.

4) Receptores de eventos (*broadcast receivers*): Estos componentes están simplemente esperando a que se produzcan determinados eventos (batería baja, cambio del idioma del dispositivo, la descarga de una imagen nueva...). Cualquier aplicación puede tener tantos receptores para tantos eventos como quiera. Cada uno de ellos debe extender la clase BroadCastReceiver.

Un ejemplo sería una galería de imágenes que indexa en la aplicación cualquier imagen que el usuario se descargue mediante el navegador u otra aplicación, para lo cual está escuchando el evento de descarga de imagen.

Como podéis ver, hay varias maneras de acceder a una aplicación y de comunicación entre las aplicaciones. Este es, sin duda, un punto fuerte de las aplicaciones Android. La comunicación entre aplicaciones, dado que se ejecutan en procesos con permisos distintos, no se puede hacer directamente. Para ello se utilizan los *intents*.

2.2. Ciclo de vida

El ciclo de vida de una aplicación está íntimamente ligado al de una *activity*, y su ciclo de vida también. Para que conozcáis mejor cómo funcionan las aplicaciones vamos a mostraros cómo se organizan a nivel de usuario.

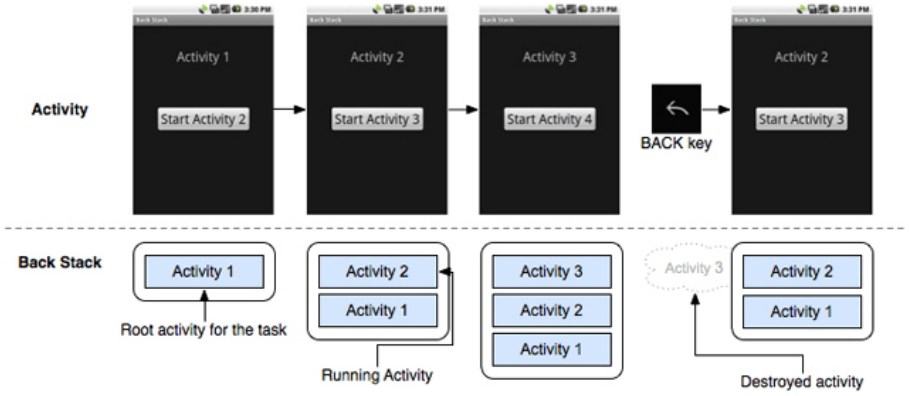
Unos elementos importantes en Android son las ***tasks***. Una *task* contiene una o varias *activities*. De alguna manera, es la tarea del usuario para conseguir algo.

Ejemplo de *task*

Por ejemplo, para poder escribir un correo electrónico, tendremos:

- La *activity* de gestor de correo electrónico, para ver el listado.
- La *activity* del cliente de correo electrónico (después se seleccionará el componer un nuevo correo).
- Posteriormente, para elegir el destinatario, la *activity* del selector de contactos.
- Si se desea añadir una fotografía, la *activity* de la galería.

Pilas de *tasks* y de *activities* y sus transiciones



Fuente: http://developer.android.com/

Podéis salir de una task pulsando el botón *atrás* (*back*) o bien con el botón *inicio* (*Home*). En el caso del botón *atrás*, se destruirán la *task* y todas las *activities* y, por tanto, si se vuelve a esa *activity*, se habrá perdido el estado (a no ser que haya sido guardado de forma explícita). En cambio, si se sale de la *task* mediante el botón *inicio* (*home*), el estado de la *activity* estará implícitamente guardado.

Por tanto, una *task* tiene varias *activities*, debido a que una *activity* puede invocar a otra *activity*, y todas ellas están en la misma *task*. Cuando una *activity* crea otra, lo hace mediante un *Intent*, y lo puede hacer de dos maneras: esperando resultados (startActivityForResult()) o no esperándolos (startActivity()). En ambos casos, una vez finalizada la *activity* creada, se vuelve a la *activity* creadora. Esto se puede producir muchas veces gracias a las pilas de actividades que hay dentro de una tarea. Estas tareas también tienen su pila para permitir interrupciones y, en cierto modo, permitir múltiples hilos de ejecución.

Múltiples hilos de ejecución

Múltiples hilos de ejecución (*multithreading*, en inglés), aplicado a la industria móvil, se refiere a la posibilidad de mantener varias aplicaciones ejecutándose sin cerrar ninguna de las dos.

Ejemplo

Estamos trabajando con una aplicación de calendario y vemos en una cita que hay una dirección de correo electrónico. Vamos a enviar un correo electrónico a esta cuenta, pero en ese momento nos llaman por teléfono.

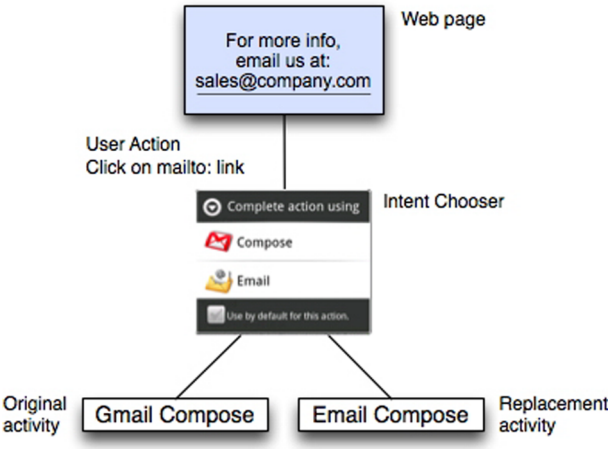
Como resultado, en la pila de *tasks* tendremos dos *tasks*, una correspondiente a la aplicación de calendario, y otra correspondiente a la aplicación de teléfono, debido a que la segunda ha interrumpido a la primera.

Además, la primera *task* tendrá varias *activities*, en concreto, en el listado de elementos del calendario, la *activity* para ver un elemento del calendario concreto, y finalmente la *activity* que corresponde a la otra aplicación, la del envío de correo electrónico.

Cuando finalice la *task* del teléfono, volveremos a la *task* anterior y con la *activity* de correo electrónico.

Por todo ello, las *activities* tienen que comunicarse entre ellas. Invocar una activity parece algo bastante habitual y un gran punto a favor de Android. Sin embargo, por otro lado se defiende hacer que las *activities* estén lo menos acopladas posible entre ellas. Esto se consigue con los *intents*.

Hay *activities* que pueden tener varios puntos de entrada, y para ello definen varios IntentFilters. Con estos IntentFilters se puede también hacer activities que esperen a *intents* conocidos, como puede ser enviar un correo electrónico o elegir una fotografía. Si hay más de una *activity* para un mismo *intent*, se nos preguntará qué *activity* iniciar al producirse este *intent*. La información sobre a qué *intents* va a reaccionar una *activity* está definida en el manifiesto.



Como vemos, una *activity* puede estar en diferentes estados mientras está ejecutándose. Estos estados se pueden resumir básicamente en tres:

1) Resumida: La actividad está en primer plano y tiene el foco del usuario. También se conoce como estado de ejecución o *running* (en inglés).

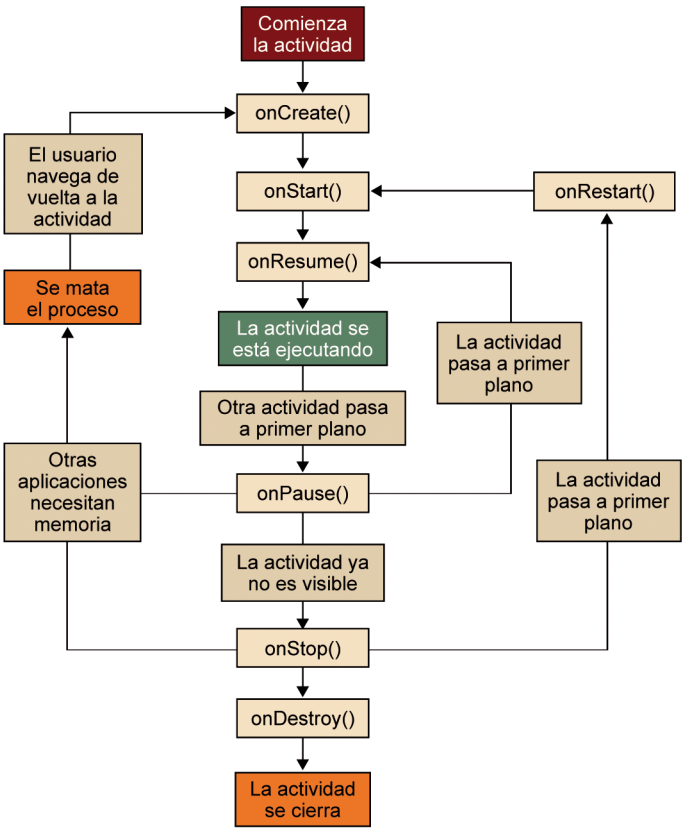
2) Pausada: Hay otra actividad en primer plano que está resumida, pero la pausada es aún visible. Esto puede suceder cuando la actividad en primer plano no cubre todo la pantalla o es parcialmente transparente. La actividad mantiene toda la información de memoria y está completamente viva, aunque puede llegar a ser destruida en un caso extremo de falta de memoria.

3) Terminada: La actividad está totalmente ocultada por otra actividad, y está en segundo plano. La actividad está totalmente viva, pero no es visible y puede ser destruida cuando el sistema necesite más memoria.

A las actividades pausadas o terminadas las puede destruir el sistema avisando (llamando al método *finish()*) o, directamente, destruyendo el proceso de sistema. Cuando una actividad vuelve después de ser destruida, se construye nuevamente.

Para conseguir programar las transiciones de estados de manera flexible y sostenible, Android provee un sistema de *callbacks* o retrollamada, de manera que el desarrollador puede tomar las acciones apropiadas en cada momento. A continuación podéis ver en la figura siguiente cuáles son:

Ciclo de vida de una *activity* en Android



Fuente: http://developer.android.com/guide

Hay algunos eventos que son especialmente importantes:

1) onCreate: Es el punto de inicio, donde tenemos que iniciar toda la interfaz gráfica, además de crear todos los elementos. Como se puede llegar después de haber sido destruida, es posible que tengáis que recuperar el estado de algún origen persistente.

2) onPause y onStop: Son los eventos correspondientes a las paradas de la aplicación, y se debe guardar el estado de la sesión correspondiente. Después de estos eventos, la aplicación pasa a estar pausada (onPause) o terminada (onStop).

3) onResume y onStart: La aplicación vuelve del estado pausada o terminada y debe recuperar la información de la sesión.

4) onDestroy: Es el punto en el que se deben liberar los recursos, justo antes de destruir definitivamente la aplicación.

En cuanto al ciclo de vida, es importante tener en cuenta que, una vez se ha pasado el estado onPause (siempre sucede antes de un onStop y un onDestroy), la aplicación puede ser destruida directamente por el sistema operativo. Por tanto, la manera segura de almacenar la información del estado de la sesión es en el método onPause, pero hay que hacerlo con cuidado, porque estamos bloqueando la siguiente actividad y podemos empeorar la experiencia de usuario.

Para facilitarle la vida al desarrollador, existe un método al que el sistema llama cuando cree que es necesario guardar el estado: onSaveInstanceState(). Todas las vistas lo implementan con una implementación por defecto (por ejemplo, un campo de texto guarda su valor). Consiste en guardar pares de clave y valor para su posterior recuperación. En el caso de las actividades, sucede lo mismo; se puede sobrescribir el comportamiento por defecto para guardar el estado, y el objeto donde se guardan las claves-valor se pasará al método onCreate cuando se recupere el estado después de ser destruido, pudiendo utilizar también el método onRestoreInstanceState.

Todo estos métodos utilizados para gestionar la destrucción y la creación de las actividades cobran especial interés, pues cuando hay cambios en la configuración (como la orientación de la pantalla, el idioma, la aparición del teclado virtual), el sistema reinicia la actividad (se llama al onDestroy y después al onCreate). Esto permite adaptar la aplicación a las nuevas configuraciones.

2.3. Intenciones (*Intents*)

Para que varias aplicaciones de una misma tarea puedan comunicarse entre ellas, se lanza una **intención** (en inglés, *intent*) o solicitud para que un componente lleve a cabo una tarea.

Los *intents* ofrecen un servicio de paso de mensajes que permite interconectar componentes de la misma o de distintas aplicaciones.

Los *intents* se utilizan para invocar una nueva actividad o para enviar eventos a múltiples actividades (llamados *broadcast intents*). Los *intents* son gestionados por los receptores de eventos o *broadcast receivers*, que pueden escuchar a cualquier *intent*.

Ejemplo de intención

Android lanza *intents* con los cambios de batería, los cambios de conexión o los mensajes entrantes para todos los receptores de estas intenciones.

Un *intent* se describe con los siguiente atributos:

- El nombre del componente que queremos avisar. Por ejemplo, la aplicación de correo electrónico.
- La acción que se quiere lanzar. Por ejemplo, editar, llamar, sincronizar, información de batería baja, etc.
- Los datos sobre la acción. Por ejemplo, escribir un nuevo correo.
- La información extra. La dirección de correo del destinatario y el título.

Estas intenciones se pueden invocar de dos maneras:

- **Explícita**: Se especifica de forma explícita en el código que componente es el encargado de gestionar el *intent*.

Ejemplo

```
Intent intent = new Intent(Context, Activity.class);
startActivity(intent);
```

- **Implícita**: Es la plataforma la que determina, mediante el proceso de resolución de intenciones, qué componente es el más apropiado para manejar el *intent*.

Ejemplo

```
Intent intent = new Intent(Intent.ACTION_DIAL,
URI.parse(tel:928-76-34-26));
startActivity(intent);
```

Un componente declara su capacidad de atender a un *intent* mediante el manifiesto, añadiendo etiquetas <intent-filter>.

2.4. *Content providers*

La manera de acceder o compartir información con otras aplicaciones es mediante los *content providers*. No existe ninguna zona común, ya que cada aplicación tiene su propio espacio de ejecución y sus propios permisos.

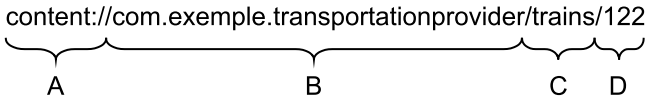
Estos *providers* os pueden permitir acceder, modificar o incluso borrar sus datos, siempre dependiendo del sistema de permisos que implementa Android. Además podréis ofrecer vuestros datos a otras aplicaciones, bien mediante la inclusión de esta información en algún *provider* ya existente, bien mediante la implementación de uno propio, al que tendremos que darle visibilidad.

Realmente no trabajaréis nunca con un *provider* concreto, sino que utilizaréis clases intermedias (por ejemplo, la clase ContentResolver). Para conseguir la instancia que os interesa, simplemente llamaréis al método de vuestra *activity* getContentResolver(), con lo que tendréis acceso para poder realizar las acciones de inserción, modificación, borrado y consulta de la información proporcionada por los *providers*. Para el caso de consulta de datos, también podéis utilizar el método managedQuery, que es el equivalente al método del ContentResolver.query.

Los datos que vais a intercambiar estarán en una única tabla, independientemente de cuál sea su representación interna. Cada columna tendrá un nombre y un tipo conocidos para poder tratar con los datos (el nombre de la columna). Cada fila de la tabla tendrá un identificador único, cuyo nombre de columna será *_ID*. Además, existen unas URI que os servirán para identificar cada uno de los recursos o conjuntos de recursos del *content provider*. Estas URI tendrán la siguiente forma:

URI

El *universal resource identified* (URI) es un identificador único dentro de la aplicación. Las URL son un tipo de URI.



donde:

- *A* es la parte fija para identificar que se trata de un *provider*.
- *B* es el nombre de la clase *provider*. Es necesario que se declare en el manifiesto.
- *C* es el *path* del *content provider*, que puede ser nulo (si solo se provee información de un tipo) o tan complejo como se desee si hay varios tipos (por ejemplo, *trains/bcn*, *trains/mdm*, etc.).

- *D* identifica una fila de los datos de manera única.

Para poder hacer una consulta de datos, tanto el método managedQuery() como el query() reciben los siguientes parámetros:

- **La URI que hemos visto antes**: Si la última parte (*D*) está presente, se consultará únicamente una fila.

- **El nombre de las columnas que queremos conseguir**: Si este parámetro es nulo, significa que queremos todos los datos.

- **Un filtro al estilo SQL**: Sería todo lo que puede venir en la sentencia WHERE, pero sin la palabra WHERE en concreto. Si es nulo se entregarán todas las filas.

- **Una serie de argumentos de selección**: Al igual que en otros lenguajes de consulta SQL, dentro de vuestros filtros podéis añadir caracteres interrogantes (?), que solo serán conocidos en el tiempo de ejecución. Aquí pondréis los valores correspondientes a esos interrogantes (en caso de existir).

- **Un criterio de ordenación**: Al igual que en SQL, podéis definir el orden ascendente o descendente.

Para modificar los datos, siempre tendréis que usar los métodos del ContentResolver. Así podréis pasarle una URI (general, si es inserción o modificación masiva, concreta de una fila si se trata de actualizaciones o borrado). Además, trabajaréis con objetos ContentValues para pasar los datos que queréis modificar.

Enlace de interés

Para saber más sobre los *content providers*, podéis visitar los siguientes sitios web.
http://developer.android.com/guide/topics/providers/content-providers.html

2.5. Manifiesto

El manifiesto de Android es un fichero XML que contiene información sobre vuestra aplicación. Este fichero se encuentra en la raíz de vuestro proyecto Android, y es imprescindible el sistema operativo ejecute vuestra aplicación.

Sirve para declarar información sobre nuestra aplicación que el sistema debe conocer de antemano:

- Permisos necesarios para poder funcionar, como acceso a la cámara.
- El nivel de la API de Android necesario para trabajar.

Nivel de la API de Android

Los niveles de las API de Android se refieren a la versión que ha sido distribuida. Estos niveles difieren de las versiones de la plataforma conocidas comercialmente y son un

número entero consecutivo. Así, la versión 1.0 tiene el nivel 1, pero la 2.0 tiene la 5. Sirven para identificar qué API o qué versión de las mismas está disponible.

- Las características *hardware* y *software* necesarias para trabajar (como, por ejemplo, tener cámara o no).
- Las API que necesita acceder, además de las que están incorporadas en la Android Framework API. Por ejemplo, la librería de Google Maps.
- Los componentes que forman nuestra aplicación.

2.5.1. Definición de componentes

Una parte importante e imprescindible en cada uno de los manifiestos es la definición de los componentes que forman la aplicación (están dentro del elemento XML <application>). Estos componentes pueden ser:

- <activity>, que corresponde a una actividad.
- <service>, para definir un servicio en segundo plano.
- <receiver>, para definir un receptor de eventos.
- <provider>, para definir un proveedor de contenidos.

En cualquiera de los casos anteriores se debe definir el nombre (android:name) que define el nombre de la clase totalmente cualificada.

Esta es, prácticamente, la única manera de definir un componente de nuestra aplicación y, por tanto, de hacer que sea utilizable en el ciclo de vida. Existe un caso en que no es necesario hacerlo así, que es en la creación dinámica de receptores de eventos, donde simplemente se crean objetos BroadcastReceiver y se registran con el método registerReceiver().

Como habéis visto anteriormente con los *intents*, es posible invocar componentes de forma explícita (excepto los proveedores de contenidos) utilizando una instancia creada de la clase, o bien de forma implícita mediante el manifiesto. Esto se hace con los elementos <intent-filter> del manifiesto, donde se define lo siguiente:

- **La acción (o acciones) a la que está asociado el componente**. Este campo es obligatorio y define la estructura del resto del filtro. Aquí se define la acción que se desea (por ejemplo, ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.).

- **La categoría**. Sirve para definir información adicional sobre la acción a ejecutar.

- **El tipo**. Define el tipo de los datos del *intent*.

- Y algunos elementos más, que no son obligatorios.

Ejemplo de definición de categoría

Con la categoría CATEGORY_LAUNCHER decimos que el componente debe aparecer en el lanzador (*launcher*) como una aplicación.

En el caso de las *activities*, un filtro muy normal es el de lanzar la aplicación. Por ejemplo, podemos tener el siguiente código para definir que nuestra aplicación se pueda ejecutar mediante el *launcher*:

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

2.5.2. Definición de requisitos

Una función importante que hace el manifiesto de Android es definir los requisitos mínimos de *hardware* y *software*. Esto sirve para filtrar las aplicaciones que se muestran en las tiendas de aplicaciones, de manera que solo se muestran aquellas que están disponibles para el dispositivo actual. Si intentáis instalar las aplicaciones en un dispositivo que no cumpla con alguno de estos requisitos, no se os permitirá.

Estos requisitos se definen en algunos elementos del manifiesto Android, la mayoría del estilo <uses-*>. Algunos de los más importantes son los siguientes:

- **Tipos de pantallas**, que se definen en el elemento <supports-screens> y especifican aspectos como el tamaño de la pantalla (con opciones como *small*, *normal*, *large* y *extra large*) o la densidad de píxeles (con las opciones de *low density*, *medium density*, *high density* y *extra high density*).

- **Configuraciones de elementos de entrada**, definidas en el elemento <uses-configuration>. Si vuestra aplicación requiere algún elemento especial (como los teclados, el *trackball*, un elemento de navegación direccional, etc.), lo debe especificar en esta sección.

- **Características del dispositivo**, definido en <uses-features>, determina si se necesita algún tipo de comunicación específica o algún sensor, o si se requiere cámara.

- **Nivel de la API**, definido en el elemento <uses-sdk>. Puede definir un mínimo y un máximo de las versiones del SDK soportadas.

- **Permisos**, definido en <permission>. Sirven para limitar el acceso a las partes sensibles del sistema, como android.permission.READ_OWNER_DATA.

Estos permisos se deben definir para que la aplicación pueda acceder a esas partes sensibles. También se pueden definir, en el mismo manifiesto, nuevos permisos.

- **Librerías externas**, definido en <uses-library>. Tendremos un elemento por cada librería externa.

No debéis asumir en vuestras aplicaciones nada más que las API estándares de Android. Cualquier otra cosa deberá ser mencionada en el manifiesto para asegurar su funcionamiento.

Enlace de interés

Para saber más sobre el manifiesto, podéis visitar el siguiente sitio web:
http://developer.android.com/guide/topics/manifest/manifest-intro.html

2.6. Recursos

Los recursos en Android son aquellos ficheros que pertenecen al proyecto. Pueden ser multimedia o estáticos. Según su organización en directorios y la información de contexto de la aplicación, se utilizarán para un tipo de dispositivo u otro. Dentro de los ficheros hay muchos ficheros XML que sirven para definir *layouts*, menús, cadenas de texto, estilos, etc.

Estos recursos constituyen una parte importante de la plataforma, ya que se usan en varias partes de la plataforma para objetivos distintos, pero tienen el mismo funcionamiento. Por tanto, si sabemos cómo funcionan para un caso, sabremos cómo funcionarán para el resto, ya que el comportamiento será idéntico.

El principal objetivo de estos recursos es ahorrarle al desarrollador su gestión y delegar la elección de qué fichero utilizar a la plataforma Android. Esta es una manera de atacar la fragmentación en Android.

Ejemplo

Por ejemplo, tenéis una imagen para la aplicación, pero la imagen ha de ser diferente para la aplicación, en función de la posición del dispositivo (vertical u horizontal). En este caso, únicamente tenéis que proporcionar las dos imágenes en el lugar correcto, y Android se encargará de gestionar qué aplicación utilizar.

Para que estos recursos puedan ser interpretados por Android, deben estar organizados de una manera especial, siempre dentro del directorio *res* de la raíz de nuestro proyecto. A partir de aquí, tenemos varios subdirectorios.

Vamos a ver alguno de ellos:

- *Drawable*: Son, básicamente, ficheros de imágenes o animaciones en XML.

- *Layout*: Son ficheros que definen el diseño o distribución de los elementos en la pantalla.
- *Values*: Son valores de cadenas de caracteres, enteros, colores, etc. que, por ejemplo, son candidatos a ser internacionalizados.
- Otros pueden ser *anim* (para ficheros XML de animaciones), *raw* (para ficheros arbitrario) o *menu* (para ficheros XML que definen un menú).

Para que estos recursos puedan ser utilizados desde el código Java, además de los usos implícitos (como algunos *layouts*), la plataforma los transforma automáticamente a un fichero llamado R.java. Esta clase Java se regenera en cada cambio de los ficheros de recursos, de manera que todas las propiedades son identificadas de forma unívoca por atributos de la clase R de Java (por ejemplo, R.layout.main).

2.6.1. Recursos alternativos

Como habéis visto, con los recursos se pretende aislar al desarrollador de la elección del recurso adecuado. Con estos recursos definidos fuera del código, se pretende diferenciar textos en diferentes idiomas o diferentes densidades de pantalla.

Los recursos alternativos al recurso por defecto se definen con nombres de directorios alternativos, que se forman con la siguiente fórmula:

<resources_name>(_<config_qualifier>)+

donde resources_name corresponde al tipo de recurso, que puede ser cualquiera de los vistos anteriormente, y config_qualifier sirve para determinar una configuración diferente. Se puede definir más de un config_qualifier (debéis usar, para separarlos, el guión bajo)

Ejemplo

Si tenéis un fichero icon.png, podéis tener el recurso por defecto en res/drawable/icon.png, mientras que la versión inglesa la tendréis en res/drawable_**en**/icon.png.

Android soporta varios calificadores de configuración. A continuación os mostramos algunas de las opciones más interesantes:

Evolución de los calificadores

Algunas de estas configuraciones se han ido ampliando a lo largo de la historia de Android para soportar los nuevos tipos de dispositivos, como en el caso de los *tablets*.

- **Operador y nación**: Sirve para identificar el operador que nos da la cobertura. Se puede identificar solo al código de nación, o nación y operador. Por ejemplo, mcc310 o mcc310-mnc004.

- **Idioma y región**: Lo idiomas con dos posibles componentes separados por un guión. Primero se define el idioma (según ISO639-1) y después se define la región (precedido de la letra *r* y, después, del código del ISO3166-1-alpha-2). Por ejemplo, fr-rFR, fr-rCA.

- **Tamaño de la pantalla**: Hay cuatro tipos, *small*, *normal*, *large* y *xlarge*.

- **Orientación de la pantalla**: Define la orientación actual. Existen dos opciones:
  - *port*, para orientación de retrato o vertical
  - *land*, para orientación de paisaje u horizontal

- **Modo noche**: Indica si es de día o de noche. Es útil para colores, por ejemplo. Tenemos dos opciones: *night* y *notnight*.

- **Densidad de la pantalla (dpi)**: Define varias opciones en número de píxeles posibles según la densidad de la pantalla. Por ejemplo:
  - ldpi: 120 dpi
  - hpdi: 240 dpi
  - xhdpi: 320 dpi

- **Modo de entrada de texto primario**: Indica el tipo de teclado del que se dispone. Así, podemos tener lo siguiente:
  - nokeys: No se dispone de teclado físico.
  - qwery: Se dispone de un teclado completo.
  - 12key: El dispositivo dispone de un teclado de doce teclas.

El orden de los calificadores es importante. Es el que aparece en lista anterior. Podéis verlo en detalle en la referencia oficial de Android.

Los nombres no son sensibles a las mayúsculas y minúsculas, y no se pueden repetir varios valores para un mismo cualificado.

Ejemplos

Con las combinaciones se pueden conseguir cosas como drawable_port_hpdi, o drawable_en_rUS_land, drawable_mcc310_ en_large_port_ night_nokeys.

Cada vez que se pide un nuevo recurso en tiempo de ejecución y mediante, por ejemplo, la clase *R*, Android realiza la búsqueda con los valores de las configuraciones actuales y las opciones disponibles en el directorio res.

Para que Android pueda escoger el recurso adecuado, utiliza un algoritmo de eliminación de opciones disponibles mediante el cual toma cada una de las configuraciones actuales y elimina las que contradicen la configuración hasta que solo quede una opción.

Para poder evitar problemas de recursos no encontrados, siempre es una buena práctica darle un valor al recurso en la carpeta y fichero por defecto, que es donde mirará Android en el caso de que no encuentre el recurso.

3. Interfaz gráfica

Una de las partes más importantes de cualquier aplicación es la interfaz gráfica con la que el usuario debe trabajar, también conocida como interfaz de usuario. Android, además, conociendo la situación de fragmentación de su interfaz, opta por una serie de soluciones para conseguir aplicaciones que se puedan mantener y adaptar de la manera más fácil posible.

Para ello, Android proporciona varias maneras de definir nuestra interfaz:

- Mediante el fichero layout.xml.
- Mediante la programación de los componentes visuales, como se hace en la mayoría de las aplicaciones de sobremesa.
- Mediante otras técnicas, que suelen ser la combinaciones de las anteriores (como, por ejemplo, el uso de LayoutInflater).

Independientemente de como se construya la interfaz gráfica, esta misma se compone de los mismos elementos, que son básicamente o View o ViewGroup, o cualquier clase que herede de estas.

Como muchas otras interfaces de usuario, Android tiene un modelo basado en el patrón de diseño observador y, por tanto, trabaja con *events* y *listeners*, lo que hace mucho más sencilla su gestión y codificación.

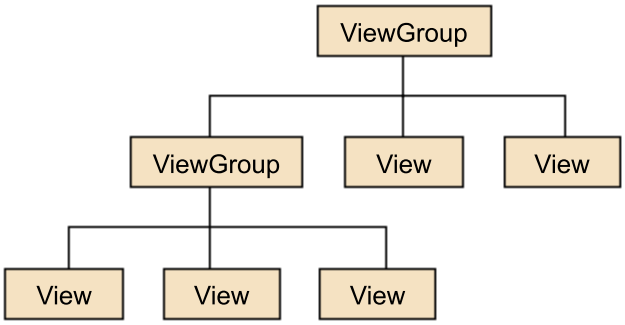
Dentro del ecosistema de desarrollo, siempre suelen aparecer facilidades para desarrollar trabajos laboriosos, y este es uno de esos casos. Veréis también cómo hay herramientas para realizar el trabajo de diseñar la interfaz gráfica de manera más sencilla y con menor probabilidad de errores.

3.1. Elementos de la interfaz gráfica

Podemos distinguir dos elementos de la interfaz gráfica: los View y los ViewGroup. Los primeros corresponden a todos los elementos gráficos (como botones, cajas de texto, elementos de selección, elementos para dibujar, etc.), y los segundos son los que sirven para agrupar a los primeros. Por tanto, un *ViewGroup* puede contener otros *ViewGroups* o más Views.

Esta manera de definir las interfaces hace muy fácil su reutilización, así como tener componentes autónomos.

Ejemplo de posible herencia de Views y ViewGroups



3.1.1. ViewGroup

Los ViewGroup son contenedores para los que Android proporciona una serie de subclases, que podéis usar para construir vuestra interfaz. Entre ellas destacamos:

- FrameLayout: Es el tipo más simple, con un espacio en blanco en el centro (que se puede rellenar con un objeto).

- LinearLayout: Dispone a todos sus hijos en una sola dirección, horizontal o vertical, uno detrás de otro.

- ListView: Muestra a todos los hijos en una columna vertical con barra de desplazamiento.

- TableLayout: Muestra a sus hijos de manera tabular, por columnas o filas.

- TabLayout: Sirve para dividir nuestra interfaz con pestañas, de manera que podamos tener, en una misma *activity*, vistas distintas organizadas por pestañas.

- RelativeLayout: Muestra los elementos de manera relativa a otros elementos o al contenedor. Es uno de los contenedores más utilizados, debido a su versatilidad a la hora de añadir nuevos elementos y a su capacidad para adaptarse a cambios de la interfaz (por nuevos datos, por ejemplo).

- AbsoluteLayout: Se definen los elementos en función de sus sus coordenadas x e y correspondientes a la esquina superior izquierda.

Es recomendable que intentéis usar contenedores relativos, en lugar de absolutos, para que os adaptéis mejor a todo tipo de pantallas y cambios.

En el caso de que ninguna de las subclases anteriores os interese, siempre podéis realizar vuestra propia subclase de ViewGroup o extender alguna ya existente.

3.1.2. View

Los View son los elementos que, finalmente, serán objetos visuales. Los objetos normales son conocidos como *widgets*. Aquí podemos ver los elementos de un formulario, como el botón, el campo de texto o elementos más sofisticados como, por ejemplo, un selector de fecha o de hora, o un elemento para votar (RatingBar).

También hay otros elementos más potentes, como los que exponemos a continuación:

- **WebView**: Es un elemento visual que incluye el resultado de visualizar un sitio web. Podéis habilitar o inhabilitar el uso de Javascript para este elemento según os convenga.

- **Gallery**: Sirve para mostrar elementos en una galería horizontal. Se selecciona un elemento central, y el siguiente y el anterior se suelen poder ver. Es, en realidad, un contenedor de elementos, pero se suele usar dentro de otros contenedores.

- **Autocomplete**: Es una caja de texto, donde se le hacen sugerencias de texto al usuario mientras escribe.

- **Google Map View**: Sirve para visualizar elementos objetos utilizando como contenedor un mapa de Google Maps.

Como ocurre con los contenedores, si no es suficiente con estos elementos, podéis realizar vuestras propias adaptaciones u objetos visuales totalmente personalizados.

3.1.3. *Fragments*

> Los fragmentos (*fragments*, en inglés) representan un comportamiento de la interfaz gráfica (o solo una porción del mismo) incluido en una *activity*.

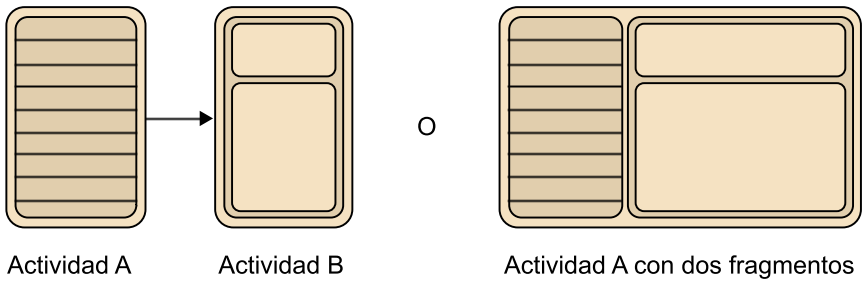
Los fragmentos aparecen en Android a partir de la versión 3.0, principalmente para soportar diseños flexibles y dinámicos en pantallas grandes (como, por ejemplo, las de los *tablets*). Se usan para evitar, de manera más sencilla, la fragmentación.

Siempre han de estar asociados a una *activity*, pero los *fragments* pueden estar definidos en un fichero XML diferente y tener un comportamiento específico, de manera que su reutilización puede ser muy sencilla. Es muy normal incluir varios *fragments* en la misma *activity* en pantallas grandes, y, en cambio, dividirlos en dos *activities* en pantallas más pequeñas.

Ejemplo de fragmentos

En la siguiente figura podéis ver un ejemplo de dos *fragments* divididos de diferente manera en función del tamaño de pantalla.

Fuente: http://developer.android.com/



El ciclo de vida de un *fragment* está íntimamente ligado al de la *activity* en la que está incluido, pero se pueden definir acciones diferentes para el *fragment* en los *events* del ciclo de vida, de manera que su funcionamiento sea más autónomo.

Una funcionalidad interesante de los *fragments* es la de realizar acciones a los *fragments* desde nuestra *activity*. Estas acciones pueden ser guardadas en la pila de *activities*, de manera que se pueda navegar hacia atrás en las acciones realizadas. Estas acciones se guardan como parte de la clase FragmentTransaction. A esta clase se le deben realizar las llamadas correspondientes a los métodos *add*, *remove* o *replace* para conseguir los cambios deseados. Para aplicar la transacción se debe llamar al método *commit*, pero si queréis que esta acción pueda ser desecha mediante la pila de *activities*, se debe llamar al método addToBackStack.

3.1.4. Otros elementos de la interfaz gráfica

Además de los elementos genéricos de la interfaz, existen algunos elementos o métodos de programación que se van añadiendo a Android para facilitar el desarrollo.

Menús

Los menús sirven para permitir al usuario realizar acciones especiales en función de a qué están asociados, si a una *activity* o a una *view* concreta. Pueden ser de tres tipos:

- **Menú de opciones**: Aparece cuando se pulsa el botón menú desde una *activity*.

- **Menú contextual**: Aparece después de pulsar de manera prolongada una *view*.
- **Submenú**: Aparece en caso necesario, cuando se pulsa un elemento de otro menú.

Estos menús. como el resto de elemento gráficos. pueden ser definidos mediante los recursos XML o bien mediante programación. Además, tienen *events* para gestionar sus acciones.

Notificaciones y diálogos

También es importante tratar el tema de las notificaciones y diálogos que se pueden mostrar al usuario. En lo que respecta a los diálogos, podemos generarlos mediante la clase *toast*, que consiste en simples mensajes emergentes con textos únicamente informativos, o bien mediante la clase *dialog* y sus subclases.

En cuanto a los mensajes informativos mediante *toast*, básicamente se generan llamando a makeText e informando del contexto visual, para poder mostrar el mensaje donde corresponda, con el texto y la duración asignada.

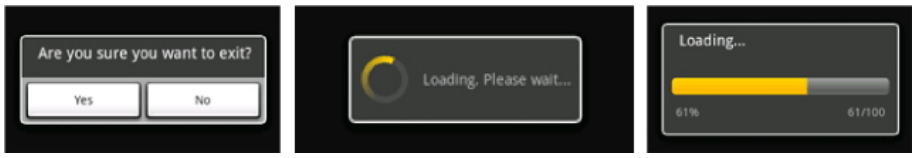
En cuanto a los diálogos, vamos a ver algunas subclases destacadas:

- **AlertDialog**: Sirve para mostrar información. Le damos de una a tres opciones al usuario y una lista de elementos entre los que elegir (con *checkboxes* o *radiobuttons*).

- **ProgressDialog**: Muestra un dialogo que informa que la aplicación está trabajando, y puede determinar su evolución.

- **DatePickerDialog/TimePickerDialog**: Sirve para escoger una fecha.

Algunos ejemplos de diálogos modales



Estos diálogos toman el foco de la aplicación y, hasta que no finalice la acción o se ignore el diálogo, no permiten continuar.

Las notificaciones, en cambio, son informaciones que no tienen que ser atendidas inmediatamente. Normalmente son informaciones generadas por servicios, que informan de *events* (como, por ejemplo, la llegada de un correo elec-

trónico). Estas notificaciones se quedan en la barra superior de Android (o en la barra de estado). Para crear estas notificaciones, debemos utilizar las clases Notification y NotificationManager.

Estilos y temas

En Android se pueden definir estilos. Estas propiedades las pueden heredar los elementos hijos del objeto visual donde lo hemos definido.

Los estilos son una colección de propiedades que definen el formato y la visualización de una ventana u objeto *view*. Estas propiedades son, por ejemplo, el tamaño, el color de fondo, el espacio exterior, el espacio interior, etc. Todos los estilos están definidos en un fichero XML, que está separado del que sirve para definir el *layout* de nuestras actividades.

Estos estilos se podrían definir de manera explícita dentro de las propias definiciones de la interfaz (ficheros XML del directorio *layout*), pero separarlos mejora la separación de conceptos y la reutilización. Es algo parecido a los estilos CSS[4] (que sirven para definir el estilo de un sitio web).

(4)CSS (*cascade style sheet*).

Estos estilos se definen con la propiedad *style* de un elemento *view*. Se pueden definir en el fichero XML, dentro de la definición del objeto *view*, o también en los constructores de estos objetos *view*.

Un estilo puede heredar de otro. Si se trata de un estilo que habéis definido vosotros, simplemente debéis dar el nombre de vuestro estilo prefijado por el estilo que queréis heredar.

Ejemplo

Tenemos un estilo *A* que define el tipo de letra. Podemos tener un estilo *B* que defina el color:

```
<style name="A.B">
   <item name="android:textColor">#FFFF00</item>
</style>
```

Si quisiéramos un estilo *C* que herede de *B* manteniendo *A*, se llamaría *A.B.C*, y así sucesivamente.

En el caso de heredar un estilo nativo de Android, debéis definir el atributo del fichero XML parent, por ejemplo, parent="@android:style/TextAppearance".

Un tema es un estilo aplicado a toda una actividad o aplicación, en lugar de a una vista en concreto. Por ejemplo, podéis definir un estilo de tamaño de letra para una *activity*, y todos los textos dentro de esa *activity* tendrán ese tamaño de letra.

La ventaja de los temas es que permiten ser cambiados fácilmente y cambiar la visualización de vuestra aplicación por completo. Estos temas se aplican mediante el atributo android:theme de una *activity*.

Objetos 3D y animaciones

Para darle mayor potencial a vuestra aplicación, podeis realizar animaciones y objetos con texturas de tres dimensiones (3D).

En el caso de 3D, podéis basaros en la librería OpenGL, en concreto, en la librería OpenGL ES API, que permite realizar renderizados con alto rendimiento de tres dimensiones. Para ello, se basa en librerías parecidas a las correspondientes de J2ME, y simplemente debéis realizar vuestro objeto *view* correspondiente e implementar el método onDraw para escribir el objeto 3D que deseéis.

También podéis tener objetos 3D mediante el sistema Renderscript, que maximiza el rendimiento, pues se trata de una librería de bajo nivel que ejecuta código nativo (en concreto, código C). Este sistema evita, además, que os tengáis que preocupar de las diferencias a bajo de nivel de los dispositivos. Eso lo consigue mediante la compilación en el momento (*just in time*) desde el *bytecode* hasta el código máquina. Este tipo de codificación maximiza el rendimiento, pero complica el desarrollo y la depuración del código.

Las animaciones sirven para definir las transiciones de nuestros objetos visuales. Estas animaciones pueden ser definidas dentro de nuestros ficheros XML, en el directorio *anim* o mediante el propio código.

Ejemplo de animación

Una animación puede ser el cambio una imagen por otras o la rotación de un botón (o la desaparición del mismo).

Otras formas de definir la interfaz

Como habéis visto, una manera típica de definir vuestros elementos visuales es mediante los ficheros XML de recursos correspondientes al contenedor. Estos ficheros permiten "setear" la mayoría de los atributos, así como el acceso desde el propio código. Es bastante habitual tener la definición inicial de la interfaz gráfica en un fichero XML y después modificarla mediante el código (accediendo a los elementos mediante la clase *R*).

Sin embargo, no todos los elementos gráficos que existen en las definiciones de XML se acaban renderizando; solo lo hacen aquellos que están en la línea de herencia de algún elemento renderizado. Por eso, en vuestra *activity* se debe llamar al método setContentView para indicar cuál es el contenedor de vuestra *activity*. Esto permite, además, definir elementos como *resources*, pero no renderizarlos inmediatamente, aunque sí se pueden renderizar a posteriori. Este proceso se conoce como *inflate*. El proceso utiliza el objeto XML procesado y genera la clase visual correspondiente.

3.2. *Events*

Los *events* son muy parecidos a otros entornos de desarrollo de interfaz gráfica basada en *events*. Estos *events* reaccionan a cualquier acción del usuario, ya sea pulsar una tecla en el teclado físico o virtual, pulsar un botón físico desde el botón menú hasta el botón de la cámara, tocar un punto en la pantalla o realizar acciones más sofisticadas (acercar o alejar, mover objetos, etc.).

Estos *events* siempre se realizan, y el hilo de la interfaz los interpreta. En el caso de que queráis realizar una acción ante uno de estos *events*, deberéis registrar un *listener* o escuchador para este *event* en el objeto visual que os interese. Según el tipo de *event* provocado, dicho *event* os traerá más información, que llegará en forma de parámetros a *listener*.

Ejemplo de *listeners*

A continuación os mostramos el uso típico de los *listeners* para escuchar la acción de hacer clic en un botón.

```
public class HelloWorldActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   Button btn = (Button) findViewById(R.id.button);
   btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Toast.makeText(HelloWorldActivity.this,
        "Click", Toast.LENGTH_SHORT).show();
      }
   });

  }
}
```

Vuestro código no siempre será el único interesado en escuchar estos *events* y, por tanto, debéis realizar un código seguro para evitar problemas en otros componentes. Unos pocos *events* son de una sola consumición; es decir, que no se pueden tener varios *listeners*, sino que el primero que lo recibe hace que el resto no lo vea.

Ahora veréis unos casos de *events* más sofisticados y muy particulares de las plataformas de dispositivos móviles: *events* táctiles y *events* de gestos predefinidos.

En el caso de los *events* táctiles o *touch events*, existen dos grupos: los *events* de una única acción y los *events* multiacción o *multi touch events*.

De los *events* de una única acción podemos tener:

- **onClick**: Es un clic básico sobre un objeto visual.

- **onLongClick**: Es un *event* de clic sin movimiento de este, pero con una duración más elevada.

- **OnDown**: Ocurre cuando un usuario pulsa la pantalla; es decir, en el momento de pulsar. Sería el equivalente al keyDown de una tecla o botón.

- **onScroll**: Se trata del *event* de hacer *scroll*, tanto vertical como horizontal.

- **onSingleTapUp**: Sucede cuando se libera la pantalla pulsada. Por ejemplo, si se realiza con un dedo, tiene lugar cuando se deja de tocar la pantalla (después de la acción táctil).

- **onFling**: Es una sucesión de *events* (tocar la pantalla, mover, dejar de tocar la pantalla). Suele utilizarse para desplazar objetos visuales.

Para el caso de los *multi-touch events*, tenemos diferentes acciones para el *event* onTouchEvent[5], que se pueden dividir en:

(5)El *event touch* se lanza en cada acción sobre la pantalla táctil e incluye las vistas anteriores de un único toque, así como las actuales de múltiples toques simultáneos.

- ACTION_DOWN: Pulsación de la primera acción de tocar la pantalla.

- ACTION_POINTER_DOWN: Pulsación de la segunda acción de tocar la pantalla.

- ACTION_MOVE: Acción de movimiento mientras se tienen los dos puntos pulsados.

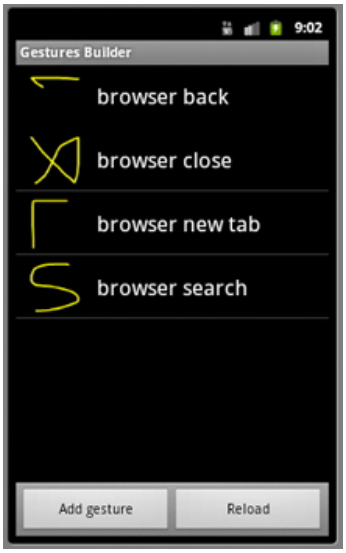
- ACTION_UP: Acción de liberar la pantalla correspondiente a la primera pulsación.

- ACTION_POINTER_UP: Acción de liberar la pantalla correspondiente a la segunda pulsación.

Estas acciones pueden ser usadas para realizar un *zoom*, por ejemplo.

Además de estas acciones complejas y personalizadas, Android ofrece unas librerías con las que gestionar los gestos predefinidos. Es decir, estas librerías son útiles para asociar acciones a gestos definidos que tengan sentido para vuestra aplicación. Esto se usa mucho para tener atajos a acciones habituales (como, por ejemplo, en navegadores para acciones como abrir una nueva ventana o ir hacia atrás).

Para definir este tipo de acciones, Android tiene una herramienta que ayuda a construirlos, el Gestoure Builder, así como un *event* para escuchar cuándo se producen dichas acciones por el usuario.



Ejemplo de definición de gestos usando el Gestoure Builder

La herramienta está como un programa de ejemplo de cada SDK (está en /samples/android-XX/GestoureBuilder/). Su función es la de registrar los *gestoures* y las traducciones que queréis que tengan estos gestos. Los gestos se van guardando en un directorio concreto de vuestro emulador para que podáis utilizarlos a posteriori.

Para utilizar los gestos creados, debéis colocarlos en vuestro proyecto, accesible desde vuestros recursos para ser referenciado desde el código. Para que vuestro código sea capaz de escuchar estos gestos, debéis comunicárselo a vuestra *activity* añadiendo una zona con una capa del tipo GestoureOverlayView, que será donde escucharéis este tipo de gestos. En el mismo código debéis añadir un *listener*, el cual estará escuchando sobre la capa antes definida y sobre el tipo de gestos que hemos construidos con el Gestoure Builder.

Ejemplo de preparación de los gestos

En este trozo de código podéis ver las partes más interesantes de la preparación de los gestos.

```
public class ListenGestouresActivity extends Activity implements
OnGesturePerformedListener {
  private GestureLibrary mLibrary;

  ...
  public void onCreate(Bundle savedInstanceState) {
    ...
          mLibrary = GestureLibraries.fromRawResource(this,
          R.raw.numbers);

          if (!mLibrary.load()) {
          finish();
          }
          GestureOverlayView gestures =(GestureOverlayView)
          findViewById(R.id.gestures);
          gestures.addOnGesturePerformedListener(this);
          ...
  }
}
```

Y su uso para predecir qué gesto se ha de realizar implementando el *event* gestourePerformed. Como podéis ver, os da una lista de posibles *events*, que llama predicciones. Esto os permite elegir en función de la precisión que queráis tener.

```
public void onGesturePerformed(GestureOverlayView overlay,
    Gesture gesture) {
    ArrayList<Prediction> predictions = mLibrary.recognize(gesture);
    ...
    Prediction p = predictions.get(i);
    String text = "got " + p.name + " with a precision of " +
    p.score;
    ...
}
```

3.2.1. AsyncTask y loaders

Siempre que se realiza una acción que pueda bloquear la interfaz, se debe realizar en un hilo de ejecución diferente. Además, es preocupante en el caso de Androides, pues puede aparecer un diálogo modal que informe que la aplicación no está respondiendo, (si tarda más de cinco segundos en hacerlo). Esto se puede hacer como siempre en Java, utilizando las clases propias. Podría ser algo así:

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      Bitmap b = loadImageFromNetwork();
      mImageView.setImageBitmap(b);
    }
  }).start();
}
```

Este código tiene un problema añadido, y es que se está manipulando la interfaz desde fuera del hilo de la interfaz (UI thread), que no está preparada para ser modificada por múltiples hilos, con los problemas que ello puede conllevar.

Para evitar este problema, Android ofrece varias formas de modificar la UI[6], como Activity.runOnUiThread(Runnable), View.post(Runnable), View.postDelay(Runnable, long) y Handler. Sin embargo, estas clases hacen que el código sea poco elegante y difícil de mantener.

(6)UI (*user interface*): interfaz gráfica o interfaz de comunicación con el usuario

También se podría hacer utilizando una clase específicamente creada para las actualizaciones asíncronas de la interfaz llamada AsyncTask. En este caso, quedaría como sigue:

```
public void onClick(View v) {
  new DownloadImageTask().execute("http://example.com/image.png");
}


private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
      protected Bitmap doInBackground(String... urls) {
         return loadImageFromNetwork(urls[0]);
      }

      protected void onPostExecute(Bitmap result) {
         mImageView.setImageBitmap(result);
      }
}
```

Con esta clase se pueden hacer todas las modificaciones de la interfaz de manera segura para ella. Como vemos, la AsyncTask utiliza los *generics* de Java para tener un código más controlado y elegante y, además, le permite al programador añadir la lógica sobrescribiendo los siguientes métodos:

- **onPreExecute**: Invocado dentro del hilo de la interfaz, justo antes de empezar la tarea. Es útil para actualizar la interfaz gráfica (por ejemplo, con una barra de progreso).

- **doInBackground(Params...)**: Invocado en un hilo distinto después del *onPreExecute*. Es donde se realiza el trabajo costoso. En esté método se pueden realizar *publishProgress(Progress...)* para informar a la interfaz del progreso.

- **onProgressUpdate(Progress...)**: Invocado en el hilo de la interfaz después de cada llamada al *publishProgress(Progress...)* (realizado en el paso anterior).

- **onPostExecute(Result)**: Se ejecuta en el hilo de ejecución de la interfaz al finalizar el trabajo.

Estas AsyncTask pueden ser canceladas en cualquier momento, lo cual supone llamar a otro método para poder cancelar correctamente nuestra AsyncTask y liberar los recursos.

Otros elementos importantes son los *loaders*, que sirven para cargar de manera asíncrona datos en nuestra aplicación.

Las características principales de los *loaders* son los siguientes:

- Se pueden asociar a cada una de las *activities* o *fragments*.

- Permiten recoger datos de manera asíncrona.

- Monitorizan los recursos, de manera que, cuando hay cambios en los datos, entregan cichos datos automáticamente.

- Después de un cambio de la configuración, se conectan automáticamente al cursor de los datos antes de dicho cambio, de manera que no es necesario volver a preguntar por dichos datos.

Es responsabilidad del desarrollador encargarse de conseguir los datos de manera correcta y mostrarla al usuario. Los *loaders* se encargarán de mantener los datos y la nueva información.

4. Otras partes del SDK

Android tiene muchas partes y cada una de esas partes es realmente grande, por lo que se hace difícil cubrirlo todo y, seguramente, muchas de estas partes no serán necesarias para el desarrollo de nuestra aplicación. Por eso, vamos a mostraros algunas de estas partes que son importantes, partes relacionadas con librerías que existen y que ofrecen almacenamiento de información, acceso a sensores o mejoras en el uso y en el desempeño de nuestra aplicación.

4.1. Almacenamiento de datos

Se pueden almacenar datos de diversas maneras:

- mediante las preferencias de la aplicación
- en ficheros en el almacenamiento interno
- en ficheros en el almacenamiento externo
- mediante bases de datos
- mediante la red

Si se almacenan datos en las preferencias de la aplicación, se utiliza la clase SharedPreferences mediante los métodos de la actividad getSharedPreferences() o getPreferences(). Aquí podréis guardar pares de clave valor, donde los valores serán tipos primitivos. Estos datos serán automáticamente persistidos entre diferentes sesiones de usuario.

El almacenamiento interno y externo consiste en guardar ficheros, bien en la memoria interna del dispositivo, bien en la memoria externa, generalmente en una tarjeta de memoria. Existen métodos para cada caso, de manera que podéis abrir un fichero con el nombre concreto, leerlo, escribirlo y, finalmente, cerrarlo. También existe la posibilidad de trabajar con los ficheros de caché, cuyo propósito es mejorar la experiencia de usuario evitando realizar acciones innecesarias.

Es importante saber que hay algunos directorios en los que los ficheros guardados serán visibles para todas las aplicaciones y no se borraran aun cuando se desinstale la aplicación (por ejemplo, los directorios *music*, *podcasts*, *ringtones*, etc.)

En el caso del almacenamiento en base de datos, se utilizará SQLite. Las bases de datos que creéis serán accesibles por nombre desde cualquier clase de nuestra aplicación, pero no desde fuera de la misma. Os recomendamos que utilicéis la clase SQLiteOpenHelper para crear vuestras bases de datos y que trabajéis sobre ellas.

Sobre estas bases de datos podréis realizar todo tipo de sentencias SQL. También tendréis la opción de recuperar datos en un cursor y trabajar con ellos.

Finalmente, para guardar datos en la red, debéis acceder a la red y enviar la información mediante servicios "en línea".

4.2. Acceso nativo

Android Native Development Kit (Android NDK) es un conjunto de herramientas que permiten empotrar componentes que usan código nativo en las aplicaciones Android tradicionales.

Estas partes de las aplicaciones se escribirán en C y C++, y se beneficiarán del hecho de ser aplicaciones nativas para volver a aprovechar ciertas partes o conseguir un mejor rendimiento.

Estas aplicaciones son más complejas, y solo por su naturaleza no conseguirán mejoras, sino que deberéis saber exactamente qué hacer para aprovechar mejor los recursos de bajo nivel. Por ese motivo es importante que sepáis cuándo utilizarlo.

> Una buena regla general es utilizar NDK solo cuando sea necesario, no sin haber examinado antes otras alternativas.

Las aplicaciones candidatas a utilizar NDK suelen tener un uso intensivo de CPU, pero no utilizan mucha memoria (por ejemplo, las aplicaciones de procesamiento de señales de radio, de simulaciones de físicas, etc.).

Hay dos maneras de añadir nuestros componentes NDK a una aplicación Android:

- Implementando nuestra aplicación de manera tradicional, con las API del SDK de Android, y utilizando JNI para hacer llamadas a nuestros componentes NDK. Está disponible desde la versión 1.5 de Android.

- Implementando las clases directamente nativas, utilizando la clase NativeActivity. Su definición cambia en el manifiesto, pero el ciclo de vida no. Está disponible desde la versión 2.3 de Android.

JNI

El Java *native interface* (JNI) es un *framework* de programación que permite que un programa escrito en Java y ejecutado en una máquina virtual pueda acceder a programas escritos en otros lenguajes (como C, C++, Python, etc.).

Dentro del NDK se encuentran una serie de herramientas, tales como compiladores, ensambladores, etc., para generar binarios para la arquitectura ARM en diferentes plataformas de desarrollo (como Linux, Windows o OS X).

4.3. *Location based application*

Location base applications o *location based services* (LBS) son aplicaciones o servicios basados en la posición geográfica del usuario.

Para determinar la posición geográfica de un dispositivo móvil en vuestra aplicación, podéis usar GPS, o bien la localización basada en las celdas de telefonía y las señales WiFi. A estas posibilidades se las conoce como LocationProvider. Las aplicaciones que quieren conocer la geoposición se pueden basar en una o en ambas alternativas.

El *global positioning system* (GPS) o sistema de posicionamiento global es un sistema que sirve para determinar la posición de un objeto, persona o vehículo en cualquier parte del mundo. Es un sistema de gran precisión, ya que su margen de error puede ser únicamente de centímetros, aunque normalmente hablamos de metros.

Desarrollo del GPS

El sistema GPS fue desarrollado e instalado por el Departamento de Defensa de los Estados Unidos. Para poder dar este servicio, existen decenas de satélites alrededor de la Tierra (a una distancia de más de veinte mil kilómetros).

Hay otras alternativas muy similares desarrolladas por otras potencias mundiales, como el GLONASS, de la Federación Rusa, o el Galileo, de la Unión Europea.

El GPS es más preciso, pero presenta algunos inconvenientes, precisamente donde la localización vía WiFi o las celda de telefonía presentan ventajas. Con GPS, el consumo de batería es mayor y el tiempo para conseguir la posición es mayor.

La precisión de las posiciones, así como el consumo de batería asociado, puede verse afectado por factores como el movimiento del usuario, el cambio de proveedor de localización, los cambios de la precisión de la señal, etc. Todo esto lo tiene que tener en cuenta vuestra aplicación, que tendrá que actuar en consecuencia para evitar al máximo el uso inapropiado del servicio.

La manera de conocer la localización en Android es mediante la suscripción a LocationListener, a los *events* relacionados con la localización. Los objetos que implementan la interfaz LocationListener deben implementar los siguien-

tes métodos: onLocationChanged, onProviderDisabled, onProviderEnabled y onStatusChanged. Estos sirven para escuchar cambios en el estado y en la disponibilidad de los proveedores de localización que nos interesen.

Este objeto, que sirve para recibir las actualizaciones, se debe registrar en el sistema mediante el método requestLocationUpdates, donde debéis registrar el proveedor que os interesa (GPS_PROVIDER o NETWORK_PROVIDER para la localización basada en celdas y redes WiFi), así como el tiempo y la distancia entre actualizaciones. Finalmente, debéis añadir vuestro objeto para recibir dichas actualizaciones.

Un método interesante es el de poder conocer la última localización conocida: getLastKnownLocation, que evita que tengamos que esperar a los proveedores de localización y, además, nos ayuda a ahorrar batería, pero con la correspondiente pérdida de precisión.

En cuanto a todas estas localizaciones que estamos mencionando (que pertenecen a la clase *location*), podréis saber cuándo fueron capturadas y mediante qué proveedor. De esta manera, podréis decidir si es suficiente con esta localización o queréis una nueva.

Además de la posibilidad de interaccionar directamente con los proveedores de localización, existe la posibilidad de utilizar librerías más propietarias, como la API de Google Maps, proporcionada por Google. Esta librería proporciona un ViewGroup llamado MapView, que se muestra como los mapas propios de Google Maps y utiliza los datos de su servicio, así como su interfaz de usuario. Incluye la gestión de la posición y las capas propias de dicha aplicación. Podéis programar sobre eso para añadir nuevas capas con vuestra información, pero para poder hacerlo es imprescindible que obtengáis una clave válida.

4.4. Comunicaciones

Existen diferentes opciones en Android para realizar comunicaciones. Es posible realizar comunicaciones de red normales mediante la API de comunicación de red o controlar el estado de nuestros proveedores de red con el método Context.getSystemService(Contex.CONNECTIVITY_SERVICE). También existe la posibilidad de utilizar API para propósitos concretos. Podemos destacar NFC, Bluetooth y SIP.

Como siempre, para cada una de estas conexiones es necesario que especifiquéis en vuestro manifiesto los permisos adecuados, ya que de lo contrario vuestra aplicación no funcionará.

4.4.1. Bluetooth

En Android, programando para trabajar con Bluetooth, se puede:

- Escanear para encontrar otros dispositivos.
- Pedir al adaptador de Bluetooth local por dispositivos ya enlazados.
- Establecer canales RFCOMM.
- Conectar con otros dispositivos mediante el servicio de *discovery*.
- Transferir datos a otros y desde otros dispositivos
- Gestionar múltiples conexiones.

Para trabajar con Bluetooth y conseguir las acciones básicas (como activar Bluetooth, encontrar dispositivos ya enlazado o nuevos, y transferir datos), Android nos proporciona una serie de clases clave:

- **BluetoothAdapter**: Es el principal punto de acceso en la interacción. Sirve para encontrar, por ejemplo, dispositivos ya enlazados o no, y para crear *sockets* para comunicaciones con otros dispositivos.

- **BluetoothDevice**: Representa un dispositivo al que se puede interrogar sobre información o contra el que se pueden abrir conexiones.

- **BluetoothSocket**: Es el canal de comunicaciones para intercambiar información.

- **BluetoothServerSocket**: Sirve para recibir conexiones cuando nuestro dispositivo está en modo servidor.

En muchas de las interacciones de nuestra aplicación con Bluetooth, Android mostrará diálogos de información para confirmar las acciones.

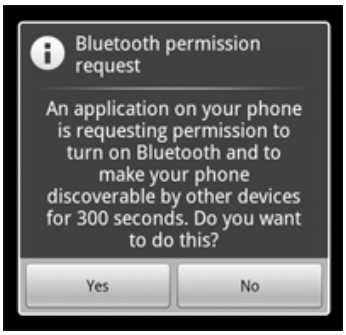


Imagen de mensaje de Android que avisa de acciones provocadas por nuestra aplicación
Fuente: http://developer.android.com

4.4.2. NFC

NFC a diferencia de otras tecnologías de comunicación por radiofrecuencias, como Bluetooth o WiFi, permite realizar las conexiones entre dos dispositivos sin necesidad de descubrir estos dispositivos ni enlazarlos; las interacciones pueden iniciarse solo con una acción.

NFC sirve para que nuestro dispositivo sea capaz de interaccionar con Tags NFC. Nuestro dispositivo podrá leer la información de estos *tags* o, incluso, interaccionar. Estos *tags* tienen diferentes tecnologías y pueden ser desde muy sencillos, con información textual, hasta muy complejos, con capacidades de cálculos o encriptación. Incluso un dispositivo Android puede simular ser un *tag*.

Para poder trabajar con NFC, Android proporciona las siguientes clases:

- **NfcManger**: Sirve para gestionar varios adaptadores físicos de nuestros dispositivo en NFC. Como normalmente solo se dispone de uno, se puede usar el método estático getDefaultAdapter.

- **NfcAdapter**: Representa nuestro adaptador al dispositivo NFC. Sirve para registrar cambios de *tag* hacia *activity*, y para trabajar con mensajes NFED.

- **NfcMessage y NfcRecord**: NfcMessage es un contenedor de cero o más NfcRecords. Cada NfcRecord tiene información NDEF.

- **Tag**: Representa al objeto pasivo de NFC. Estos *tags* pueden tener diferentes tecnologías y se pueden preguntar mediante el método getTechList().

NDEF

NDEF es una estructura de datos definida por el NFC Forum para almacenar, de manera eficiente, datos en los Tags NFC.

4.4.3. SIP

Session initiation protocol (SIP) es un protocolo que sirve para controlar las sesiones de comunicaciones multimedia, como por ejemplo las de llamadas de voz o de vídeo sobre IP. El protocolo se puede usar para crear, modificar y finalizar sesiones entre dos puntos (*unicast*) o entre múltiples puntos (*multicast*). En las modificaciones se puede cambiar el número de puntos implicados o los *streamings*.

En Android existen una serie de clases que ayudan a trabajar con este protocolo y, por tanto, facilitan su desarrollo. Esto os permitirá realizar videoconferencias en vuestras aplicaciones, así como usar mensajería instantánea.

Debéis tener en cuenta algunos requisitos y limitaciones que existen en el desarrollo de SIP para Android:

- Solo está disponible para dispositivos 2.3 o superiores.
- SIP funciona con una red de datos inalámbricos y, por tanto, no se puede probar en los emuladores, solo en dispositivos físicos.
- Cada miembro de la comunicación necesita una cuenta SIP (identificada en un servidor SIP).

Hay una serie de clases clave, que son las principales encargadas de trabajar con el protocolo:

- **SipManager**: Sirve para trabajar con las tareas SIP (como iniciar conexiones o dar acceso a servicios SIP).
- **SipProfile**: Define un perfil SIP e incluye la información del servidor y de la cuenta.
- **SipAudioCall**: Sirve para gestionar una llamada sobre IP mediante SIP.

4.4.4. *Widgets*

Los *widgets* son aplicaciones en miniatura que permiten mostrar partes de una aplicación dentro del escritorio o pantalla de inicio.

Ejemplo de *widgets* en la pantalla de inicio de una tableta PC y de un smartphone

Se puede programar la búsqueda de actualizaciones de los *widgets*, que se lleva a cabo a través de un tipo especial de *broadcast receiver*, con la periodicidad que se crea conveniente.

Hay tres elementos básicos para la construcción de un *widget*:

- AppWidgetProviderInfo. Sirve para definir, generalmente a través del Android Manifest, la información de configuración del *widget*. Entre esta información destaca el tamaño mínimo (alto y ancho), el periodo de actualización, el *layout* visual (definido de igual manera que en el caso de las Activity) y la opción de añadir una Activity sólo dedicada a configurarlo.

- AppWidgetProvider. El encargado de actualizar el *widget* según los eventos que vayan llegando. Algunos de los eventos importantes que tiene que gestionar son los de actualización (onUpdate), activación (onEnabled), borrado (onDelete). El más importante es el de actualización, que normalmente se realiza desde un *service* para evitar bloqueos y el consiguiente mensaje de error de una aplicación que tarda mucho en ejecutarse (ANR: *application not responding*).

- El *layout* propio, que ha sido configurado anteriormente en el primer punto. Se define de igual manera que el caso de las Activity: utilizando un fichero XML y los *views* tradicionales.

Atención

Debemos ser cuidadosos al programar las búsquedas de actualizaciones porque los *widgets* también se actualizan mientras el dispositivo está apagado, lo que puede consumir mucha batería.

Podéis obtener más información sobre los *widgets* en la siguiente página:

http://developer.Android.com/guide/topics/appwidgets/index.html

4.4.5. Sincronización de datos

Android permite crear sistemas de sincronización de datos entre nuestro dispositivo y otros puntos, como pueden ser los servidores propios o servidores en la nube.

Como requisito para poder sincronizar datos, es imprescindible tener una cuenta de usuario de Android. Generalmente, se suele tener al menos la cuenta de Google, que nos permite sincronizar contactos, calendarios, etc. Además, podemos crear nuestros propios servicios de usuario a través del AccountManager y el AbstractAccountAuthenticator, lo que nos daría acceso a cuentas de otros servicios.

El proceso de sincronización puede ser lanzado de manera automática por parte del propio sistema operativo, lo cual es ideal para mantener todas las aplicaciones sincronizadas desde un punto central. Para ello, tendremos que desarrollar un *service* que se encargará de llamar a un adaptador (SyncAdapter), que será quien realice la lógica de la sincronización. Este *service* de sincronización, a través del SyncAdapter, deberá trabajar con algún *ContentProvider*, para leer los datos a sincronizar y guardarlos posteriormente.

Ejemplo

Un caso muy habitual es sincronizar los datos de nuestros contactos con alguna red social y añadir actualizaciones de estos contactos a las redes sociales.

Para más información, tenemos un ejemplo de implementación disponible en http://developer.Android.com/resources/samples/SampleSyncAdapter/index.html y también como parte de los ejemplos del SDK.

5. Herramientas de desarrollo de Android

Hasta ahora hemos visto la potencia de nuestras aplicaciones y hemos podido definir acceso a prácticamente todas las partes de nuestro dispositivo en nuestra aplicación, con varias maneras de realizar la misma acción (por ejemplo, mediante ficheros XML o en código). Pero muchas veces hay que tener en cuenta mucha información, y hacerlo sin ayuda es difícil.

Para ello, en este apartado os mostraremos las principales herramientas de las que disponéis. Veréis las herramientas estándar del SDK de Android (algunas de ellas en profundidad, pero no todas, ya que además de que se escapan del alcance de este apartado, se añaden constantemente nuevas herramientas).

Después veréis los entornos de desarrollo principales, con especial antención en el *plugin* oficial de Google para desarrollo.

Finalmente, veréis algunas herramientas externas que os permitirán conocer la potencia de vuestra plataforma.

5.1. SDK de Android

Algunas de las herramientas que vienen con el SDK[7] oficial de Android son generales y se usan en cada una de nuestras aplicaciones. En cambio, algunas son específicas para el tipo de desarrollo que deseamos hacer, o las que habíamos pensado para la aplicación.

(7)SDK (*Software development kit*): kit de desarrollo de *software*

Algunas de las más interesantes pueden ser:

- **Sqlite3**: Herramientas para gestionar bases de datos mediante sentencias SQL.

- **Hprof-conv y dmtracedump**: Herramientas para hacer perfilado (en inglés, *profiling)* de nuestra aplicación; es decir, para mejorar el rendimiento mediante la detección de posibles problemas de memoria o de CPU.

- **Layoutopt y Draw 9-patch**: Herramientas para mejorar nuestra interfaz gráfica. La primera, para mejorar y analizar el rendimiento de nuestros *layouts* y, así, optimizar su rendimiento. La segunda, para la creación de gráficos.

- **Monkey y monkeyrunner**: Herramientas para realizar tests de estrés sobre nuestra aplicación.

- **Zipalign**, **ProGuard**: Herramientas para mejorar los ficheros correspondientes a nuestra aplicación.

- **Emulator**: El emulador donde podemos probar nuestras aplicaciones.

- **Logcat**: Visualizador de los *logs* de sistema de un dispositivo o emulador.

- ***Android***: Gestor de los dispositivos virtuales o AVD.

- **Adb**: Herramienta para poder ver el estado de nuestro emulador.

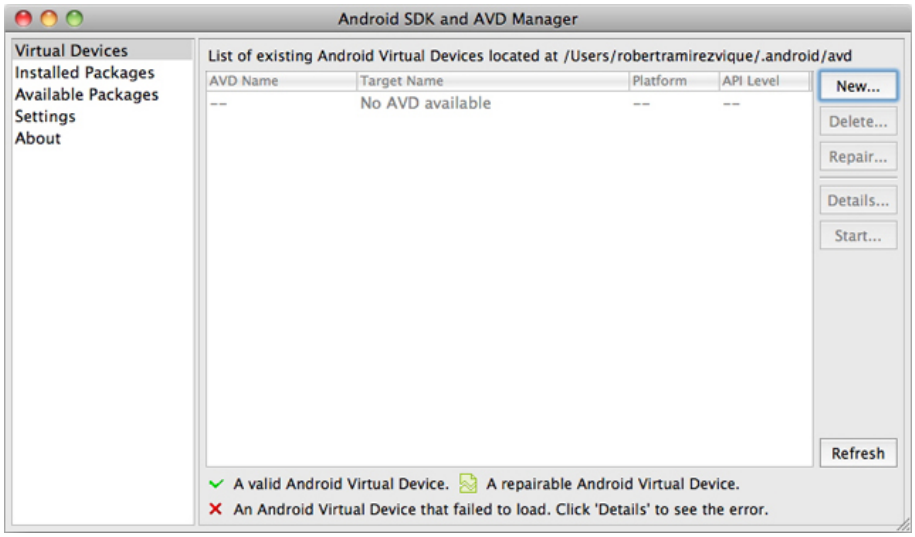
Algunos se usan constantemente y requieren de una explicación más detallada.

Para empezar a desarrollar, lo primero que debéis hacer es crear un AVD o dispositivo virtual, que corresponde a una configuración del emulador, y definir las opciones de *hardware* y *software* que queréis que tenga dicho emulador. Si no disponéis de una herramienta más potente, como el AVD Manager, debéis hacer líneas de comandos (con el comando android se pueden listar, crear, borrar o actualizar los AVD disponibles). También lo podéis hacer desde el AVD Manager, que os permite hacer lo mismo, tal como podéis ver en las imágenes siguientes.

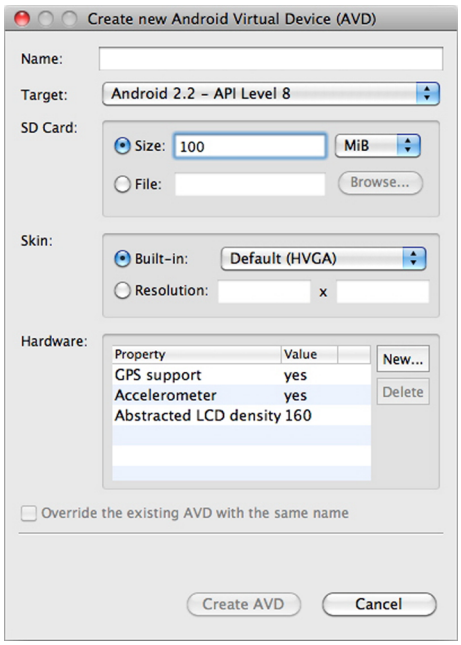
Ayuda de la orden android

Para conocer las opciones exactas de la orden android, podéis ver la ayuda con android –help.

Vista del AVD Manager



Creación de un nuevo AVD



Además, os permite bajar nuevas versiones de las librerías del SDK, de manera que siempre podéis actualizar desde aquí, antes de la aparición de una nueva versión para desarrollo.

Una herramienta importante es el adb, que se instala en el emulador o dispositivo y permite ver información de estado de la aplicación, hacer volcados de estado para su análisis, acceder a los *logs e*, incluso, debugar la aplicación o tener un entorno de comandos del emulador.

> Es importante que tengáis una trazabilidad de las acciones que van sucediendo. Para esto, podéis utilizar en vuestro código la clase android.util.Log con los métodos para escribir sobre el *log* según la verbosidad que se desee. Para ver esta información, podéis acceder a ella con el subcomando logcat del comando adb.

Sin duda, una de las herramientas esenciales es el emulador de Android, accesible con el comando android. Con este emulador podéis emular muchas acciones de usuario, muchas de ellas directamente con combinaciones de teclas (desde acceder al menú o abrir la cámara, hasta cambiar la orientación de la pantalla). También hay algunas otras que se pueden hacer desde la consola del emulador, donde podemos cambiar la velocidad de la red, simular SMS, cambiar la posición geográfica, etc.

La mayoría de estas herramientas son de entorno de comandos y requieren una serie de parámetros específicos para trabajar. Además, cambian de contexto constantemente para poder realizar trabajos habituales. Para mejorar esto, muchas de estas herramientas se agrupan en los *plugins* o extensiones de desarrollo para Android, de los cuales cabe destacar el que tiene soporte oficial de Google: ADT *plugin for Eclipse*.

Otros entornos de desarrollo

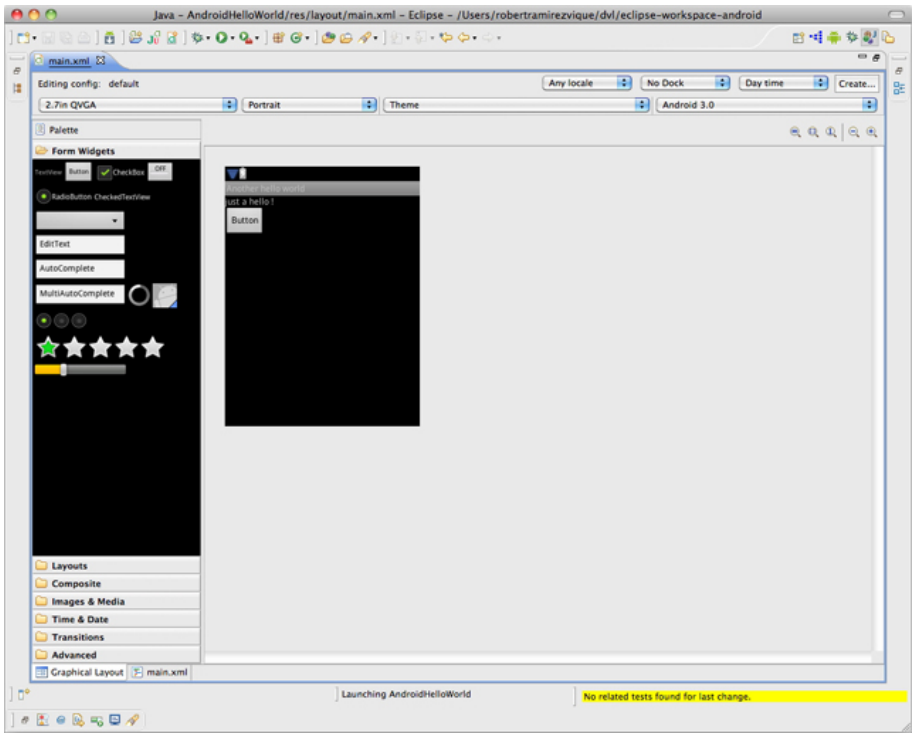
También existen otros entornos de desarrollo que dan soporte al desarrollo basado en Android (por ejemplo, el Netbeans de Oracle o el IndelliJ IDEA de JetBrains).

5.2. Plugin ADT para Eclipse

El plugin de Eclipse incorpora la gran mayoría de las herramientas previamente vistas y de manera transparente para el desarrollador. Así, se puede realizar un *debugging* de una aplicación en un dispositivo o emulador, o ver la información de perfilado de la memoria o el sistema de ficheros actual y trabajar con él. Todo esto está integrado dentro del IDE Eclipse.

Otra utilidad que nos ofrece la interfaz de desarrollo de android es la gestión integrada de los ficheros XML mediante editores gráficos. Uno de los más interesantes es la edición gráfica de los las interfaces gráficas, como podéis ver en la figura siguiente.

Edición de interfaz gráfica desde Eclipse



Esta edición lleva incorporada, además, la gestión de las diferentes versiones de los recursos alternativos, que dependen de cada característica, lo que que facilita mucho el desarrollo.

5.3. Depurando la aplicación en dispositivo real

A pesar de que nuestro emulador es una herramienta potente, hay aspectos que no se pueden probar del todo, como los *events* de *multi-touch* o algunos sensores, o conexiones como Bluetooth, o sensores como el acelerómetro o similar. Existen algunas herramientas en la comunidad para mejorar esta falta, pero la mayor seguridad la proporcionan, sin duda, las pruebas sobre el dispositivo real.

Además, siempre es deseable que realicéis pruebas lo más reales posibles con el dispositivo objetivo para poder apreciar también la responsabilidad de vuestra aplicación.

Para depurar vuestra aplicación sobre un dispositivo real, debéis hacer los siguiente:

- Marcar vuestra aplicación como *debuggable*. Para ello debemos fijar el atributo *Android:debuggable* a valor *true*.

- Permitir que se conecte el *debugger* en vuestro dispositivo En vuestro dispositivo, id a la pantalla principal, pulsad MENU, después Applications > Development > USB debugging.

- Hacer que vuestro sistema operativo reconozca el dispositivo. Dependerá del sistema operativo que estéis usando para desarrollar.

Finalmente, podéis lanzar la aplicación o depurar como lo haríamos normalmente (por ejemplo, desde Eclipse), y dispondréis de la misma información que si tuvierais un emulador.

5.4. Herramientas de testeo

Android, al igual que el resto de desarrollos móviles, requiere de muchas pruebas para conseguir un *software* de calidad (pruebas con herramientas de testeo y pruebas en dispositivos reales).

Hay algunas herramientas que están integradas en el propio SDK, basadas, sobre todo, en JUnit, que es una librería típica de testeo unitario para Java.

5.4.1. Herramientas de testeo incluidas en el SDK

Las herramientas incluidas para testeo en el SDK de Android se pueden dividir en las siguientes:

- Tests unitarios normales, basados en las librerías de JUnit, pero sin utilizar ni instalar objetos especiales de Android. Sirven para probar la lógica de aplicación, principalmente.

- Tests unitarios que utilicen la infraestructura de Android.

- Tests unitarios de estrés.

Los primeros son ejecuciones normales de test unitarios, que se basan en probar que la lógica hace lo que debe hacer y en comprobarlo con *asserts* tradicionales de JUnit. Es decir, comprueban que el resultado de la ejecución es el esperado o está dentro de los rangos esperados. En caso de ser cierto, continúan hasta acabar y, por tanto, el test es válido. En caso de no serlo, el test es fallido.

El segundo caso tiene una especificidad más cercana a Android, pues permite interceptar o provocar los *events* del ciclo de vida de una *activity*, por lo que se pueden probar cosas para una aplicación y su estado se recupera correctamente. Esto lo hace gracias a la API *instrumentation*.

Si realizáis los tests con la infraestructura de Android, se os proporcionan muchas clases base para vuestros tests, como ActivityTestCase, ServiceTestCase, ApplicationTestCase, etc. Además de ellos, hay muchos objetos Mock que permiten emular el comportamiento estándar de vuestra aplicación para que sea más fácil probar la aplicación de manera aislada. La clase que se encarga de ejecutar los test se llama InstrumentationTestRunner, que os permite interaccionar de una manera más directa sobre las clases de vuestra aplicación.

Finalmente, existe una herramienta que sirve para probar la aplicación de manera automática y pseudoaleatoria. Esta herramienta se llama Monkey (tiene la metáfora de un mono tocando el dispositivo), y lanza *events* de clic, *events* táctiles o gestos, además de *events* de sistema.

A Monkey se le pueden definir las siguientes opciones:

- número de *events* a lanzar
- limitaciones (como solo lanzar *events* hacia un paquete de código)
- tipo y frecuencia de los *events*
- opciones de *debugging*

A partir de este momento, podéis lanzar los test. La herramienta os avisará en los siguientes casos:

- Si queréis testear solo un paquete, y se intenta ejecutar alguna parte fuera de este paquete, la aplicación bloquea esta ejecución, con las posibles consecuencias.

- Si la aplicación tiene un problema o una excepción no controlada, la herramienta parará e informará sobre el error.

- Si la aplicación provoca que aparezca un error del tipo "*Application not responding*", la herramienta parará e informará sobre el error.

Con esta herramienta podéis comprobar si vuestra aplicación se comporta correctamente ante situaciones de estrés.

5.4.2. Herramientas externas de testeo

Además de estas herramientas, existen algunas herramientas externas que ayudan a realizar tests más específicos, por ejemplo:

- **Roboelectric**: Tests unitarios para cada *activity* de Android.

- **Robotium**: Tests de aceptación. Es decir, se lanza la aplicación y vuestro test va realizando acciones como si fuera un usuario normal. Si este test funciona, significa que la parte probada de vuestra aplicación está aceptada y funcionando.

Enlaces de interés

Para saber más sobre Roboelectric, podéis visitar los siguientes sitios web:

http://pivotal.github.com/robolectric

Para saber más sobre Robotium podéis visitar los siguientes sitios web:

http://code.google.com/p/robotium

Roboelectric permite ejecutar tests unitarios, como en el caso de los tests de *activities* lanzados desde las herramientas de Android, pero sin necesidad de lanzar un emulador. Esto hace que los tests sean mucho más rápidos.

En cambio, Robotium sirve para hacer pruebas automatizadas del tipo de caja negra; es decir, tests en los que no sabemos qué hay dentro de la aplicación, pero que podemos probar como si fuéramos usuarios, de modo que recibiremos respuestas como tales.

Lo que intentan este tipo de tests es lanzar *events* como si fuéramos un usuario, sin necesidad de conocer los elementos (Spinners, Buttons, TextBoxes, etc.) que existen en el código de las *activities*. Por tanto, se basan mucho en lo que el usuario ve.

Ejemplo

En este ejemplo, se persigue realizar cambios navegando por los menús. Como veis, no especifica el tipo de objeto sobre el que hacemos clic, sino el texto que contiene.

```
public class EditorTest extends
             ActivityInstrumentationTestCase2<EditorActivity> {

...

 public void testPreferenceIsSaved() throws Exception {

           solo.sendKey(Solo.MENU);
           solo.clickOnText("More");
           solo.clickOnText("Preferences");
           solo.clickOnText("Edit File Extensions");
           Assert.assertTrue(solo.searchText("rtf"));
```

```
            solo.clickOnText("txt");
            solo.clearEditText(2);
            solo.enterText(2, "robotium");
            solo.clickOnButton("Save");
            solo.goBack();
            solo.clickOnText("Edit File Extensions");
            Assert.assertTrue(solo.searchText
            ("application/robotium"));
    }
...
}
```

Estos tests son muy potentes, ya que no requieren del conocimiento de la aplicación (como en el resto de los casos) y, además, prueban la aplicación al completo. Como contrapartida, presentan algunos problemas (dejan de funcionar con pequeños cambios de la interfaz, realizan cambios en el texto y son lentos de construir y de ejecutar).

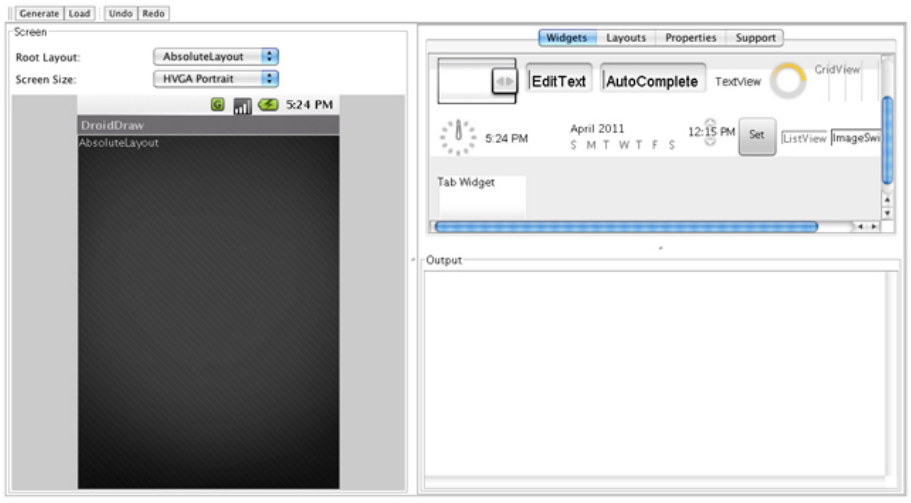
5.5. Otras herramientas

También existen otras herramientas generadas por Google o por la comunidad. Entre ellas destacamos DroidDraw, App Inventor y Sensor Simulator.

5.5.1. DroidDraw

DroidDraw es un herramienta que sirve para generar, de manera sencilla, interfaces visuales de Android en formato XML. La peculiaridad de esta herramienta es que se puede ejecutar dentro de nuestro navegador o de manera nativa en el sistema operativo del desarrollador.

Ejemplo de edición de un *layout* dentro de DroidDraw

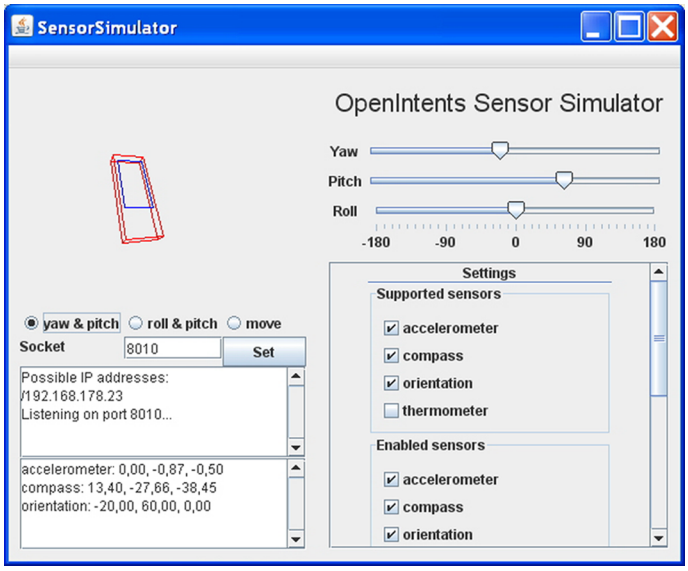


La aplicación permite descargar en el dispositivo una versión previa de la interfaz creada en nuestro dispositivo. Para ello se utiliza la aplicación para el móvil llamada AnDroidDraw.

5.5.2. Sensor Simulator

Sensor Simulator es una herramienta creada por la comunidad para poder simular los sensores de nuestro emulador o dispositivo. El objetivo es poder simular cambios de orientación (o de temperatura, etc.). Para ello se necesita un servidor en funcionamiento en nuestro entorno de desarrollo y una aplicación instalada en el emulador.

Uso de Sensor Simulator para cambiar datos de los sensores

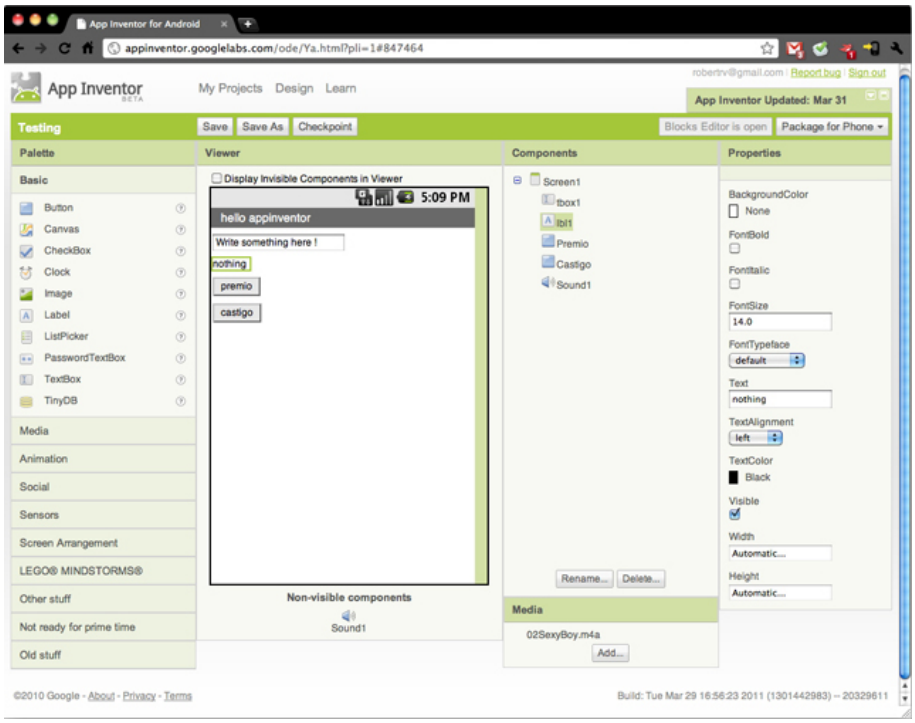


5.5.3. App Inventor

App Inventor es una aplicación "en línea" de Google cuyo propósito es que personas sin conocimientos de programación sean capaces de programar una aplicación sencilla y descargarla y ejecutarla en su móvil.

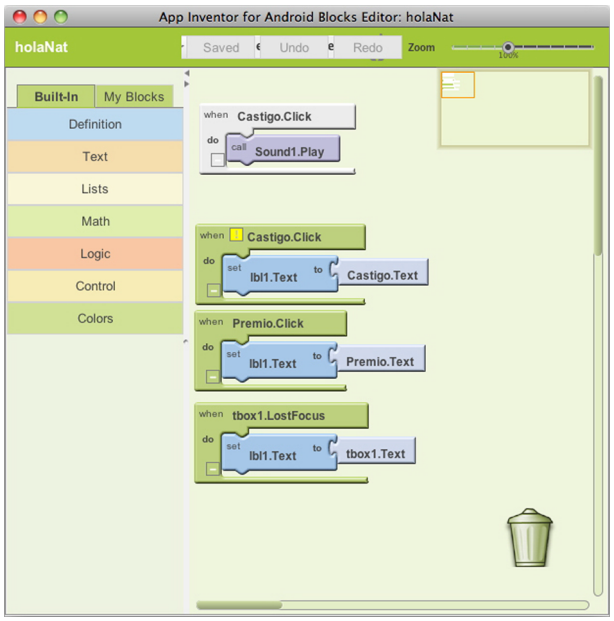
Para facilitar la edición visual, incorpora un editor gráfico con capacidades reducidas y algunas peculiaridades que hacen más sencillo su uso, al estilo de DroidDraw:

Ejemplo de edición de la interfaz gráfica con App Inventor



Además, para programar la lógica de la aplicación, utiliza objetos que representan las formas de control más típicas de una aplicación móvil:

Ejemplo de edición de una lógica de aplicación utilizando App Inventor



6. Distribución y negocio

Una vez terminada vuestra aplicación (o, al menos, una versión), es el momento de que la distribuyáis.

En el caso de Android, se puede distribuir directamente el fichero .apk correspondiente a la aplicación, pero también se puede hacer mediante alguno de los mercados de aplicaciones (*markets*) o, por ejemplo, mediante el mercado oficial de Google. En estos mercados podéis ofrecer la opción de vender la aplicación y conseguir remuneración por ella, pero lo más importante es que estos mercados abren el nuevo canal de distribución de vuestra aplicación.

Market oficial de Google

El *market* oficial de Google, como aplicación de Android, solo está disponible en los dispositivos verificados por Google. Dado que Android se puede utilizar libremente según su licencia, se puede usar en otros dispositivos.

También existen otros métodos de remuneración por las aplicaciones, como la publicidad o los pagos en la propia aplicación.

6.1. Firma de la aplicación

Firmar una aplicación es un proceso en el que se utiliza un certificado digital privado para marcar la aplicación, de manera que se puede saber de forma unívoca quién es el autor de la aplicación. Esta unicidad se valida mediante la clave pública correspondiente.

Para poder utilizar vuestra aplicación en un dispositivo físico, siempre debe estar firmada, bien con una clave definitiva, bien con una de prueba. El certificado, a diferencia de los utilizados en los navegadores, no necesita estar certificado por una entidad certificadora, sino que puede estar generado por el propio desarrollador. A pesar de esto, los certificados deben ser auténticos y vigentes, aunque solo se comprueba su fecha de validez la primera vez que se instala la aplicación.

Entidad certificadora

Una entidad certificadora (CA) emite certificados digitales (verificando el origen y su validez). VeriSign y Thawte son dos ejemplos de entidades certificadoras.

Es importante que la firma de la aplicación se haga con un certificado que no caduque durante la vida de la aplicación. Si sucede, deberéis asignar un nuevo paquete a vuestra aplicación para poder firmarla correctamente y, finalmente, el usuario deberá instalarla como una aplicación diferente. En concreto, para instalar la aplicación mediante el *market* oficial de Android, se debe firmar la aplicación con un certificado cuya fecha de final de validez sea posterior al 22 de octubre del 2033.

Los certificados se pueden utilizar para firmar más de una aplicación. Esto puede ocurrir en varios casos:

- Dos aplicaciones quieren compartir información en tiempo de ejecución. Esto es debido a que dos aplicaciones con el mismo certificado se pueden ejecutar dentro del mismo proceso y, por tanto, tienen los mismos recursos de sistema, como si fueran un único proceso. Con esto se pueden tener aplicaciones distintas que interactúan intensamente.

- Modularizar nuestra aplicación. Si tenéis vuestra aplicación dividida en partes y queréis que cada una de estas partes se pueda actualizar por separado, una manera de hacerlo es realizar aplicaciones distintas, pero firmadas con el mismo certificado.

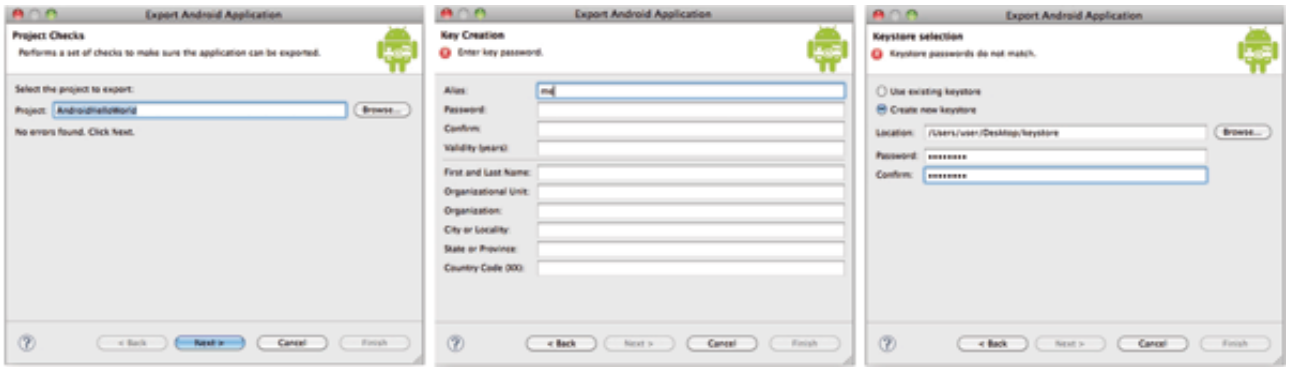
Las firmas de *debugging* se pueden generar con las herramientas estándar del JDK. En el caso de trabajar con otro IDE (como Eclipse), este hará el trabajo por nosotros.

A la hora de generar una versión pública deberemos:

1) Obtener una clave privada.
2) Compilar la aplicación para su distribución.
3) Firmar la aplicación con la clave obtenida en el primer paso.
4) Alinear el paquete APK.

Estas fases, si no disponéis de un IDE, se realizan mediante diferentes comandos (como keytool, jarsigner, ant, zipalign). Si disponéis de un IDE (como Eclipse), simplemente deberéis exportar la aplicación para que sea distribuida. Las acciones de firmado y alineación son controladas por el *plugin* ADT de Eclipse.

Ejemplo de firmado y empaquetado de una aplicación mediante el *plugin* de Eclipse



Es importante que mantengáis en un lugar seguro las claves privadas con las que firmáis aplicaciones públicas, pues si alguien malintencionado las consigue, puede cambiar el código y realizar actualizaciones de vuestra aplicación sin vuestro permiso, pero todos los usuarios verán que es una actualización más.

6.2. Versionado de las aplicaciones

Las aplicaciones de Android permiten gestionar las versiones de la misma aplicación. Esto sirve para mantener información de lo que contiene cada aplicación, para pedir nuevos permisos en el caso de que sea necesario, o bien para que otras aplicaciones sean conscientes de la versión actual. Esto último es útil para evitar incompatibilidades entre versiones, de manera que otra aplicación puede saber con qué versión es compatible y preguntar cuál es la versión actual de dicha aplicación o (por lo general) servicio.

Las versión actual se define en el manifiesto con dos atributos:

- **android:versionCode**: Corresponde a un entero que representa la versión del código de la aplicación y es relativo a versiones previas.

- **android:versionName**: Es un texto que identifica, para los usuarios, el nombre de la versión. Puede ser algo (como sucede con Android) que identifique la versión de manera decimal (X.Y.Z), para identificar claramente el nivel de cambios de una versión respecto a otra.

Se puede obtener información de la versión de una aplicación concreta mediante la clase PackageManager y el método getPackageInfo(packageName, flags).

Es posible que queráis actualizar datos de una versión a otra, cosa que ocurre con las versiones del SDK que están soportadas, al igual que con todas las aplicaciones que nos permiten "setear" los requerimientos mínimos de la aplicación. Si actualizamos la versión y esta es incompatible o requiere nuevos permisos, Android tomará las acciones correspondientes para garantizar su correcto funcionamiento según las definiciones del manifiesto.

6.3. Consideraciones previas a publicación

Hay algunos puntos que debéis tener en cuenta antes publicar la aplicación para el usuario final:

- Debéis haber probado la aplicación con dispositivos reales. Como habéis visto anteriormente, existen herramientas en el SDK de Android (y fuera del mismo) que os ayudan a ello. Esto es importante para evitar la frustración de los usuarios y la posible mala reputación de nuestra aplicación.

- En el caso de que sea necesario, debéis añadir un contrato de licencia de usuario final (EULA), en inglés *end user license agreement*, para informar de las condiciones de nuestra aplicación y, en especial, de sus repercusiones.

- Si se trata de una aplicación no gratuita, considerad añadirle una licencia a la aplicación, de manera que podáis gestionara dentro del sistema de licencias de Android.

License verification library

Android aporta una librería llamada *license verification library* (LVL) para gestionar la propiedad de las licencias por parte de los usuarios. El proceso de verificación se hace mediante la aplicación del *market*, que se comunica con un servidor de licencias.

- Definir el icono y el nombre de la aplicación.

- Verificar que vuestros filtros del manifiesto son correctos, de manera que no se podrá instalar la aplicación en un dispositivo o tipo de dispositivo que haya sido probado.

- Eliminar del código correspondiente la depuración, ya sea de *logging* excesivo o de otras fuentes.

- Conseguir las claves propias para producción de librerías, como, por ejemplo, las claves de la API de Google Maps.

- Realizar la firma para distribuir la aplicación.

- Finalmente, probar la aplicación firmada en dispositivos reales.

6.4. Publicación y seguimiento

Cuando tengáis la aplicación preparada para su distribución, se puede transmitir de manera habitual (con el correspondiente fichero .apk) o en el *market*. Este canal de distribución añade mucho valor, por lo que es importante que lo conozcáis. Este canal ofrece la posibilidad de llegar a millones de potenciales clientes con una sola acción.

Para publicar en el *Android market* de Google, necesitáis una cuenta de Google Checkout. Además, debéis abrir una cuenta de desarrollador, que tiene un coste único de veinticinco dólares. A partir de ese momento, si vuestra aplicación es de pago, recibiréis en vuestra cuenta de Google Checkout el 70% de las ganancias directas o indirectas del *market*.

Google Checkout

Google Checkout es la herramienta de Google para realizar pagos y cobros electrónicos.

Las posibles opciones de publicación de una aplicación en el mercado y sus modelos de negocio son las siguientes:

- **Una aplicación gratuita**: Puede descargarse fácilmente y ser utilizar sin ninguna restricción. En este caso, suele tratarse de aplicaciones de soporte a negocios mayores fuera del entorno móvil, como bancos, publicidad bajo demanda, etc.

- **Una aplicación gratuita con publicidad**: Estas aplicaciones suelen ser muy populares, y lo que consiguen es que la aplicación sea totalmente gratuita. A cambio, el usuario debe ver ciertos anuncios, por ejemplo mediante AdMob.

- **Una aplicación de pago**: Son aplicaciones que requieren que el usuario pague por ellas. Por lo general, al usuario final se le da la posibilidad de probar la aplicación durante un corto periodo de tiempo, con la opción de devolverla y retirar el pago.

- **Una aplicación en versión *lite***: Este tipo de aplicaciones son versiones de aplicaciones de pago que tienen unas funcionalidades reducidas o eliminadas. El objetivo es atraer al usuario final y que este compre la versión de pago.

- **Pagos por bienes virtuales**: Se utiliza para pedir pagos en la propia aplicación, que son gestionados por ella. Android tiene actualmente una API para ello, llamada In-app billing. Podéis ver el esquema de pagos en siguiente imagen.

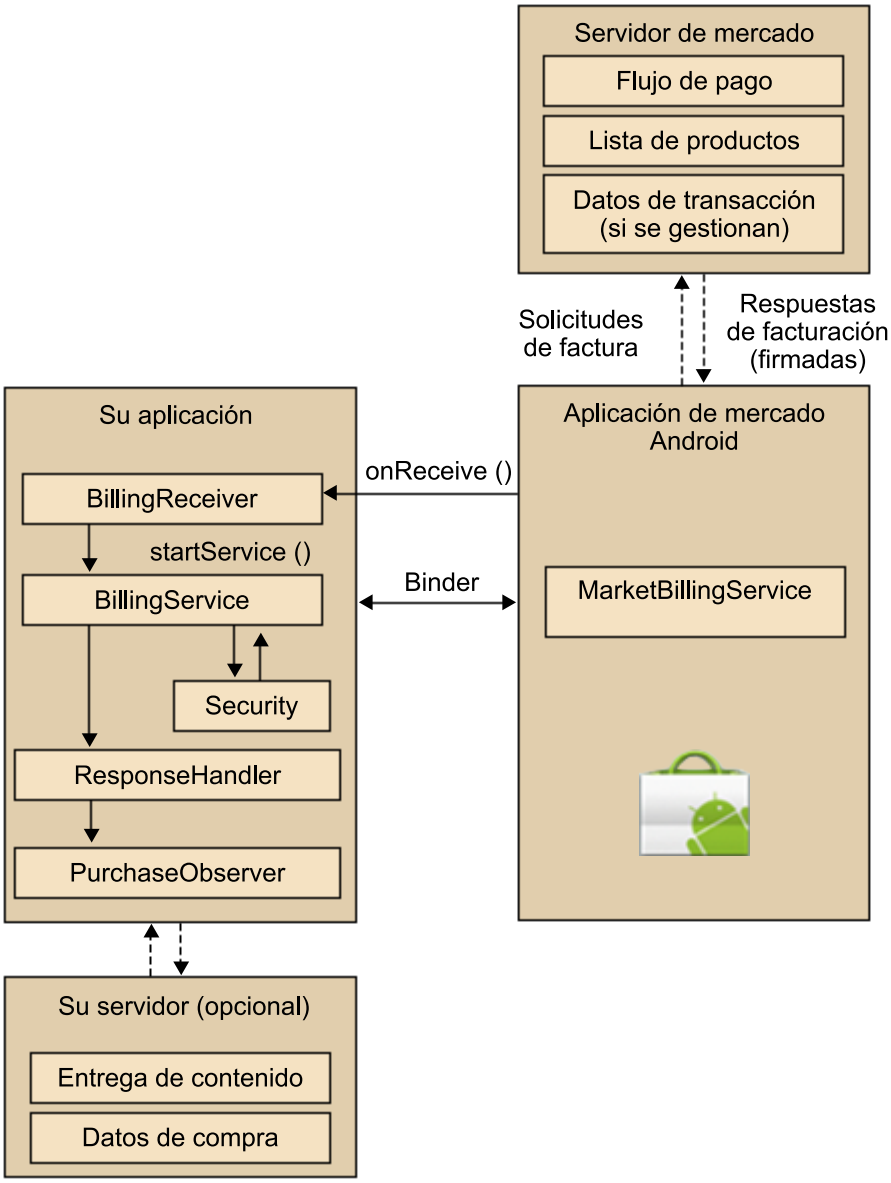
AdMob

AdMob es una compañía de publicidad móvil comprada por Google en el 2009. AdMob ofrece soluciones de anuncios para plataformas nativas como Android, iOS, webOS, Flash Lite y para todos los navegadores móvil estándar.

Periodo de prueba

En el 2010, Google cambió su mercado oficial; el tiempo permitido para probar una aplicación pasó de veinticuatros horas a quince minutos.

Explicación de la compra de bienes virtuales en una aplicación mediante el In-app billing



http://developer.android.com/

Todas las aplicaciones que hemos comentado se comercializan en un *market*. Podéis llegar al usuario de tres formas distintas:

- **Mediante los listados de aplicaciones más recientes**: Estas están organizadas por tipo de aplicación (juegos o aplicaciones) y categoría. En este listado de aplicaciones entran todas las nuevas aplicaciones, por lo que el tiempo de permanencia es limitado. Por ello, es importante elegir bien la categoría, sopesando el número de aplicaciones nuevas que van a llegar.

- **Mediante las búsquedas por *keywords***: Un usuario puede realizar una búsqueda en cualquier momento, por lo que debéis elegir bien las palabras e, incluso, incluir palabras internacionalizadas.

- **Mediante las aplicaciones *top***: Este listado depende de las descargas y de las valoraciones de los usuarios. Este listado es el más difícil de mante-

ner y, a la vez, el que puede dar mayores beneficios. Para conseguir que se mantenga, es importante que escuchéis a vuestros usuarios y realicéis las mejoras que sugieran. También es bueno que realicéis actualizaciones habitualmente.

En los mercados de aplicaciones, las aplicaciones suelen tener diversos componentes de promoción que hacen que el usuario se decante o no por ellas. Entre los más importantes están el nombre y la descripción, las imágenes o vídeos de la aplicación y, finalmente, los comentarios y las votaciones de los usuarios.

Estadísticas de la aplicación

Dentro del mercado, podréis seguir las estadísticas relacionadas con vuestra aplicación. Tendréis el historial de las instalaciones de la aplicación en el *market e* información de la distribución de los dispositivos, podréis ver el porcentaje de instalaciones que se realizan con cada versión del *framework* e, incluso, el dispositivo concreto al que corresponde. Esto puede ayudaros cuando prioricéis las actualizaciones o mejoras de vuestra aplicación.

Resumen

Este módulo didáctico introduce al estudiante a los principales conceptos del desarrollo móvil nativo basado en Android.

Introducimos el módulo con la historia reciente de esta tecnología, poniendo las bases sobre cómo se estructuran las aplicaciones.

Posteriormente hemos visto los principales componentes comunes a todas o gran parte de las aplicaciones Android, como son las fases del ciclo de vida, los *Intents*, el Manifiesto, etc.

A continuación hemos profundizado en la parte relacionada con la interfaz gráfica, viendo las posibilidades que ofrece Android para combatir la fragmentación. Y posteriormente hemos introducido al estudiante algunas de las partes más interesantes del SDK como son las aplicaciones LBS, los almacenes de datos, o tecnologías como NFC o SIP.

Finalmente hemos visto las herramientas que existen para facilitar su desarrollo, y también las herramientas necesarias para su distribución y comercialización.

Actividades

1. Hemos visto que las *activities* pueden volver al punto en que dejamos la *activity* en caso de ser interrumpida, pero hay algunas especiales que no tienen este comportamiento. ¿Cuáles son y por qué?

2. Cuando invocamos una nueva *activity* en la misma *task*, la *activity* suele estar asociada a la task y pertenece por completo a su pila de *activities*. Sin embargo, hay algunos casos en que esto no es del todo cierto. ¿Cuáles son estos casos y qué aportan?

3. Desarrollad pruebas de concepto para cada uno de los componentes explicados, desde las vistas más simples hasta las aplicaciones para la sincronización de datos.

4. Averiguad en qué se diferencian las nuevas versiones de Android para la gestión de dispositivos tipo tabletas PC de las antiguas.

5. Enumerad los tipos de componentes principales que se utilizan para el desarrollo de aplicaciones Android.

6. Identificad el comando que se utiliza para acceder al terminal de Android.

7. Haced una lista de los objetos utilizados para lanzar aplicaciones diferentes a la actual (como por ejemplo, enviar un correo) e indicad cómo se configuran.

Glosario

API *f* Sigla de *application programming interface*

bytecode *m* Código formado por instrucciones ejecutables independientes del hardware final de la máquina.

dalvik *f* Maquina virtual Java donde se ejecutan las aplicaciones Android.

debug *m* Proceso que identifica y corrige errores de programación.

driver *m* Componente de software responsable de ofrecer acceso a un componente hardware.

evento *m* Acción que se inicia generalmente fuera del ámbito de aplicación de un programa y que se maneja por un fragmento de código dentro del programa.

GPS *m* Sigla de *global positioning system*. El sistema de posicionamiento global determina la posición de un objeto, persona o vehículo en todo el mundo, pudiendo llegar a precisiones de centímetros, aunque normalmente hablamos de metros. Fue desarrollado e instalado por el Departamento de Defensa de Estados Unidos. Para poder dar este servicio se utilizan decenas de satélites que giran alrededor de la Tierra en una órbita superior a los 20.000 km.
Alternativas al GPS: GLONASS, de la Federación rusa y Galileo, de la Unión Europea.

jUnit *m* Conjunto de bibliotecas utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.

keywords *f* Palabras clave que pueden servir para identificar un objeto a través de otros conceptos, especialmente útil para búsquedas.

layout *m* Distribución visual de los objetos gráficos en la aplicación.

LBS *m* Sigla de *location based service*. Servicio basado en la posición geográfica del usuario.

listener *m* Parte del patrón de diseño Observer, que es notificado cuando sucede un evento para el cual ha registrado interés.

middleware *m* Capa de software encargada de independizar el desarrollo de los detalles que están por debajo.

modularizar *v tr* Dividir una aplicación en módulos.

multithreading *m* Aplicado a la industria móvil, quiere decir poder mantener varias aplicaciones ejecutándose a la vez.

Open GL *f* Especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

Open Handset Alliance *f* Consorcio de empresas: Broadcom Corporation, Google, HTC, Intel, LG,Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile y Texas Instruments.

renderización *f* Proceso de generación de una imagen desde un modelo.

retrollamada *f* Llamada a un método o función donde uno de los parámetros será una referencia al método o función.

SQL Lite *m* Motor de base de datos relacionales.

UI *f* Sigla de *user interface*. Interfaz gráfica o interfaz de comunicación con el usuario.

URI *m* Sigla de *universal resource identified*. Identificador único dentro de la aplicación. Las URL son un tipo de URI.

WebKit *m* Plataforma para aplicaciones que funciona como base para los navegadores Safari, Google Chrome, Epiphany o Midori, entre otros. Está basada originalmente en el motor de renderizado KHTML del navegador web del proyecto KDE, Konqueror.

XML *m* Sigla de *extensible markup language* (lenguaje de marcas extensible). Metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML, es a su vez un lenguaje definido por SGML). Por lo tanto,

XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

Bibliografía

Murphy, Mark L. (2009). *The Busy Coder's Guide to Android Development*. Commonware.

Steele, James; To, Nelson (2011). *The Android Developer's Cookbook*. Developer's Library.

Enlaces de Internet

http://developer.android.com/

http://columna80.wordpress.com/

http://code.google.com/p/openintents/wiki/SensorSimulator

http://appinventor.googlelabs.com/about/

http://www.droiddraw.org/

Seguridad en dispositivos móviles

Marc Domingo Prieto

PID_00178751

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

La seguridad en dispositivos móviles se ha convertido en un asunto muy importante debido al incremento de "ataques" recibidos y a las consecuencias que estos tienen. Los ataques vienen incentivados por la popularización de los dispositivos móviles, el aumento de información personal y confidencial que almacenan y las operaciones realizadas a través de ellos, como por ejemplo las bancarias.

Ataque

El ataque es un método mediante el cual un individuo intenta tomar el control, desestabilizar o dañar un dispositivo móvil.

Los dispositivos móviles están formados por un conjunto de componentes de hardware capaces de soportar una gran variedad de tecnologías inalámbricas (GSM, UMTS, Wifi, Bluetooth, etc.), donde destaca uno o varios procesadores de altas prestaciones que permiten ejecutar un sistema operativo muy complejo y un gran número de aplicaciones que requieren una gran capacidad de cálculo. Todo ello incrementa significativamente las distintas vulnerabilidades a las que están expuestos este tipo de dispositivos.

Un hardware más potente implica que pueden ser tratados más datos (normalmente personales), tanto los que se almacenan en la memoria de los dispositivos móviles como los que se reciben por los diferentes sensores que estos incorporan. Además, el hecho de soportar una gran variedad de tecnologías inalámbricas abre más vías de ataque.

Una mayor complejidad del sistema operativo también puede aumentar la vulnerabilidad de los dispositivos móviles. Cuando los sistemas crecen es más fácil que se produzca algún error en el software. Además, su peligrosidad aumenta debido a que todavía no somos conscientes de estos posibles problemas de seguridad.

error

En inglés, *bug*.

Este módulo no pretende ser un manual de seguridad ni recoger todos los riesgos existentes, sino introducir los conceptos y principios de seguridad en los dispositivos móviles. Este material ha de servir para que nos familiaricemos con los riesgos en los que están inmersos los dispositivos móviles, así como con las medidas de seguridad aplicables con el fin de reducir los daños causados por un ataque.

Empezaremos el módulo viendo algunos conceptos básicos de seguridad e identificando las diferentes capas donde los dispositivos móviles pueden implementar seguridad: comunicaciones inalámbricas, sistema operativo, aplicación y usuario. Posteriormente, analizaremos la seguridad en cada una de estas capas.

Objetivos

Con el estudio de este módulo se pretende que el estudiante alcance los objetivos siguientes:

1. Entender los conceptos básicos de seguridad.

2. Ver las medidas de seguridad que se utilizan en las tecnologías de comunicaciones inalámbricas utilizadas en los dispositivos móviles.

3. Conocer las medidas de seguridad que se aplican en el sistema operativo.

4. Revisar los riesgos que presentan algunas aplicaciones.

5. Comprender las consecuencias de una pérdida o robo de un dispositivo móvil y conocer los mecanismos para limitar sus efectos.

6. Saber cuáles son las prácticas de seguridad recomendadas cuando se utiliza un dispositivo móvil.

1. La problemática de la seguridad

1.1. Conceptos básicos de seguridad

Cuando hablamos de seguridad, existe una nomenclatura imprescindible para identificar el grado de protección que estamos utilizando. Si tomamos como ejemplo los diferentes pasos que se efectúan durante una llamada telefónica sobre una red inalámbrica, podemos identificar los cuatro conceptos clave de seguridad de la información:

- **Confidencialidad**: ha de ser posible que nadie pueda captar nuestra llamada y enterarse de lo que decimos.
- **Autenticación**: solo los usuarios con un teléfono de la red, es decir, pertenecientes a la compañía, pueden utilizar la red.
- **Integridad**: es necesario que la información, de voz o datos, que viaje por la red inalámbrica no se pueda alterar sin que se detecte.
- **No repudio**: debe ser imposible que un usuario que ha utilizado la red pueda negarlo; es decir, se debe garantizar que haya pruebas que demuestren que un usuario ha hecho una determinada llamada.

De manera más genérica, y al mismo tiempo formal, podemos definir estos conceptos del modo siguiente:

La **confidencialidad** es la propiedad que asegura que solo los que están autorizados tendrán acceso a la información. Esta propiedad también se conoce como *privacidad*.

La **integridad** es la propiedad que asegura la no alteración de la información. Por *alteración* entendemos cualquier acción de inserción, borrado o sustitución de la información.

La **autenticación** es la propiedad que hace referencia a la identificación. Es el nexo de unión entre la información y su emisor.

El **no repudio** es la propiedad que asegura que ninguna parte pueda negar ningún compromiso o acción realizados anteriormente.

Privacidad

El término *privacidad* es la traducción de la palabra inglesa *privacy*. Para el mismo concepto en inglés también se utiliza la palabra *secrecy*.

Hay que destacar que la propiedad de autenticación es quizá la más importante de las que acabamos de mencionar, ya que sirve de poco conseguir confidencialidad e integridad si resulta que el receptor de la información no es quien nosotros pensamos.

Estos cuatro conceptos básicos de seguridad de la información que acabamos de definir son la base de la totalidad de los requisitos de seguridad que se pueden necesitar, tanto si se trata de comunicaciones inalámbricas como de información en general.

1.2. Capas de seguridad en dispositivos móviles

Para poder entender la importancia de la seguridad en los dispositivos móviles hay que conocer las capacidades de estos dispositivos y cómo se ha llegado a ellas. Principalmente, los dispositivos móviles han vivido una importante revolución en cuanto a las aplicaciones que pueden ejecutar. Esta revolución ha estado marcada por tres motivos:

1) Un hardware potente, con muchos sensores.

2) Un sistema operativo complejo que facilita un SDK[1] sencillo y potente para los desarrolladores.

(1)SDK son las siglas en inglés de *software development kit*, traducido como kit de desarrollo de software.

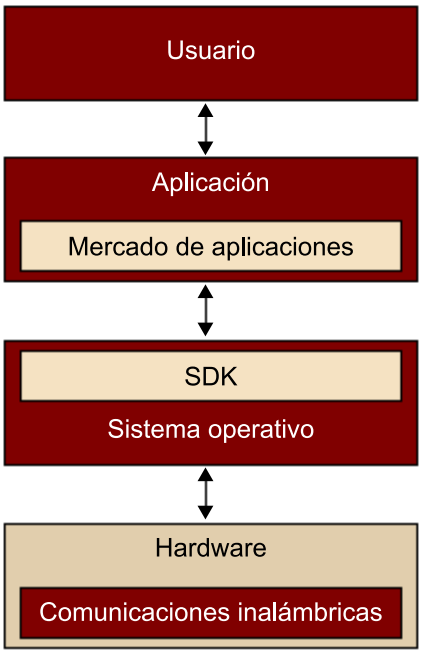
3) Un mercado de aplicaciones completamente integrado en el sistema y muy intuitivo, lo que facilita las transacciones tanto a los usuarios como a los desarrolladores.

Debido a estas nuevas funcionalidades que ofrecen los sistemas operativos para móviles y a las aplicaciones que se han creado sobre ellos, los dispositivos móviles acaban almacenando gran cantidad de datos, generalmente confidenciales. Ya no únicamente guardamos los números de teléfono de nuestros contactos, el registro de llamadas o los SMS, sino que también almacenamos una gran cantidad de información personal, como pueden ser cuentas bancarias, documentos o imágenes.

Este aumento de la información personal almacenada provoca que más personas puedan estar interesadas en obtenerla. Además, la complejidad actual de los sistemas operativos para móviles ha incrementado los agujeros de seguridad expuestos. Por lo tanto, cuando utilizamos dispositivos móviles, es recomendable seguir unas prácticas de seguridad, que serán parecidas a las utilizadas en los ordenadores.

Para poder analizar la seguridad de los dispositivos móviles de manera eficiente, se ha organizado este módulo en cuatro apartados: la comunicación inalámbrica, el sistema operativo, la aplicación y el usuario. Cada apartado

contendrá una pequeña introducción, una breve descripción tanto de los principales ataques como de los principales mecanismos de prevención y un caso de estudio.

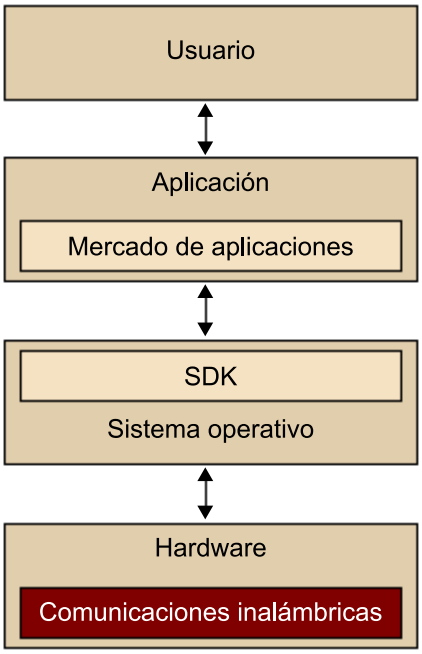


2. Comunicaciones inalámbricas

Las comunicaciones inalámbricas permiten que dos o más dispositivos móviles puedan comunicarse entre ellos sin necesidad de estar conectados por un medio físico, como es el cable. Además, crean una capa de abstracción, lo que permite que dispositivos móviles con diferentes sistemas operativos puedan intercambiar información.

Ved también

Este módulo no pretende extenderse en el funcionamiento y la seguridad de las comunicaciones inalámbricas, sino únicamente mostrar de una manera global los mecanismos de protección utilizados y las posibles vulnerabilidades que podrán ser explotadas por un atacante. Para los que quieran profundizar en la materia, se recomienda la lectura de los módulos didácticos de la asignatura *Seguridad en comunicaciones inalámbricas*, así como los documentos de la bibliografía.



El uso de las comunicaciones inalámbricas presenta una serie de ventajas, como la escalabilidad y la movilidad. Ahora bien, cuando nos centramos en temas de seguridad, las comunicaciones inalámbricas muestran su vertiente más oscura.

El uso del espectro electromagnético como medio de comunicación implica que la información viaja por el aire sin que nada ni nadie le pueda poner límites. Esto provoca que la información sea más difícil de proteger y, por lo tanto, que las propiedades de seguridad de las que se dispone en otros entornos no siempre se puedan alcanzar en las comunicaciones inalámbricas.

2.1. Ataques

Para poder obtener cotas de seguridad elevadas en las redes inalámbricas, es preciso analizar cuáles son las amenazas más importantes de este entorno y qué relación tienen con las propiedades que se han definido en el subapartado 1.1.

Aunque no hay una clasificación estricta de los posibles ataques a la seguridad (y por eso es importante la definición de las propiedades del subapartado 1.1), a continuación apuntamos los más importantes:

1) ***Masquerading***. Acción en la que el atacante suplanta la identidad de alguna entidad del sistema (estación base o dispositivo móvil) para obtener acceso a recursos de este sistema. Este ataque incide directamente en la propiedad de autenticación. Para prevenirlo, es necesario un buen proceso de autenticación, tanto por parte del dispositivo móvil como de los puntos de acceso a la red.

Ejemplo de *masquerading*

Un atacante puede suplantar una estación base de la red (emitiendo una señal de más potencia que la de la estación base legítima) y así capturar mensajes de autenticación de los usuarios. Una vez obtenida la información de estos mensajes, se puede hacer pasar por uno de los usuarios legítimos de la red para obtener acceso a los recursos.

2) Denegación de servicio[2]. Acción en la que el atacante consigue que el servicio no esté disponible para los usuarios legítimos o que el servicio se retrase o se interrumpa. Este tipo de ataque es quizá el único que no se puede identificar con ninguna de las propiedades de seguridad definidas en el apartado anterior. Este hecho se debe a que estos ataques se suelen llevar a cabo incluso con anterioridad al proceso de autenticación, justamente con el envío masivo de solicitudes de este tipo.

(2)En inglés, *deny of service* (DoS).

Ejemplo de ataque de denegación de servicio

Un ataque de denegación de servicio en una red inalámbrica se puede llevar a cabo mediante la generación de una señal de radio de la misma frecuencia que la de la red inalámbrica, pero con una potencia superior. De esta manera se atenúa la señal de la red, con lo cual no se permite a los usuarios utilizarla. Este tipo de ataques también se conoce como *jamming*.

3) ***Eavesdropping***[3]. Acción en la que el atacante obtiene información de una comunicación de la que no es ni emisor ni receptor. En este caso, el atacante vulnera la confidencialidad de la información que ha interceptado. Este tipo de ataque se clasifica como *ataque pasivo*, ya que el atacante obtiene información de los datos en tráfico, pero no puede realizar ninguna acción sobre la red ni sobre la información que circula. Es importante tener en cuenta, sin embargo, que la información obtenida en un ataque de *eavesdropping* puede dar lugar a un posterior ataque de *masquerading*.

(3)La palabra *eavesdropping* no tiene equivalente en castellano y se traduce como *escuchar secretamente*.

4) Confidencialidad de posicionamiento. Acción en la que el atacante obtiene, mediante diferentes técnicas, la posición física de un dispositivo móvil y, por lo tanto, la de su propietario. Este tipo de ataque puede afectar seriamente a la privacidad de las personas, en tanto que pueden ser localizadas en cualquier momento por el mero hecho de tener un dispositivo móvil en funcionamiento.

Ejemplo de ataque de confidencialidad de posicionamiento

La posición del dispositivo móvil se puede utilizar de manera positiva en aplicaciones para controlar flotas de transportes, pero también se puede utilizar de manera maligna, por ejemplo, para el envío de propaganda no deseada (*spamming*) relacionada con establecimientos próximos a la localización del dispositivo móvil.

Especificidades del entorno inalámbrico

La confidencialidad de posicionamiento es un rasgo característico de las comunicaciones inalámbricas; en este entorno es donde tiene sentido hablar de posición física (en los entornos con hilos, los dispositivos tienen una localización física concreta o, en cualquier caso, una movilidad muy limitada).

2.2. Mecanismos de prevención

La preocupación por la seguridad en los entornos inalámbricos es creciente, ya que el uso de este entorno para aplicaciones de comercio electrónico requiere un grado de seguridad elevado.

Cuando hablamos de *técnicas para prevenir riesgos de la seguridad*, podemos hacer una distinción clara entre las que trabajan en el nivel físico de la comunicación y las que trabajan en el resto de los niveles, tanto si se trata del nivel de enlace como del nivel de aplicación.

Las técnicas más habituales que se aplican al nivel físico son las de *difusión de espectro*[4]. Estas basan su funcionamiento en fraccionar la señal de radio y transmitirla de manera imperceptible por diferentes frecuencias. De esta manera, si no se conoce el modo como la señal ha sido distribuida por las diferentes frecuencias, no se puede reconstruir, ya que las diferentes señales que se reciben en cada frecuencia son percibidas como ruido.

(4)En inglés, *spread spectrum*.

Las técnicas de difusión de espectro también permiten atenuar los ataques de *jamming*, ya que la señal emitida en una frecuencia concreta para producir el ataque solo afectará a una parte de los datos enviados.

En cualquier caso, las técnicas de difusión de espectro ofrecen poca o nula seguridad y debemos buscar la justificación de su uso más en cuestiones de eficiencia que de seguridad. Sí es cierto que las técnicas de difusión de espectro permiten la reutilización de un espectro de radio para diferentes tecnologías de comunicación, ya que se minimizan las interferencias.

Desde el punto de vista de las capas superiores al nivel físico, el uso de la criptografía permite conseguir unos buenos niveles de seguridad. Por medio de la criptografía se pueden obtener servicios de autenticación y confidencialidad que permiten alcanzar las propiedades de seguridad descritas anteriormente y, por lo tanto, ayudan a reducir el éxito de los ataques descritos.

Criptografía

Este módulo no pretende introducir los conceptos básicos de la criptografía. A los que quieran profundizar en la materia, se les recomienda la lectura de los módulos didácticos de la asignatura *Criptografía* o de alguno de los libros sobre el tema que se incluyen en la bibliografía del módulo.

La mayoría de los sistemas de comunicación inalámbrica llevan a cabo el servicio de autenticación por medio del modelo **reto-respuesta**[5]. Este protocolo consiste en un intercambio de mensajes entre las dos partes que se quieren autenticar para asegurarse de que cada una de ellas conoce cierta información previamente intercambiada y que, por lo tanto, es quien dice ser.

[(5)]En inglés, *challenge-response*.

Ejemplo del modelo reto-respuesta

Supongamos que Anna y Bernat se conocen y han decidido que compartirán el número k = 7 para autenticarse. Cuando Anna (*A*) y Bernat (*B*) se encuentran, *A* se autentica ante *B* de la manera siguiente:

- *B* elige aleatoriamente como reto *c* un entero, por ejemplo el 4, y lo envía a *A*.
- *A* suma al reto *c = 4* el valor k = 7, que previamente habían acordado, y envía el resultado, r = 11, a *B*.
- *B*, que también ha calculado $r' = 4 + 7 = 11$, verifica que $r' = r$ y que, por lo tanto, *A* es quien dice ser, ya que conoce el valor *k* que previamente han acordado.

Obviamente, en los modelos de autenticación que utilizan este esquema, no es fácil obtener el valor *k* a partir de los valores intercambiados *r* y *c*.

La ventaja de este sistema es que el valor del reto *c* varía aleatoriamente para cada proceso de autenticación, de manera que la intercepción de los datos en un proceso de este tipo no compromete, en principio, los posteriores procesos.

Es importante destacar que el modelo reto-respuesta que hemos descrito y que utilizan la mayoría de los sistemas de comunicación inalámbrica para la autenticación necesita una información *k* que el emisor y el receptor conocen previamente al mismo proceso de autenticación.

En cuanto al servicio de confidencialidad, las tecnologías inalámbricas implementan esquemas basados en la **criptografía de clave compartida**. La criptografía de clave compartida, a diferencia de la criptografía de clave pública, se basa en el hecho de que tanto el emisor como el receptor comparten una misma clave. Este mecanismo encaja perfectamente en el modelo de autenticación de reto-respuesta que acabamos de describir, en el que las dos partes también tienen que compartir cierta información.

Lectura recomendada

La criptografía de clave compartida que se utiliza para proteger la confidencialidad en la mayoría de las comunicaciones inalámbricas son las cifras de flujo. En el módulo didáctico "Cifras de flujo" de los materiales de la asignatura *Criptografía* podéis encontrar más información sobre los esquemas de cifrado en flujo.

El uso de la criptografía de clave compartida no representa un problema excesivamente grave para algunos modelos de comunicación inalámbrica. Por ejemplo, si pensamos en la telefonía móvil, el usuario y el operador intercambian las claves en el momento en el que el usuario adquiere el terminal móvil; de hecho, más concretamente, cuando obtiene la tarjeta SIM. Por otra parte, en una WLAN los usuarios tienen acceso a ella por el hecho de pertenecer a una entidad o grupo, de manera que el intercambio de claves también es fácil de llevar a cabo. Este hecho, sin embargo, dificulta el proceso de apertura

de las redes inalámbricas en el sentido de que si se pretende dar cobertura de WLAN en un aeropuerto ofreciendo servicios de confidencialidad, las cosas se pueden complicar.

2.3. Caso de estudio: IEEE 802.11

El estándar IEEE 802.11 define la arquitectura de una red de área local en un entorno inalámbrico, donde se especifican dos servicios de seguridad: uno para obtener la propiedad de autenticación y otro para la propiedad de confidencialidad e integridad.

Dado que en uno de los procesos de autenticación se utilizan los algoritmos que se describen en el proceso de confidencialidad, describiremos primero este último.

El servicio de confidencialidad del estándar IEEE 802.11 se basa en el algoritmo *wired equivalent privacy* (WEP).

El algoritmo WEP proporciona las propiedades de confidencialidad e integridad. La confidencialidad se consigue utilizando criptografía de clave simétrica, en particular el cifrador en flujo RC4. La integridad se obtiene mediante un *checksum* CRC32.

RC4

El RC 4 responde a las iniciales de *Ron cryptosystem number 4* (Ron Rivest fue su creador, en el año 1987). Rivest cedió el desarrollo del algoritmo a la empresa RSA Data Security. Por otra parte, el algoritmo fue secreto durante siete años, hasta que en el año 1994 apareció anónimamente en Internet y actualmente ya es público.

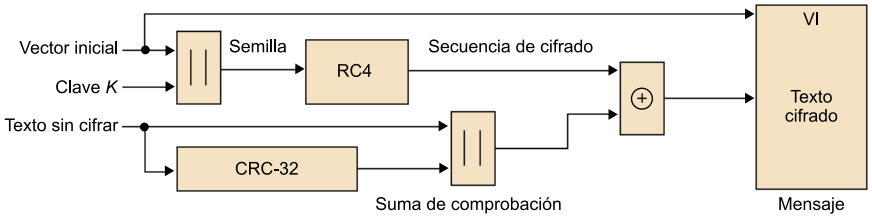
Tal como menciona el estándar, el algoritmo WEP pretende dotar las redes inalámbricas de las mismas propiedades de seguridad que las redes con hilos. Este argumento ha sido utilizado a menudo para rebatir los problemas de debilidad que tiene el algoritmo, ya que muchos de estos problemas también existen en las redes con hilos.

CRC-32

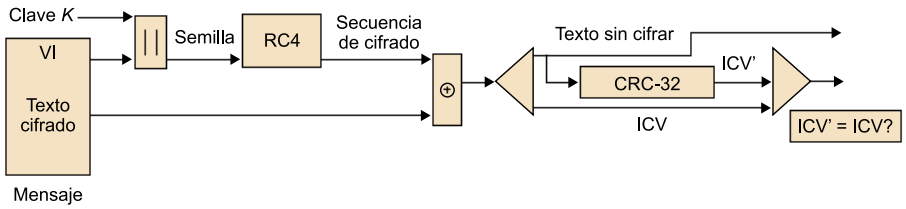
La función CRC-32 es una función lineal que tan solo utiliza sumas y multiplicaciones. Este hecho garantiza que sea fácil predecir el *checksum* resultante de una modificación en el texto en claro.

Esquema de cifrado del WEP

El WEP toma como entrada, por una parte, el texto en claro –es decir, la información que se ha de transmitir por la red inalámbrica– y, por otra parte, un vector inicial *VI* (de 32 bits) y una clave *K* (de entre 40 y 128 bits). Tal como muestra la figura, la clave *K* y el vector *VI* se concatenan y se obtiene la semilla del cifrador en flujo. El RC4 genera una secuencia pseudoaleatoria de bits que se suma al texto en claro para obtener el texto cifrado. Previamente, y para conseguir la propiedad de integridad, se aplica al texto en claro la función CRC-32 para obtener un *checksum* (*integrity check value* –ICV) de la información. Este valor se transmite para poder verificar posteriormente que la información no ha sido alterada.



El proceso de descifrado de la información que lleva a cabo el receptor es exactamente el inverso del que acabamos de describir, tal como se muestra en la figura siguiente.



Dado que el RC4 es un criptosistema de clave compartida, la estación y el punto de acceso necesitan intercambiar tanto el vector inicial *VI* como la clave *K* para poder comunicarse utilizando el WEP. Como el *VI se* envía en claro, la seguridad del algoritmo depende solo de la clave. Aun así, hay que asegurar la integridad de la transmisión del *VI*, ya que si el *VI* utilizado por el emisor no es exactamente igual que el del receptor, los procesos de cifrado y descifrado no serán inversos. Este intercambio de información se realiza durante el proceso de autenticación, que describiremos a continuación.

El estándar IEEE 802.11 tiene dos variantes que implementan el servicio de autenticación:

- *Open system authentication* (OSA).
- *Shared key authentication* (SKA).

El OSA es de implementación obligada en el estándar y lo incluyen por defecto la mayoría de los productos que se pueden encontrar en el mercado. Como su nombre indica, es un sistema de autenticación abierto y que, por lo tanto, no limita el acceso, lo que implica que, desde el punto de vista de la seguridad, este sistema por sí solo no tenga ningún tipo de interés.

El OSA simplemente intercambia mensajes entre una estación y el punto de acceso inalámbrico. Cualquier estación que pueda enviar y recibir mensajes correctos podrá tener acceso a la red.

El proceso de autenticación se establece a partir de dos pasos entre la estación y el punto de acceso:

- En el primer paso, la estación indica al punto de acceso su dirección MAC y un identificador que informa de que el mensaje es de autenticación.
- En el segundo paso, el punto de acceso responde con un mensaje, indicando si el proceso de autenticación ha tenido éxito o no.

Éxito del proceso de autenticación

El éxito del OSA solo depende de la capacidad que tenga la estación de generar correctamente tramas WLAN. Esto, de hecho, es poca cosa, porque obviamente el punto de acceso y la estación solo se podrán comunicar si utilizan el mismo protocolo con el mismo formato de tramas. Si son diferentes, no solo el proceso de autenticación OSA no tendrá éxito, sino que, posiblemente, la propia comunicación tampoco.

A partir de este momento, si se utiliza el método OSA la estación ya está autenticada.

Muchos de los sistemas de redes LAN inalámbricas que están en el mercado implementan un mecanismo adicional de control de acceso sobre el OSA basado en la dirección MAC de la estación. Este mecanismo consiste en no admitir la conexión de direcciones MAC no autorizadas. Así, cada punto de acceso tiene que gestionar la lista de las direcciones MAC autorizadas. La problemática del mantenimiento de las listas, la escalabilidad (imaginémonos un campus universitario donde cada estudiante tiene su portátil) y la suplantación de direcciones MAC provocan que este sistema no sea una solución idónea para el proceso de autenticación. Por este motivo, es recomendable utilizar el método de autenticación SKA.

El método SKA[6] permite la autenticación de las estaciones y los puntos de acceso por medio del algoritmo WEP, junto con un sistema de reto-respuesta.

(6)SKA son las siglas de *shared key authentication*.

Es importante destacar que, dado que el estándar IEEE 802.11 no especifica la obligatoriedad del algoritmo WEP y el SKA lo utiliza, las versiones del estándar que no tengan activado el WEP no podrán utilizar el SKA.

Este proceso de autenticación consiste en el intercambio de cuatro mensajes entre la estación que se autentica y el punto de acceso. El sistema de intercambio de claves no implica el envío de claves en claro, tal como veremos más adelante, pero requiere que la clave secreta compartida se haya proporcionado por un canal seguro con anterioridad al proceso de autenticación.

El punto de acceso envía continuamente una señal de baliza con el fin de anunciar su presencia. Una estación que quiera acceder a la red, al encontrar la señal de baliza, inicia el proceso de autenticación con el punto de acceso cuya dirección figura en la señal de baliza.

Proceso de intercambio de mensajes

El proceso de intercambio de mensajes es el siguiente:

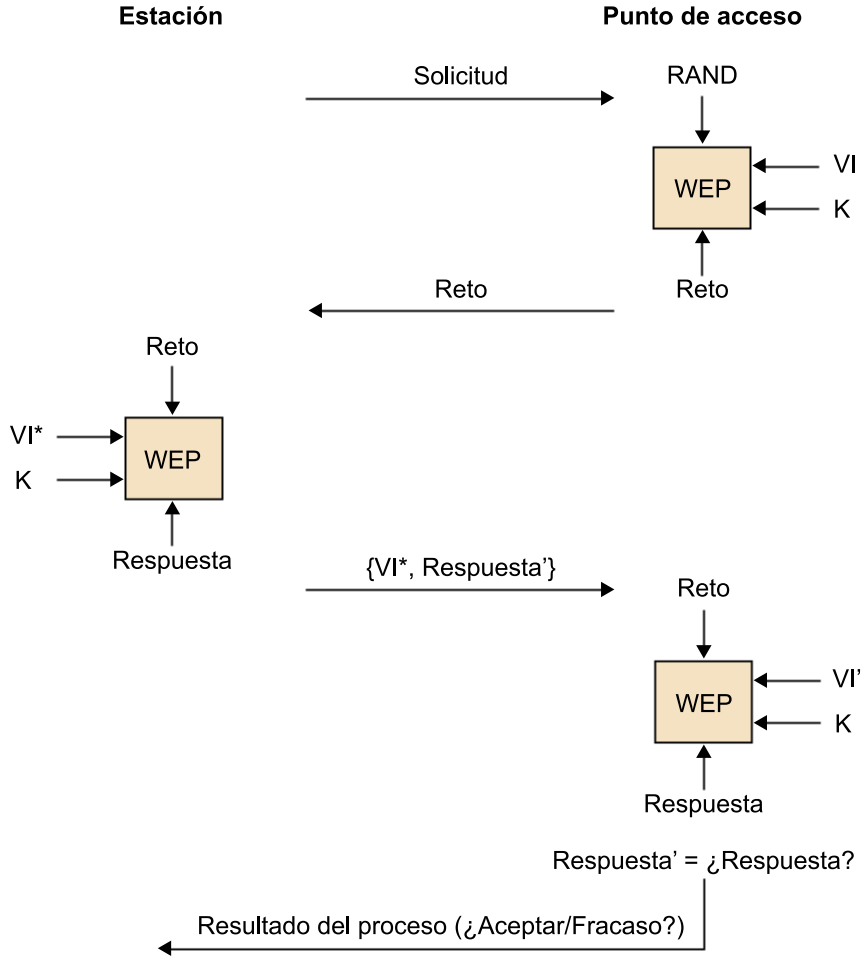
1) La estación envía un mensaje al punto de acceso para solicitar la autenticación.

2) El punto de acceso genera un **Reto** de 128 bytes utilizando el algoritmo WEP a partir de un valor pseudoaleatorio (RAND), una clave *K* que comparte con la estación y un vector inicial (*VI*), que la envía a la estación.

3) La estación genera la **Respuesta**' utilizando también el algoritmo WEP con el valor **Reto**, la clave *K* y un vector inicial (*VI**) diferente del que se ha utilizado en el paso anterior. La estación envía la **Respuesta**' al punto de acceso, junto con el vector inicial *VI** utilizado en su generación.

4) El punto de acceso calcula **Respuesta** utilizando el algoritmo WEP con la clave compartida, el valor **Reto** que ha enviado a la estación y el valor VI***** que ha recibido de la estación. Si los dos valores (**Respuesta** y **Respuesta**') coinciden, significará que la estación ha sido correctamente autenticada y el punto de acceso enviará un mensaje para indicarlo. En caso contrario, el resultado de la autenticación será fallido.

El proceso se repite para autenticar el punto de acceso.

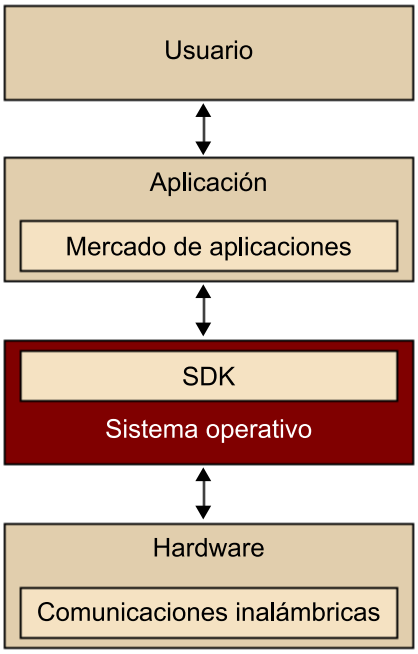


3. Sistema operativo

El sistema operativo es la capa que se encuentra entre el hardware y las aplicaciones de software. Es quien se encarga de gestionar los recursos de hardware del dispositivo y ofrecer servicios comunes para facilitar la programación de aplicaciones.

Ved también

Este módulo no pretende extenderse en el funcionamiento del sistema operativo, sino únicamente identificar los mecanismos de prevención básicos que este presenta en dispositivos móviles. Para los que quieran profundizar en la materia, se recomienda la lectura de los módulos didácticos de la asignatura *Sistemas operativos* y *Seguridad en sistemas operativos*, o alguno de los libros sobre el tema que se incluyen en la bibliografía.



En los últimos años, la complejidad del sistema operativo de los dispositivos móviles ha aumentado considerablemente. Se ha pasado de tener sistemas muy simples, a unos comparables con los de un ordenador. Este crecimiento ha hecho que su seguridad se convierta en un requisito primordial. Sin embargo, la obtención de esta seguridad en los sistemas operativos para dispositivos móviles es compleja.

Al principio, cuando los dispositivos móviles no eran un centro multimedia y de entretenimiento, los sistemas operativos para móviles eran bastante simples. De hecho, en aquellos dispositivos que tenían como función principal llamar o enviar SMS, el sistema operativo tenía poca importancia. Eso era así porque aquellos primeros sistemas operativos tenían como función principal ser compactos, eficientes y fiables. En cambio, las funciones adicionales que pudieran ofrecer eran secundarias. Pero, al ser sistemas tan limitados en recursos físicos y en funcionalidades, no interesaban a los atacantes.

Series 40 de Nokia

Uno de los sistemas operativos para dispositivos móviles (no *smartphones*) más usados fue el Series 40 de Nokia. Este sistema operativo era simple, no ofrecía multitarea real pero era muy fiable. La navegación por sus menús era instantánea y su consumo de recursos, muy eficiente.

Multitarea

Multitarea, en inglés *multitasking*, es la capacidad de ejecutar simultáneamente varios procesos.

En cambio, actualmente, cuando hablamos de un sistema operativo para móviles, nos referimos a un sistema complejo, que contiene características que hasta hace unos años eran impensables, como ejecutar aplicaciones 3D, navegar en la Red o multitarea. Estas nuevas funcionalidades abren más vías de ataque y aumentan el interés de los atacantes.

Symbian OS

Symbian OS ejecutando la interfaz gráfica de Series 60 fue el sistema operativo más utilizado en dispositivos *smartphones* en el año 2010. Este sistema operativo más complejo ya era multitarea, ofrecía mejoras en muchas aplicaciones, como el navegador web, y mejores juegos, debido a que en vez de utilizar únicamente Java para crear aplicaciones, se podían utilizar otras aplicaciones propias de esta plataforma que tenían acceso mucho más directo y eficiente al hardware.

En la actualidad, la mayoría de los móviles de gama media-alta ejecutan un sistema operativo muy potente, hasta el punto de que estamos hablando de ordenadores de bolsillo. En consecuencia, se heredan vulnerabilidades y ataques que hasta ahora han sido exclusivos de los ordenadores.

3.1. Ataques

Las principales amenazas que existen en el ámbito del sistema operativo son causadas por errores en el aislamiento de los recursos, ya sea debido a sus diseños, a errores en el software o a una mala configuración de sus servicios.

Ejemplo de ataque

En ciertas circunstancias, un proceso podría modificar los parámetros ya verificados por otro proceso, pero que este todavía no ha utilizado. Las consecuencias de tal ataque son imprevisibles, ya que el proceso realizará operaciones utilizando parámetros para los que no ha sido diseñado.

Estas vulnerabilidades pueden ser aprovechadas para ejecutar ataques tanto desde los servicios que ofrece el propio sistema operativo, como desde la capa de aplicaciones.

Si el ataque se realiza sobre un servicio del sistema operativo, sus consecuencias dependerán de la vulnerabilidad expuesta. Una vulnerabilidad que solo se puede explotar localmente es mucho menos crítica que una que se pueda explotar remotamente.

Ved también

Trataremos la vulnerabilidad aprovechable desde la capa de aplicaciones en el apartado 4.

Por otra parte, también circulan versiones no oficiales de los sistemas operativos móviles, denominadas ROM. Pueden ser copias de las versiones oficiales de los sistemas operativos o ROM personalizadas. Además, las ROM personalizadas pueden contener código malicioso.

Ejemplo

Un desarrollador puede modificar una versión del sistema operativo Android incluyendo un *keylogger* para registrar las pulsaciones del teclado. Posteriormente, estos datos registrados se envían al correo electrónico del desarrollador. Si el desarrollador cuelga en la Red esta ROM personalizada de Android y otros usuarios la instalan en sus dispositivos, el desarrollador empezará a recibir datos confidenciales de estos usuarios.

Keylogger

Keylogger es una aplicación encargada de almacenar todas las pulsaciones de teclado.

3.2. Mecanismos de prevención

Como ya hemos visto, los recursos y la información que gestiona el sistema operativo pueden estar en riesgo. Por lo tanto, es importante ver de qué mecanismos de seguridad disponen los sistemas operativos para dispositivos móviles. Los mecanismos de seguridad más importantes son los privilegios de usuarios, el aislamiento de procesos y las actualizaciones.

3.2.1. Privilegios de usuarios

Una de las características que suelen poseer estos sistemas operativos es la gestión de usuarios y privilegios, teniendo como mínimo dos usuarios: el usuario normal y el superusuario. Esta distinción de usuarios, tan común en ordenadores, a priori puede parecer extraña a un dispositivo móvil, ya que aquí no existe el concepto de iniciar sesión, pero es muy importante para las cuestiones de seguridad.

Superusuario

El superusuario en muchos sistemas operativos es conocido como *root*.

Un sistema operativo que gestione varios usuarios y privilegios puede aportar robustez al sistema, dado que los daños que un ataque pueda causar van ligados a los permisos del usuario que esté efectuando este ataque. Un usuario con privilegios limitados tendrá un impacto bajo sobre el sistema, mientras que un superusuario podrá producir una pérdida total del sistema operativo.

Generalmente, todas las aplicaciones se ejecutan con los privilegios del usuario normal, limitando mucho los cambios o desperfectos que el usuario puede causar al sistema. Por una parte, esto es muy importante, ya que en caso de que haya una vulnerabilidad, los daños que se podrán producir estarán limitados por los privilegios que el usuario tenga. Pero también es una limitación para el usuario, ya que únicamente podrá realizar las acciones que el sistema operativo le permita llevar a cabo con los privilegios actuales.

Usuarios del sistema iOS

iOS distingue como mínimo entre dos usuarios: *root* y *device*. El usuario *root* es un superusuario, mientras que el usuario *device* tiene permisos limitados, aunque tiene acceso a todos los datos almacenados. En los orígenes de este sistema operativo, todas las aplicaciones se ejecutaban como *root*, dando acceso total a la aplicación sobre el dispositivo, pero a la vez poniéndolo en peligro. Un fallo de una aplicación podía tumbar el sistema; una aplicación maliciosa tenía total disponibilidad para producir un ataque. Afortunadamente, en las primeras actualizaciones de iOS se corrigió este funcionamiento y las aplicaciones ya se ejecutan con el usuario *device*.

Por lo tanto, por defecto, el usuario no tiene nunca permisos de *root*. Esto, sin embargo, limita mucho las operaciones que el usuario puede realizar sobre el sistema operativo. Por ejemplo, únicamente se pueden instalar aplicaciones desde su mercado de aplicaciones y se restringen mucho las customizaciones sobre iOS. Debido a esto, el término *jailbreak*[7] se ha hecho famoso. Este proceso permite obtener el usuario *root* del sistema. Con estos nuevos privilegios se eliminan las limitaciones antes mencionadas, pero estas nuevas características pueden implicar una reducción de la seguridad del sistema.

El hecho de tener más control sobre el sistema operativo no significa necesariamente que se reduzca la seguridad si se sabe exactamente qué se está haciendo. Pero, de todas maneras, es más fácil que se olvide una puerta abierta o que alguna parte contenga un agujero de seguridad. Además, es importante cambiar las contraseñas de los usuarios. Por defecto, tanto el usuario *root* como el *device* tienen una contraseña conocida.

(7)Realizar *jailbreak* es una práctica legal en Estados Unidos desde el 2010.

Cabe recordar que los sistemas operativos móviles utilizan una autenticación basada en usuario y contraseña. Por lo tanto, si es posible, se recomienda cambiar la contraseña con la que vienen por defecto.

Además, como usuarios de un sistema operativo móvil deberíamos activar únicamente los servicios que necesitamos en un momento dado y sabiendo qué estamos haciendo. De esta manera, estaremos previniendo posibles ataques a nuestro dispositivo.

3.2.2. Aislamiento de procesos

Otra medida que se está implementando en sistemas operativos móviles es limitar los permisos que tiene cada aplicación, aislándolas. De esta manera, cada aplicación únicamente tendrá acceso a sus recursos y no podrá perturbar el funcionamiento de ninguna otra. En caso de que sea necesario acceder a algún recurso compartido, como puede ser una región de memoria, la aplicación debe tener validado el permiso para hacerlo. De esta manera podrá acceder a aquellos recursos a los que se le haya permitido el acceso.

Este aislamiento de procesos[8] suele estar implementado utilizando un lenguaje de programación que lo habilite, como Java, y creando para cada aplicación un nuevo usuario con privilegios muy restringidos, que le permiten únicamente acceder a los recursos a los que haya solicitado acceso. De esta manera, si una aplicación solicita únicamente acceso a la posición mediante GPS, no podrá conectarse a Internet.

(8)En inglés, *sandbox*.

Aislar la ejecución de cada aplicación garantiza que no pueden interferir en el funcionamiento de las otras, lo que hace el sistema operativo mucho más robusto. No obstante, cuando varias aplicaciones tienen que compartir algún servicio, hay que utilizar un sistema de permisos más complejo. La buena implementación de este aislamiento y la gestión de los recursos compartidos son fundamentales para que esta medida sea efectiva.

Dalvik VM de Android

Las aplicaciones en Android se ejecutan sobre una máquina virtual (Dalvik VM) de una manera parecida a como se hace en Java. Además, cada aplicación se ejecuta con un usuario y grupo de Linux diferente. Así, por defecto las aplicaciones no tienen permiso para realizar ninguna tarea que pueda interferir en las otras aplicaciones. Eso requiere el uso de unos permisos de seguridad más minuciosos y restringidos que los utilizados generalmente en Linux, permitiendo especificar qué operaciones puede ejecutar cada aplicación, y un permiso basado en una dirección *URI* para personalizar el acceso a una parte de los datos. Estos permisos son estáticos, los define el desarrollador en el fichero de configuración *AndroidManifest.xml* y el usuario debe aceptarlos cuando instala la aplicación. Algunos de estos permisos son hacer una llamada telefónica, modificar/borrar datos de la tarjeta de memoria, leer los números de teléfonos de tu agenda u obtener la información del GPS.

URI

URI es el acrónimo de *uniform resource identifier*. Consiste en una cadena corta de caracteres que identifica inequívocamente un recurso.

3.2.3. Actualizaciones

Adicionalmente, estos sistemas operativos disponen de actualizaciones periódicas. En caso de actualizaciones menores, estas suelen ser frecuentes para solucionar alguna carencia detectada, generalmente debida a errores en el software que presentan un riesgo para su seguridad. La manera de recibir estas actualizaciones difiere en cada sistema. Hay sistemas que las pueden recibir mediante la comunicación inalámbrica, a través del aire[9] (OTA), cuando el dispositivo se conecta con cable al ordenador o mediante la memoria externa que incorpora.

[(9)]En inglés, *over-the-air*.

iTunes

iOS utiliza el reproductor iTunes para sincronizar los datos del dispositivo. Además, este permite realizar actualizaciones de seguridad sobre el sistema operativo de una manera rápida y transparente para el usuario.

Los sistemas operativos también ofrecen actualizaciones mayores, las que además de solucionar errores en el software, también añaden nuevas funcionalidades y mejoran su rendimiento. Dado que estas actualizaciones realizan grandes cambios en el sistema, algunas llevan a cabo un borrado completo del dispositivo. Eso implica que toda la información personal será borrada. Por lo tanto, es muy importante que antes de realizar una actualización, se haga una copia de seguridad de todo el sistema, y en particular de los datos personales, para posteriormente restaurarlos en el nuevo sistema.

Finalmente, también es importante que las ROM de los sistemas operativos para dispositivos móviles se descarguen de sitios de confianza. Algunas de estas ROM son personalizadas y pueden contener código malicioso.

3.3. Caso de estudio: ssh en iOS

SSH

SSH son las siglas de *secure shell*, que es el nombre de un protocolo y el programa que lo implementa, y sirve para acceder a máquinas remotas a través de la Red.

Uno de los primeros casos donde muchos dispositivos móviles quedaron expuestos a acceso remoto no autorizado de atacantes se dio en iOS, causado por una mala configuración por parte de los usuarios del servicio de SSH.

En alguno de los métodos para obtener acceso de *root* a iOS, mediante *jailbreak*, se instala el servicio de SSH. La configuración por defecto de este servicio permite acceder remotamente al dispositivo móvil con el usuario *root*. Esto es así porque el usuario *root* viene por defecto con una contraseña conocida: *alpine*. De esta manera, cualquier dispositivo con iOS que tenga activado el servicio de SSH y utilice la contraseña del usuario *root* por defecto puede ser controlado remotamente cuando está conectado a una red inalámbrica abierta.

Una manera adicional de activar el servicio de SSH en iOS es mediante la instalación del paquete OpenSSH. La configuración por defecto del servicio es la misma, con lo que se mantiene el problema de seguridad mencionado.

Sin embargo, la solución de este problema de seguridad es bien sencilla: una correcta configuración en el mecanismo de autenticación del servicio de SSH. Esta configuración es tan fácil como cambiar la contraseña con la que viene por defecto el usuario *root*. Este proceso requiere una serie de pasos:

1) Descargar e instalar una aplicación que funcione como terminal de línea de comandos. Un ejemplo de esta aplicación es *MobileTerminal*.
2) Acceder a la aplicación de terminal de línea de comandos.
3) Obtener permisos de *root* introduciendo el comando *su root* y la contraseña *alpine*.

```
Leanders-iPhone-3GS:~ mobile$ su root
Password:
```

4) Cambiar la contraseña mediante el comando *passwd*. A continuación hay que introducir dos veces la nueva contraseña.

```
Leanders-iPhone-3GS:/var/mobile root# passwd
Changing password for root.
New password:
Retype new password:
```

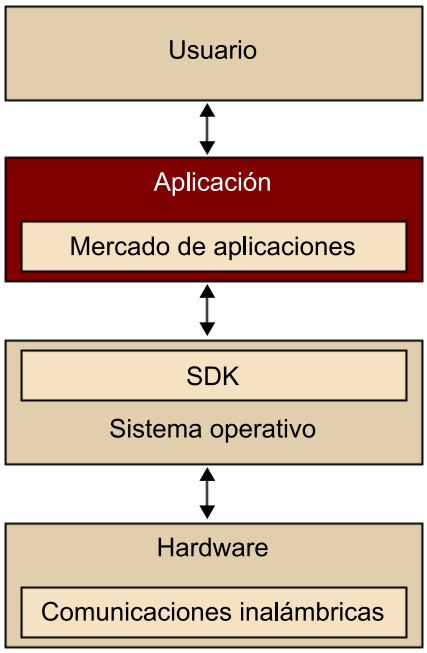
Una vez completados estos pasos, el servicio de SSH del dispositivo móvil queda protegido por la nueva contraseña introducida.

Es muy importante la correcta configuración de todos los servicios activados en el sistema operativo del dispositivo móvil, sobre todo de los que permiten accesos remotos al dispositivo.

4. Aplicaciones

Uno de los componentes más importantes en los dispositivos móviles son las aplicaciones que se pueden ejecutar sobre ellos, ya que con ellas interactuarán los usuarios.

En este nivel es muy importante revisar las medidas de seguridad que se han tomado para que las aplicaciones no puedan desestabilizar el sistema. Obviamente, estas medidas están complementadas por todas las que hay en las capas inferiores. Sin embargo, las vulnerabilidades que se detecten en este nivel son críticas, ya que podrán ser usadas directamente por las aplicaciones.



4.1. Ataques

Los ataques a la capa de aplicaciones los dividiremos en dos tipos: los ataques que se pueden ejecutar desde cualquier tipo de aplicación y los ataques que se pueden realizar únicamente desde el navegador web, ya que esta es una aplicación muy potente pero a la vez un punto crítico en la seguridad de estos sistemas.

4.1.1. Ataques al software: *malware*

Cuando hablamos del software también es importante revisar las diferentes amenazas que este puede sufrir. Nos centraremos en el *malware*[10], también denominado *código malicioso*, y revisaremos los diferentes objetivos que este puede tener.

(10)La palabra *malware* viene de la abreviatura de las palabras inglesas ***mal****icious soft****ware***.

Malware es una aplicación de software que tiene un objetivo malicioso en el dispositivo móvil donde se instala y se ejecuta sin el consentimiento del propietario. Puede tener objetivos muy variados, siendo los más comunes obtener datos personales y beneficio económico. Su modo de funcionamiento puede ser automático o controlado remotamente. Los principales tipos de *malware* son:

- **Virus**. Es un programa malicioso que infecta a otros archivos del sistema con la intención de modificarlos o hacerlos inservibles. Una vez un archivo ha sido infectado, también se convierte en portador del virus y, por lo tanto, en una nueva fuente de infección. Para que un virus se propague, este archivo debe ser ejecutado por el usuario. Generalmente tiene un objetivo oculto, como puede ser obtener contraseñas o realizar un ataque de denegación de servicio.

- **Gusano**. Es un programa malicioso autorreplicable que aprovechará las vulnerabilidades de la red para propagarse. Al igual que el virus, generalmente tiene un objetivo oculto.

- **Troyano**[11]. Es un pequeño programa oculto en otra aplicación. Su objetivo es pasar inadvertido por el usuario e instalarse en el sistema cuando el usuario ejecuta la aplicación. Una vez instalado, puede realizar diversas acciones, pero todas ellas sin el consentimiento del usuario. Además, estas acciones pueden realizarse instantáneamente o estar fijadas para realizarse en un futuro.

(11)El nombre *troyano* viene de las similitudes con el caballo de Troya.

- **Puerta falsa**[12]. Es un programa cuyo objetivo es abrir un acceso al ordenador para el desarrollador del *malware*, ignorando el proceso normal de autenticación. Esto implica que el dispositivo móvil infectado puede ser controlado remotamente por el atacante.

(12)En inglés, *backdoor*.

- ***Spyware***. Es una aplicación que recoge información sobre una persona u organización sin su consentimiento. Generalmente, el objetivo final de esta información recopilada es venderla a empresas de publicidad.

- ***Keylogger***. Es una aplicación encargada de almacenar todas las pulsaciones de teclado. Por lo tanto, puede capturar información confidencial, como el número de la tarjeta de crédito o las contraseñas.

- ***Hijacker***. Es un programa que realiza cambios en la configuración del navegador web. Un ataque típico es cambiar la página de inicio por una página de publicidad.

- ***Dialer***. Es un programa que de manera oculta realiza llamadas a teléfonos con tarifas especiales. De esta manera, el atacante puede obtener beneficios económicos.

Hablar de *malware* en ordenadores es normal hoy en día, pero no lo es tanto cuando nos referimos a dispositivos móviles. Sin embargo, en realidad es normal que cuanto más se parecen los dispositivos móviles a los ordenadores, más vulnerabilidades comparten.

Al principio, los pocos programas *malware* que existían para dispositivos móviles eran más una prueba de concepto que un código malicioso real. Esto era así porque los datos personales que se guardaban en el móvil eran muy pocos y era difícil poder obtener beneficio económico de alguna actividad maliciosa. Aun así, existieron aplicaciones *malware* que conseguían beneficio a base de enviar SMS a servicios publicitarios de los propios desarrolladores del *malware*.

Cabir

El primer *malware* destinado a dispositivos móviles complejos fue Cabir, detectado en junio del 2004. Fue desarrollado como una prueba de concepto para demostrar que era posible contagiar SymbianOS. La característica destacable de este *malware* era que podía propagarse mediante Bluetooth. Fue el inicio de una era.

Actualmente, el *malware* tiene una gran peligrosidad y muchas posibilidades de estafarnos, principalmente porque los usuarios somos mucho más vulnerables cuando utilizamos dispositivos móviles, ya que no tomamos las mismas medidas que tomamos cuando estamos delante de un ordenador.

Hoy en día, *malware* ha sido encontrado en las diferentes plataformas. La peligrosidad de este *malware* aumenta debido al desconocimiento por parte de los usuarios de los posibles peligros a los que están sometidos los dispositivos móviles.

DroidDream en los Android

En Android, uno de los primeros *malware* que se encontró fue DroidDream. Se detectó en muchas aplicaciones del Android Market. Su principal propósito era recopilar información sobre el dispositivo infectado, como el identificador del usuario, el tipo de dispositivo, el lenguaje o la región. Posteriormente, esta información era enviada a un servidor remoto. Pero sus objetivos maliciosos no acababan aquí. Mediante *exploits*[13], podía obtener permiso de *root* para algunas versiones del sistema operativo y a partir de allí romper el aislamiento de la aplicación y descargar código malicioso desde un servidor remoto. De hecho, este *malware* quedaba a la espera para recibir comandos de un servidor externo, pudiendo ejecutar cualquier acción sobre el sistema.

(13)*Exploit* es una pieza de software que automatiza el aprovechamiento de una vulnerabilidad.

Ikee.A en los iPhone

En iOS, en el año 2009 apareció Ikee.A, el primer gusano para los iPhone. Tuvo varias versiones. La primera versión fue únicamente una prueba de concepto que cambiaba el fondo de escritorio. Las versiones posteriores fueron mucho más peligrosas: permitían enviar la información confidencial del usuario a un servidor remoto o su control a distancia. Este gusano únicamente afectaba a los iPhones que habían obtenido permisos de *root* mediante *jailbreak* y que, después de instalar el servicio de SSH, no habían cambiado su contraseña por defecto. A continuación, se propagaba por la red buscando más víctimas.

Ved también

El problema de la seguridad del protocolo SSH se ha descrito en el subapartado 3.3.

Una práctica muy común a la hora de desarrollar software es la de reutilizar trozos de software de otros desarrolladores, como por ejemplo utilizar librerías externas. Esta práctica provoca que los desarrolladores no siempre conocen el 100% del código fuente de su programa. En consecuencia, un programa puede ser comprometido, ya que utiliza una librería externa que contiene código malicioso.

Uno de los pocos factores a favor de los usuarios es la diversidad de tecnologías móviles que se utilizan. Las diferentes tecnologías requieren que los desarrolladores de *malware* hayan de escribir un código para cada plataforma, lo que puede frenar la velocidad de su propagación.

4.1.2. Ataques en la web

A continuación veremos una pequeña descripción de las principales vulnerabilidades que pueden afectar a los navegadores web y de los mecanismos de seguridad existentes. De esta manera, podremos hacernos una idea de cómo de vulnerables podemos ser ante estos ataques y de que debemos tener mucho cuidado cuando navegamos por la web.

Ved también

Este módulo no pretende extenderse en el funcionamiento de la web ni de los distintos ataques que se pueden realizar en ella. Para los que quieran profundizar en la materia se recomienda la lectura de los módulos didácticos de la asignatura *Seguridad en aplicaciones web*, o de alguno de los libros sobre el tema que se incluyen en la bibliografía.

Aunque hay un largo listado de ataques que afectan a las páginas web, únicamente nos centraremos en los dos que más relevancia tienen cuando hablamos de dispositivos móviles: *web spoofing* o *phishing* y *clickjacking*.

Web spoofing o *phishing*

Web spoofing o *phishing* es un tipo de ataque que consiste en suplantar una página web y a partir de allí intentar obtener información confidencial de manera fraudulenta. Esta información suele consistir en contraseñas o información bancaria, por ejemplo las tarjetas de crédito.

El estafador o atacante puede suplantar una página web de muchas maneras, aunque la más común es utilizando una dirección web muy parecida a la original. La página web fraudulenta tendrá una estructura idéntica a la original para que el usuario no pueda detectar a primera vista que está siendo víctima de este tipo de ataque.

Además, las páginas web fraudulentas pedirán información confidencial del usuario que teóricamente no tendrían que pedir, como por ejemplo el número de su tarjeta de crédito.

Por lo tanto, para el usuario es difícil detectar este tipo de ataque, ya que puede llegar a la página web suplantada por medio de un enlace y la única diferencia que podrá detectar a simple vista está en la dirección web. Además, el problema se agrava a causa de que los dispositivos móviles disponen de una pantalla limitada, lo que hace que pueda ser difícil ver la dirección web completa.

No obstante, como usuarios hemos de saber que nunca deberíamos introducir información confidencial que una página web no nos tendría que pedir. Por defecto, debemos desconfiar siempre. En este tipo de ataque estar totalmente pendiente de lo que se hace es clave. Cualquier descuido te puede llevar a ser víctima de una estafa.

Ejemplo de *phishing*

Al buzón de correo electrónico nos llega un mensaje con un enlace a nuestra página bancaria y que nos dice que debemos actualizar nuestros datos de la tarjeta de crédito, ya que se ha aplicado una nueva política de seguridad para evitar estafas. En la parte inferior se encuentra el logotipo del banco, que en lugar de apuntar a http://www.bancdeprova.es apunta a http://www.bancdaprova.es. Si decidimos hacer un clic en el enlace, ya estamos un paso más cerca de ser estafados.

Una vez dentro de la web, vemos que la página es extremadamente similar a la web original, hasta el punto de que es imperceptible la diferencia. Se nos solicita que rellenemos un formulario donde se nos pide tanto el número de seguridad de la tarjeta (los tres números de la parte posterior) como una actualización de nuestros datos, incluida la contraseña. Supongamos que no nos damos cuenta de que hemos entrado en una página fraudulenta, y no desconfiamos de los datos que se nos pide. Por lo tanto, rellenamos el formulario. En este punto ya no hay marcha atrás. El atacante ha obtenido nuestros datos bancarios e intentará sacarnos dinero con el fin de rentabilizar sus ataques.

Cuando se opera con datos bancarios la atención es clave. Los atacantes intentarán aprovechar que estamos ocupados, con prisa o distraídos. También se aprovecharán de que somos demasiado crédulos. Antes de seguir algún enlace con el navegador móvil, nos debemos asegurar de que enlaza al sitio correcto. Si no estamos seguros, no deberíamos seguir. Siempre que sea posible, se tiene que acceder a la web escribiendo manualmente la dirección de la web y, sobre todo, vigilar los enlaces que nos lleguen por correo electrónico. La pequeña pantalla del dispositivo, sumada a la falsa sensación de seguridad en dispositivos móviles, nos puede jugar una mala pasada.

Clickjacking

Clickjacking es una técnica que engaña al usuario para que haga un clic sobre elementos de un sitio web que no haría voluntariamente. Esto se consigue superponiendo dos páginas. La principal es la que contiene un elemento que al atacante le interesa que pulsemos, como puede ser la confirmación de la habilitación de un permiso. La otra, la que nos engaña, está superpuesta a la

primera. Obviamente, la página superpuesta debe tener elementos que nos incentiven a pulsar los botones que al atacante le interesan, como por ejemplo, algún tipo de juego.

Sin entrar demasiado en detalle, esta técnica se basa en la superposición de *iframes*. Estos son elementos HTML que permiten la inclusión de un recurso externo dentro de nuestra página. Y aunque tienen una serie de limitaciones a la hora de su acceso mediante JavaScript, es posible utilizarlos para engañar a los usuarios.

4.2. Mecanismos de prevención

En la capa de aplicación trataremos varios mecanismos de prevención. Empezaremos por ver cómo se ha creado una primera capa de seguridad utilizando los mercados de aplicaciones. A continuación, veremos mecanismos de seguridad que se pueden utilizar en la web. Finalmente, describiremos distintos tipos de aplicaciones que pueden aumentar el nivel de seguridad en los dispositivos móviles.

4.2.1. Mercado de aplicaciones

En los dispositivos móviles actuales, casi toda la seguridad se ha centrado en la creación de un punto centralizado y fiable para descargar y gestionar las aplicaciones. Se trata de lo que denominamos mercado de aplicaciones. Cada mercado de aplicaciones puede tener políticas más o menos restrictivas, ya sea respecto al tipo de contenido, o a su potencial peligrosidad. No obstante, a grandes rasgos, los mercados se pueden dividir en dos grupos:

- Mercados que permiten todas las aplicaciones, indiferentemente de quién las haya publicado y de su funcionalidad.
- Mercados que tienen un control para limitar las aplicaciones accesibles. Esto está ligado a la existencia de un proceso previo de revisión.

El primer grupo crea un lugar central de distribución de aplicaciones, pero se desentiende de las consecuencias que una aplicación descargada pueda tener. Esta centralización facilita al usuario el acceso a las aplicaciones, pero a priori no añade ninguna medida de seguridad. De todos modos, se pueden ofrecer sistemas de valoración de las aplicaciones, ya sean numéricas o a modo de comentarios. Estos sistemas aportan un grado de seguridad en el sentido de que cuando se detecte una aplicación maliciosa, la propia comunidad la desprestigiará y la aplicación perderá relevancia. No obstante, puede haber aplicaciones maliciosas que no se hayan descubierto y que, por lo tanto, sigan teniendo una valoración positiva.

Android Market

Android pertenece al grupo de mercados que permiten todas las aplicaciones. Tiene el mercado de aplicaciones Android Market, que permite a cualquier desarrollador alojar su aplicación para que los usuarios la descarguen. Por lo tanto, nadie revisa las aplicaciones antes de que sean publicadas. Sin embargo, todas las aplicaciones han de ser firmadas con un certificado, cuya clave pública pertenece al desarrollador. Con este mecanismo, se autentica de una manera segura al desarrollador. Además, el sistema dispone tanto de valoración numérica como de comentarios para cada aplicación. Finalmente, existe la posibilidad de que Google retire una aplicación del mercado si en algún momento llega a su conocimiento que es maliciosa; e incluso, puede desinstalar aplicaciones remotamente. Adicionalmente, aplicaciones que no estén en el Android Market se pueden instalar de una manera sencilla.

Como ya se ha comentado antes, Android añade una capa de seguridad que limita los accesos que una aplicación puede hacer al sistema operativo por medio de permisos. Por lo tanto, cada aplicación antes de instalarse pedirá los accesos a los recursos que quiere tener. Si no los pide, no los podrá utilizar. Esta política depende de la intervención del usuario, pero es muy potente a la hora de limitar los daños que cada aplicación puede causar. Por ejemplo, una aplicación para controlar la velocidad con la que te mueves no necesita ver tus mensajes.

En cambio, el segundo grupo de mercados crea un lugar central de distribución en el que ejerce un control sobre la calidad y el contenido. Este control elimina aplicaciones que puedan causar un mal funcionamiento del sistema o que directamente sean maliciosas. El problema es que cada aplicación tiene que ser revisada por completo antes de ser publicada, lo que ralentiza su publicación. Además, no siempre es fácil revisar una aplicación por completo y ver si lo que está haciendo es malicioso o no. Desafortunadamente, esta revisión puede implicar una especie de censura, por lo que únicamente las aplicaciones que la compañía considere aceptables se publicarán.

AppStore

iOS de Apple incorpora AppStore, un mercado de aplicaciones basado en la segunda opción, que ejerce un control total sobre las aplicaciones que pueden ser descargadas al dispositivo. Por ello, cada aplicación se revisa con detalle. Pero revisar por completo una aplicación no siempre resulta fácil y, de hecho, se han producido casos en los que aplicaciones que declaraban hacer una cosa, por debajo de esta hacían otra adicional. Por ejemplo, una aplicación que decía ser una linterna, por debajo habilitó el *tethering*, es decir, compartir la conexión móvil del dispositivo a través de conexión inalámbrica, acción no permitida por la compañía. Por lo tanto, cuando la aplicación se hizo popular, fue eliminada del AppStore.

Además, por defecto iOS no permite instalar aplicaciones de otras fuentes. Para hacerlo, hay que habilitar el usuario *root* del sistema por medio del método antes mencionado, *jailbreak*.

Estos mercados de aplicaciones también pueden permitir una actualización de las aplicaciones. Esta debe ser una tarea realizada periódicamente, ya que las nuevas aplicaciones, aparte de mejoras en el rendimiento y funcionalidad, suelen solucionar vulnerabilidades detectadas que podrían ser explotadas.

Actualización de aplicaciones Android

Android permite realizar una actualización automática de sus aplicaciones mediante Android Market. Pero no todas las aplicaciones se pueden actualizar automáticamente; por ejemplo, aplicaciones cuyos permisos han cambiado respecto a la aplicación anterior requerirán la interacción del usuario para que este acepte los nuevos permisos.

Además, las compañías también se reservan el derecho de eliminar a distancia y automáticamente cualquier aplicación considerada como peligrosa. Esta actuación asegura una rápida eliminación de la aplicación una vez se ha declarado nociva.

Un caso especial son las aplicaciones que internamente cobran por determinados servicios, como puede ser desbloquear un nuevo nivel en un juego. En este caso, la seguridad vendrá marcada por cómo cada aplicación gestione los datos bancarios y por los mecanismos de prevención que internamente implemente.

4.2.2. Navegador web

Por defecto, todos los sistemas operativos se venden con una aplicación encargada de la navegación web. Aunque este pueda no parecernos un punto crítico en la seguridad del sistema, la experiencia en los ordenadores corrobora que sí lo es.

Por lo tanto, es muy importante que tratemos esta aplicación con tanto respeto como la tratamos en el ordenador, ya que son aplicaciones que pueden ejecutar código muy complejo. Los navegadores ejecutan código a nivel de usuario, con el nivel de privilegios que este tenga establecido. Los códigos que se ejecutan son potencialmente peligrosos, debido a la gran cantidad de funcionalidades que se puede implementar con ellos. Los lenguajes más utilizados son HTML/DHTML y JavaScript. Además, el contenido multimedia puede ser directamente abierto desde aquí, ejecutando otras aplicaciones que también pueden tener vulnerabilidades.

No obstante, no hay ninguna protección a nivel de usuario que sea infalible, ya que existen técnicas para realizar ataques de *man in the middle* (MiM) que no dependen de la intervención del usuario y, por lo tanto, no podrán evitar el éxito del ataque. Por lo tanto, hay que vigilar las páginas a las que se accede y las ejecuciones de código, como por ejemplo JavaScript.

Vulnerabilidad de las BlackBerry

En el navegador web de las BlackBerry se encontró un error de software en el motor de renderizado que permitía la ejecución remota de código. Esta vulnerabilidad se producía cuando el usuario accedía a una web que el atacante había diseñado malintencionadamente. Una vez se había accedido a la web, el atacante era capaz de leer y escribir en la sección de almacenamiento de la memoria interna o en la tarjeta externa de almacenamiento.

Debido a los ataques cuyos objetos son las aplicaciones web, se han tomado medidas para intentar reducirlos, como por ejemplo utilizar HTTPS.

HTTPS es un protocolo de aplicación basado en HTTP, pero destinado al acceso seguro a páginas web. Es utilizado básicamente por cualquier servicio que necesite el envío de datos personales, como tiendas virtuales o bancos.

Básicamente, HTTPS pretende crear un canal seguro sobre una red insegura, garantizando protección contra ataques de *eavesdropping* y *man in the middle*. Su seguridad se basa en utilizar cifrado, certificados digitales y una autoridad de certificación, que actúan como *terceras partes de confianza*[14], en las que tanto la web como el usuario tienen que confiar.

(14)En inglés *trusted third party* (TTP).

Para que una conexión HTTPS sea segura, debe cumplirse que:

- El usuario confíe en la autoridad de certificación.
- El sitio web proporcione un certificado válido, firmado por la misma autoridad de certificación en la que confía el usuario, y que este certificado identifique correctamente el sitio web.
- El navegador web del usuario alerte cuando detecta que hay un certificado inválido.
- El protocolo utilizado para el cifrado sea de confianza.

4.2.3. Aplicaciones de seguridad

Aunque hasta ahora hemos visto las aplicaciones como una fuente de vulnerabilidad, estas también pueden contribuir a aumentar el nivel de seguridad del sistema. Existen diferentes aplicaciones que pueden añadir nuevas capas de seguridad a los dispositivos móviles, ya sea con métodos de autenticación adicionales más restrictivos, sistemas de copia de seguridad[15], cifrado de los datos, aplicaciones antivirus o cortafuegos[16].

(15)En inglés, *backup*.

(16)En inglés, *firewall*.

Autenticación adicional

Por defecto, los *smartphone* se venden como mínimo con una medida de autenticación como es el código PIN, que se tiene que introducir al iniciar el dispositivo. Pero, una vez el dispositivo está encendido, no es común utilizar más métodos de autenticación, lo que lo hace muy vulnerable. Por ello, se han desarrollado aplicaciones que aseguran la autenticación durante la utilización del dispositivo.

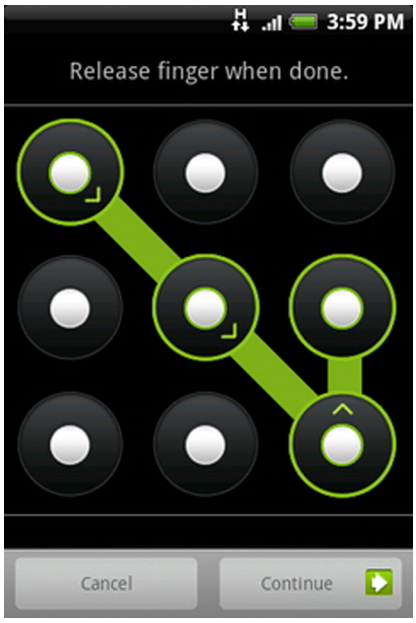
Aplicaciones de autenticación en Android

La aplicación Carrot App Protector de Android permite utilizar una contraseña adicional para ejecutar ciertas aplicaciones, tales como clientes de correo, galería de imágenes o mensajes de texto. No obstante, para que esta aplicación sea efectiva al 100% es necesario bloquear otras aplicaciones que permitan saltarse esta limitación, por ejemplo los gestores de tareas, las consolas o las aplicaciones que permiten instalar otras aplicaciones.

Algunas versiones de Android incorporan lo que se denomina *patrón de desbloqueo*, que consiste en dibujar sobre la pantalla del dispositivo móvil un patrón que previamente ha sido definido para desbloquear el móvil. Es una medida poco intrusiva (tardamos poco

tiempo en introducir el patrón), pero a la vez potente, ya que desbloquear el dispositivo sin conocer el patrón puede costar mucho tiempo.

Ejemplo de patrón de bloqueo de pantalla



Fuente: HTC

Copia de seguridad

Una de las acciones imprescindibles cuando trabajamos con información es la creación de copias de seguridad. En los dispositivos móviles almacenamos mucha información, el problema es que a veces no es fácil realizar una copia de seguridad global, ya que cada aplicación puede almacenar los datos internamente, en una parte concreta de memoria. Por esto, es importante disponer de una aplicación que facilite y automatice este proceso.

Titanium Backup

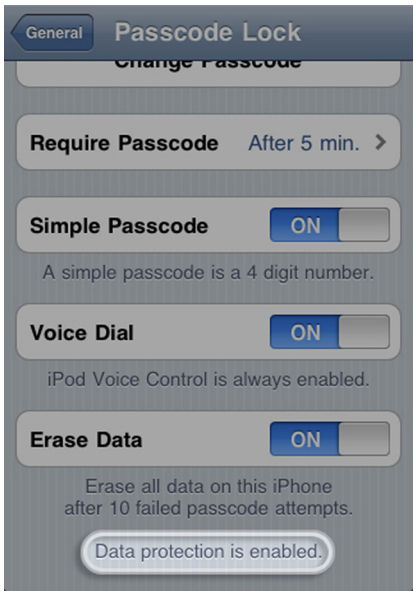
La aplicación Titanium Backup de Android permite realizar copias de seguridad tanto de las aplicaciones instaladas como de los datos que contienen. Eso sí, esta aplicación requiere permisos de *root*.

Cifrado

Los dispositivos móviles pueden almacenar mucha información delicada, como documentos confidenciales o datos bancarios. Por ello, resulta útil añadir una capa más de seguridad y cifrar estos datos. Así se garantiza que, en caso de pérdida, la información sea ilegible para alguien no autorizado.

Cifrado en iOS 4

Desde iOS 4 el propio sistema operativo incorpora una opción para habilitar el cifrado de datos. En esta versión, únicamente la aplicación de correo electrónico la tiene implementada, aunque esta funcionalidad está disponible desde la API[17] para que aplicaciones de terceros puedan utilizarla.



Fuente: Apple

(17)Son las siglas en inglés de *application programming interface*, que se traduce como interfaz de programación de aplicaciones.

Antivirus

El *malware* ya es una realidad en los dispositivos móviles. Por lo tanto, empiezan a ser necesarias aplicaciones que analicen los ficheros para evitar infecciones. Estas soluciones pueden heredar toda la experiencia adquirida en los ordenadores y, por lo tanto, los antivirus más populares en los ordenadores están creando sus versiones para dispositivos móviles.

AVG

AVG ha sacado una versión de su antivirus para Android denominada AVG Anti-Virus Free. Esta permite escanear el dispositivo buscando virus, revisar una aplicación en busca de *malware* antes de descargarla y revisar el contenido de una página web, correo electrónico o SMS antes de descargarlo al dispositivo.

Cortafuegos

Ya que los dispositivos móviles cada vez realizan y reciben más conexiones con dispositivos externos, soluciones comunes a los ordenadores, como los cortafuegos, también se empiezan a popularizar en este entorno. Se trata de los programas que permiten controlar las comunicaciones.

FirewayIP

FirewayIP es una aplicación para iOS que realiza la función de cortafuegos. Se puede utilizar para llevar a cabo cualquier acción de la que suponemos que requiere cortafuegos, como fijar reglas de conexión para una aplicación, de modo que cuando una aplicación intente acceder a la Red, se nos notifique y podamos permitirlo o no.

Fuente: iHackintosh

4.3. Caso de estudio: ZEUS *man in the mobile*

Zeus es un troyano informático para ordenadores que ejecutan Windows, que tiene como objetivo robar información bancaria mediante un *keylogger*. Fue detectado por primera vez en el 2007, pero su popularidad aumentó en el 2009 cuando infectó ordenadores de compañías importantes. Zeus ha sido usado para crear grandes *botnets*. Básicamente, Zeus captura contraseñas y datos bancarios de los ordenadores que ha infectado de una manera oculta.

Botnet

Botnet es un grupo de dispositivos conectados, que, una vez infectados, son controlados remotamente para realizar tareas sin la autorización del propietario.

Pero Zeus ha avanzado un paso más, convergiendo el *malware* de los ordenadores y *smartphones* en un único esquema, conocido como MITMO (*man in the mobile*).

Actualmente, muchos bancos, aparte de la autenticación tradicional de usuario y contraseña, ofrecen una segunda autenticación que se basa en recibir un SMS con un código, que posteriormente hay que introducir para realizar una transacción, también conocido como TAN[18]. Y es aquí donde Zeus ha visto una oportunidad para efectuar un nuevo ataque.

(18)TAN son las siglas de *transaction authentication number*.

Básicamente, Zeus utiliza tres técnicas para efectuar el robo de información:

- **Redirección**: el usuario es redirigido a un sitio web diferente del original, aunque visualmente pueda ser una réplica exacta. Toda la información introducida es almacenada por el atacante.

- **Captura**: mediante *keyloggers* o capturas de pantalla.

- **Inyección**: inyectando código HTML en el navegador web del usuario infectado que pide datos que normalmente la entidad bancaria no pediría, como el código de seguridad de la tarjeta de crédito.

La técnica más potente y popular es la de inyección. Una de las últimas versiones de este troyano utiliza la inyección de código para, una vez el usuario ha sido autenticado en la web de su servicio bancario, pedirle su número de

teléfono y el modelo del dispositivo móvil. Entonces, el usuario recibe en su dispositivo móvil un mensaje con un enlace de donde descargar una aplicación que dice ser complementaria al sistema de autenticación de su banco.

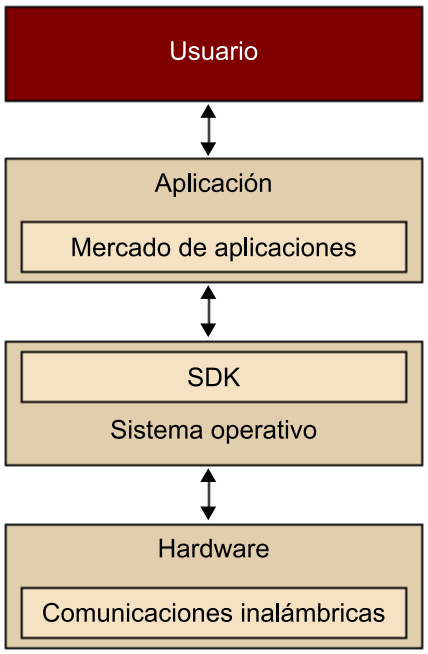
Una vez se instala la aplicación en el dispositivo móvil, el usuario ha sido infectado. Lo primero que hace la aplicación es enviar un SMS a un teléfono móvil preestablecido diciendo que la aplicación ha sido correctamente instalada. A continuación, monitoriza todos los mensajes; si vienen desde el número preestablecido, los analiza en busca de comandos que ejecutar y después los borra. Los comandos permiten, entre otras cosas, ignorar todas las peticiones mediante SMS, cambiar el número preestablecido desde donde se controla y borra contactos.

Debido a este funcionamiento, todo el sistema es transparente para el usuario, ya que en ningún momento ve los mensajes. Además, Zeus también implementa funcionalidades para el reenvío de mensajes SMS con la finalidad de enviar los TAN usados como segunda medida de autenticación por algunos bancos.

Además, como el dispositivo infectado puede recibir órdenes por SMS, produce infecciones mucho más resistentes. No hay un punto central que pueda ser bloqueado para frenar la infección. Sin embargo, debemos recordar que la infección se debe a que el usuario ha instalado el software malicioso a partir de un enlace recibido. De hecho, este ataque no es más que un ataque de *phishing* pero, como se ha comentado antes, la gravedad de este proceso se acentúa por la poca percepción que los usuarios tienen de los riesgos existentes y la peligrosidad que representan.

5. Usuario

En este apartado nos centraremos en las intrusiones físicas en el dispositivo móvil por parte de un usuario que ha tenido acceso físico al dispositivo móvil. Cuando hablamos de intrusión física, nos referimos a que alguien ha tenido acceso físico al dispositivo. Este tipo de intrusión es el más peligroso, ya que es muy vulnerable. Aun así, hay medidas que se pueden tomar con el fin de que los datos que almacenamos en nuestro dispositivo móvil permanezcan seguros.



5.1. Ataques

Los ataques que se pueden realizar dependen básicamente de si el dispositivo está en funcionamiento o no y del tiempo que el atacante tenga para comprometer el sistema. Con el fin de estructurar mejor la explicación del tema, seguiremos el segundo criterio, diferenciando cuándo el móvil ha sido vulnerable durante un tiempo reducido de cuándo ha sido expuesto al atacante durante mucho tiempo.

En el caso de ataques momentáneos, el atacante tendrá poco tiempo para realizar acciones. Si el dispositivo está apagado y protegido con el código PIN, lo podemos considerar como seguro. En cambio, si está encendido, se pueden emprender varias acciones:

- Leer la información fácilmente accesible, como los contactos, SMS, correo electrónico o fotografías.

- Eliminar información fácilmente accesible.

- Reenviar alguna información importante, como un correo electrónico que contenga la contraseña para algún servicio web.

- Conectar el dispositivo móvil a un ordenador y copiar parte del contenido de su memoria, donde puede haber, por ejemplo, documentos, imágenes, contraseñas o datos de aplicaciones.

- Instalar código malicioso.

En el caso de ataques de duración indefinida, el dispositivo móvil ha sido sustraído durante un tiempo indeterminado. Este puede ser por pérdida, robo o incluso por una intervención por parte de la policía. En este caso, el atacante dispondrá de todo el tiempo que quiera para intentar romper el sistema. Las acciones que puede emprender, aparte de las mencionadas anteriormente, son:

- Copia de toda la memoria.

- Técnicas forenses para recuperar información que ha sido borrada recientemente.

- Intentar encontrar las contraseñas mediante ataques de fuerza bruta[19].

[19] En criptografía se denomina *ataque de fuerza bruta* al proceso de recuperar una clave probando todas las combinaciones posibles hasta encontrar la que permite el acceso.

5.2. Mecanismos de prevención

Dividiremos los mecanismos de prevención con el mismo criterio seguido en los ataques: según si su protección será efectiva en caso de sustracciones momentáneas o indefinidas.

5.2.1. Sustracción momentánea

En este caso, como el atacante tendrá poco tiempo para realizar acciones, las medidas de seguridad de acceso al dispositivo que tengamos activadas serán muy efectivas.

Antes de nada, hay que establecer una autenticación cuando se encienda el teléfono. Esta puede ser más segura, utilizando usuario y contraseña, o menos, introduciendo el código PIN. De todas maneras, en un acceso momentáneo, cualquier código que no sea totalmente previsible, como por ejemplo el 0000, será suficiente.

Una vez encendido, es importante que el dispositivo se bloquee automáticamente y que pida una autenticación para desbloquearlo. En caso contrario, un acceso momentáneo al dispositivo encendido permitirá el robo de datos. La autenticación en este nivel puede ser más sencilla, ya que la utilizaremos más a menudo, pero debe seguir siendo imprevisible.

Un ejemplo es el patrón de desbloqueo que incorporan algunas versiones de Android, del que se ha hablado anteriormente.

Únicamente con estas dos medidas podemos garantizar con una gran probabilidad que los datos almacenados en nuestro dispositivo estarán protegidos ante un acceso momentáneo al dispositivo.

5.2.2. Sustracción indefinida

En el caso de sustracción indefinida, el atacante dispondrá de todo el tiempo que quiera para intentar romper el sistema.

Las medidas iniciales que se deben tomar son las descritas en el apartado anterior: autenticación tanto al iniciar como al desbloquear. Además, es recomendable que el bloqueo del dispositivo se produzca automáticamente después de un tiempo de inactividad. Y a partir de aquí, hay que intentar dificultar cualquier tipo de extracción de información.

Como se ha comentado, la medida de desbloqueo del dispositivo una vez encendido frecuentemente es menos segura que la inicial, por lo cual sería útil poder apagar el dispositivo remotamente. Sin embargo, antes de este apagado del sistema, nos interesaría realizar otras acciones, como eliminación del contenido almacenado.

El contenido puede ser eliminado tanto remota como localmente. Remotamente vendría producido por el envío de nuestro comando sobre el dispositivo. Localmente, podría realizarse por una aplicación que, en caso de introducción incorrecta del código de autenticación un cierto número de veces, automáticamente elimine todo el contenido del sistema.

WaveSecure

La aplicación WaveSecure de Android es una herramienta de seguridad para el dispositivo y los datos que este almacene. Permite realizar copias de seguridad, bloquear el dispositivo remotamente, bloquearlo al cambiar el SIM y monitorizar en un mapa la localización actual en la que se encuentra el dispositivo.

Sin embargo, en algunos casos es posible que no sea necesaria la eliminación del contenido, esto es, si se ha utilizado cifrado. Además, el uso de cifrado también garantiza la protección de nuestros datos contra análisis forenses que se puedan realizar en el dispositivo.

Por el modo como se almacenan los datos en la memoria, incluso un borrado normal no es suficiente para que los datos (o parte de ellos) no puedan ser recuperados. Eso sí, si están cifrados, aunque los datos puedan ser recuperados, si no se dispone de la clave correcta para descifrarlos, se mantendrán protegidos.

Finalmente, es importante tener siempre una copia de seguridad de los datos. Sobre todo si hay que realizar un borrado de estos.

5.3. Caso de estudio: WaveSecure

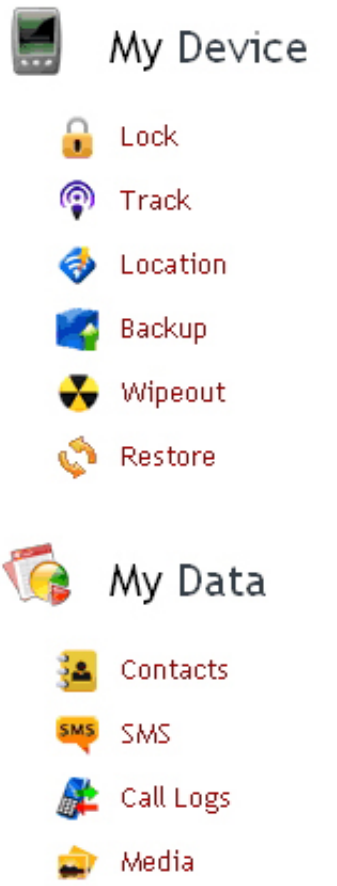
Una aplicación que permite garantizar la privacidad de los datos almacenados incluso después del robo del dispositivo móvil es WaveSecure. Hay más de una aplicación con las mismas características, pero revisaremos esta por ser una de las más extendidas y de las más completas. Además, esta aplicación tiene detrás a McAfee, una compañía con un largo recorrido en la seguridad informática.

WaveSecure consta de dos partes: la primera es la página web https://www.wavesecure.com y la segunda, una aplicación cliente que se debe instalar en el dispositivo móvil. Esta aplicación está desarrollada para Android, BlackBerry, Symbian, Windows Mobile y Java. En este estudio, nos centraremos en la versión de Android, ya que es el sistema operativo que más se ha tratado en este módulo.

La página web es la interfaz de control para tu dispositivo móvil. Desde aquí se pueden realizar diversas tareas remotamente, como localizar, bloquear o acceder a los datos de tu dispositivo móvil. Sin embargo, antes de poder utilizar este servicio, es necesario crear una cuenta de usuario. En este registro se nos pide tanto nuestro número de teléfono como una contraseña. Adicionalmente, también se nos pide un segundo número de teléfono, que será utilizado como oyente de los cambios producidos en nuestro dispositivo móvil. Por ejemplo, se le enviará un SMS cuando cambiamos el SIM.

Página web de WaveSecure

Una vez hemos accedido a la web con nuestro número de teléfono y la contraseña, en la parte izquierda aparece un menú con todas las opciones que se pueden realizar. En la sección de tu dispositivo están las opciones: bloquear, rastrear, ubicar, copia de seguridad, borrado y restauración. En la sección de tus datos están las opciones: contactos, SMS, registro de llamadas y multimedia.

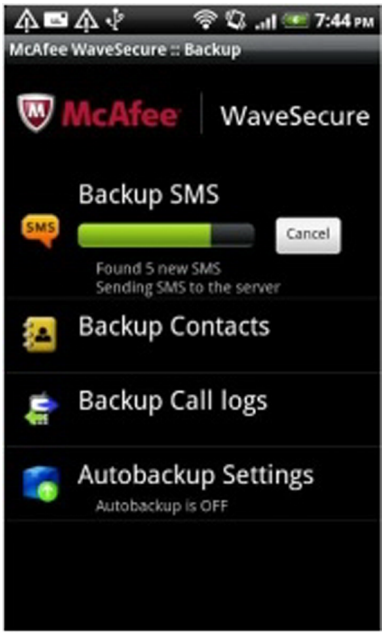


Menú de WaveSecure

En caso de robo de nuestro dispositivo móvil, pulsando la opción de bloqueo se inutiliza completamente nuestro dispositivo móvil y adicionalmente podemos personalizar el mensaje que se visualizará en la pantalla. Además, el dispositivo podrá ser desbloqueado únicamente si se dispone del código PIN de seguridad.

Una vez bloqueado el dispositivo, se puede obtener información de este, como el número de teléfono que se está utilizando actualmente en tu dispositivo móvil y su localización exacta, mediante una interfaz con Google Maps. Además, también podemos realizar una copia de seguridad de nuestros datos en la web para posteriormente poder restaurarlos. En caso de que no podamos recuperar nuestro dispositivo, con el fin de garantizar la privacidad de nuestros datos, podemos borrar remotamente todos los datos personales almacenados en el dispositivo.

Asimismo, desde el propio dispositivo se puede ir realizando copias de seguridad de una manera automática:



6. Prácticas de seguridad

Como hemos visto en este módulo, los dispositivos móviles ya deben ser tratados como un ordenador en cuanto a la seguridad se refiere, puesto que han heredado muchas de sus características. Por lo tanto, muchas de las prácticas de seguridad que aquí veremos serán similares a las que utilizamos cuando estamos delante de un ordenador, pero, como todavía lo vemos como un dispositivo inferior, tenemos una falsa sensación de seguridad. Por lo tanto, es importante seguir estas prácticas de seguridad cuando se utiliza un dispositivo móvil:

- **Activar el control de acceso inicial**. Este acceso puede ser mediante el código PIN o usuario y contraseña.

- **Configurar el bloqueo automático**. Después de un tiempo de inactividad es conveniente que el dispositivo se bloquee.

- **Activar autenticación para desbloquear**. Esta autenticación para desbloquear el dispositivo puede ser más simple y rápida que la inicial, como reconociendo un patrón dibujado en la pantalla.

- **Controlar las aplicaciones que se instalan**. Se deben tratar con precaución las aplicaciones que se instalen en el sistema, intentando bajarlas de fuentes de confianza y con una reputación positiva. También hay que revisar los permisos que estas aplicaciones requieren para su funcionamiento (en caso de que el sistema operativo limite las acciones de las aplicaciones por medio de permisos).

- **Mantener todo el software actualizado**. Con el fin de corregir lo mejor posible los problemas de seguridad, es importante mantener tanto las aplicaciones como el sistema operativo actualizados. Además, si es posible, hay que configurarlos para que realicen la actualización automáticamente.

- **Realizar copias de seguridad**. Es muy importante que periódicamente se realicen copias de la información importante que se almacena en el dispositivo. Además, los datos copiados tendrían que encontrarse fuera del dispositivo, por ejemplo en la web.

- **Cifrar la información delicada**. Este cifrado puede realizarse tanto utilizando los servicios que ofrece el sistema operativo como mediante aplicaciones de terceros.

- **Monitorizar el uso de recursos**. Se pueden detectar anomalías realizando un control de la utilización de los recursos del dispositivo móvil por parte

de las aplicaciones. Esto incluye revisión de la factura telefónica para detectar posibles usos fraudulentos.

- **Deshabilitar los sistemas de comunicación cuando no se utilicen**. Además de reducir el consumo energético, deshabilitar los sistemas de comunicación cuando no se utilizan puede evitar ataques. Los sistemas de comunicación únicamente se deben utilizar en redes de confianza.

- **Permitir control remoto**. En caso de robo, es importante tener una aplicación en el dispositivo móvil que permita controlarlo remotamente. Así, se puede localizar el dispositivo, recuperar sus datos almacenados o borrar los datos confidenciales para que no sean comprometidos. También se puede tener una aplicación que borre los datos automáticamente después de varios intentos de acceso fallidos.

- **Contactar con el proveedor de servicios en caso de pérdida**. En caso de pérdida del dispositivo, lo primero que hay que hacer es informar al proveedor de servicios para que efectúe el bloqueo del dispositivo.

- **Eliminar la información confidencial antes de desechar el dispositivo**. Al deshacerse del dispositivo, no sabemos en qué manos puede caer, por lo tanto es importante eliminar toda la información que contiene.

- **Tener sentido común**. Se deben seguir las mismas precauciones que se tienen con los ordenadores cuando tratamos con archivos adjuntos a correos electrónicos, enlaces desde SMS y, en general, navegación por Internet.

Bibliografía

Bibliografía complementaria

C. Fleizach; M. Liljenstam; P. Johansson; G.M. Voelker; A. Méhes. "Can You Infect Me Now? Malware Propagation in Mobile Phone Networks". En: *Proceedings of WORM 2007*. ACM Press.

Enlaces de Internet

http://www.android.com/

http://developer.apple.com/technologies/ios/

http://www.infospyware.com/. ¿Qué son los Malwares?

http://www.cnccs.es/. Malware en Smartphones CNCCS

http://www.genbeta.com/. Tapjacking, un problema de seguridad en los móviles Android

http://www.securitybydefault.com/. Aplicaciones de seguridad desde Android

http://www.diariopyme.com/. Decálogo de seguridad para dispositivos móviles