Introducción

Este libro surge con el propósito de acercar al lector a los aspectos más importantes en el desarrollo de aplicaciones para dispositivos móviles y en particular, a las características de nuevos entornos y lenguajes que le permitirán adquirir fácilmente experiencia en este campo. Con tal propósito, este libro puede servir de apoyo para estudiantes de los Ciclos Formativos de Grado Superior de **Desarrollo de Aplicaciones** y de las **Ingenierías Técnicas** y **Superiores**.

La alianza entre grandes compañías que apuestan por las tecnologías móviles ha favorecido la creación de plataformas para el soporte de aplicaciones desarrolladas para contextos como el entretenimiento, la educación, el turismo, entre otros, adaptándose al vertiginoso cambio tecnológico que vivimos en la actualidad y que exigen por tanto, adquirir los conocimientos y destrezas necesarios más demandados hoy en día en el mercado laboral.

Para todo aquel que use este libro en el entorno de la enseñanza (ciclos formativos o universidad), se ofrecen varias posibilidades: utilizar los conocimientos aquí expuestos para inculcar aspectos genéricos del desarrollo de aplicaciones para dispositivos móviles o simplemente centrarse en preparar a fondo alguno de ellos. La extensión de los contenidos aquí incluidos hace imposible su desarrollo completo en la mayoría de los casos.

Ra-Ma pone a disposición de los profesores una página web para el desarrollo del tema que incluye las soluciones a los ejercicios expuestos en el texto, un glosario, bibliografía y diversos recursos para suplementar el aprendizaje de los conocimientos de este módulo. Puede solicitar el medio de acceso a dichos recursos a *editorial@ra-ma.com*, acreditándose como docente y siempre que el libro sea utilizado como texto base para impartir las clases.

1 Análisis de tecnologías para aplicaciones en dispositivos móviles

OBJETIVOS DEL CAPÍTULO

- ✓ Describir la evolución, características y limitaciones de los dispositivos para el desarrollo de aplicaciones móviles.
- ✓ Dar una visión global de la Computación Ubicua.
- ✓ Analizar las tecnologías disponibles, los lenguajes y los entornos integrados de trabajo y compilación.
- ✓ Proporcionar los conceptos básicos y las herramientas que permitan al alumno adquirir las competencias necesarias para trabajar con los distintos entornos de desarrollo de aplicaciones para dispositivos móviles.

1.1 DISPOSITIVOS MÓVILES: TIPOS, HISTORIA Y EVOLUCIÓN

La revolución de los dispositivos para comunicaciones móviles tiene tan solo 20 años. Durante este tiempo la tecnología ha evolucionado desde la voz a la información inalámbrica y el uso de dispositivos móviles se ha convertido en algo cotidiano. Gracias a ello, en la actualidad tenemos la posibilidad de comunicarnos con cualquier persona, en cualquier momento y desde casi cualquier lugar.

La primera semilla para este gran desarrollo fue plantada por el célebre matemático escocés James Clerk Maxwell, quien formuló, en el año 1860, un par de ecuaciones cuya solución predijo la propagación de las ondas electromagnéticas a la velocidad de la luz. Se necesitaron 20 años para comprobar dicha predicción en un laboratorio y otros 20 años más para que se llevara a cabo la primera aplicación móvil.

Existen diversos tipos de dispositivos móviles, como los que se muestran en la Figura 1.1.

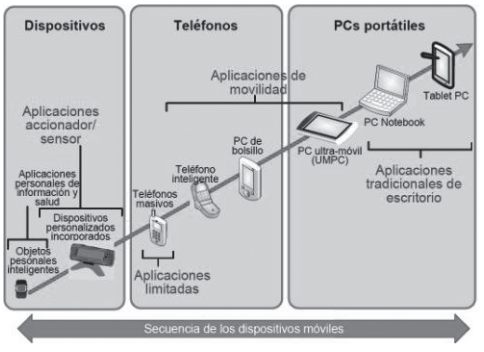

Figura 1.1. *Tipos de dispositivos móviles*

Desde los enormes terminales móviles a los teléfonos inteligentes, los teléfonos han recorrido un largo camino, en relativamente no demasiado tiempo. Durante ese recorrido, las tecnologías han ido mejorando para ofrecer al usuario una amplia gama de prestaciones. Así, hemos pasado de dispositivos con grandes dimensiones, costosos y con baterías que duraban alrededor de 60 minutos, a dispositivos con precios al alcance de los usuarios, con baterías que duran entre 4 y 10 horas y con cada vez mejores prestaciones como el acceso de alta velocidad a Internet.

Y gracias a estos avances, podemos hablar del concepto de Computación Ubicua, formulado por Mark Weiser en 1991, quien la definía como completamente opuesta a la realidad virtual, la cual pone a las personas dentro del ordenador para generar mundos, mientras que la computación ubicua pone al ordenador dentro del mundo de las personas.

La computación ubicua se fundamenta en cuatro paradigmas: *(i)* descentralización, *(ii)* diversificación, *(iii)* conectividad y *(iv)* simplicidad.

La **descentralización** se caracteriza por el paso de la era mainframe en la que la potencia estaba centralizada, a la era PC, en la que se define la arquitectura cliente-servidor. A partir de este paradigma se empezó a hablar de la

sincronización como un aspecto fundamental para mantener los datos actualizados. La descentralización benefició a los proveedores, pues les dio más flexibilidad para proporcionar servicios.

Por su parte, la **diversificación** surgió de la necesidad de ofrecer el mismo servicio, pero adaptado a diferentes dispositivos, ya que hasta el momento se habían desarrollado herramientas especializadas para dispositivos con funcionalidades específicas, dependientes del entorno y del grupo de usuarios. Así pues, la diversificación permitió tener la misma función desde varios dispositivos.

El desarrollo de dispositivos trajo consigo un problema, la integración entre plataformas. Por lo tanto fue necesaria la definición de estándares y protocolos que permitieran una *conectividad* sin límites, en la que cualquier tipo de software se pudiera ejecutar sobre cualquier plataforma y red. Este fue el punto de partida para el desarrollo de importantes estándares como WAP (*Wireless Access Protocol*, Protocolo de Acceso Inalámbrico), UMTS (*Universal Mobile Telecommunications System*, Sistema Universal de Comunicaciones Móviles) o BlueTooth (protocolo de comunicación para redes inalámbricas mediante radiofrecuencia).

El último paradigma de la computación ubicua es la *simplicidad*, centrada en el diseño de dispositivos con interfaces de usuario intuitivas. Apuesta además por la integración entre el software y el hardware para lograr mejor cobertura de las necesidades del usuario final, proporcionando además un acceso a los datos rápido y de forma sencilla.

Así pues, con la evolución de los dispositivos móviles las tendencias son: que la computación ubicua pueda ofrecer conectividad en tiempo real 24/7, que los precios de los dispositivos y de los planes de datos ofrecidos por las operadoras de telecomunicaciones bajen, que la infraestructura de las redes mejore, que se pueda tener acceso a todo desde cualquier parte de la nube y que se dé una gran explosión en el desarrollo de aplicaciones para dispositivos móviles.

A continuación, las Figuras 1.2, 1.3 y 1.4 resumen la historia y evolución de los dispositivos móviles entre los años 1973 y 2011.

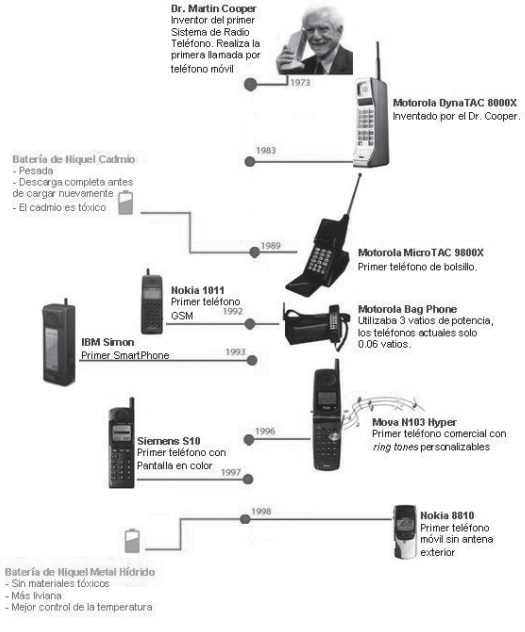

Figura 1.2. *Evolución de los dispositivos móviles (1)*

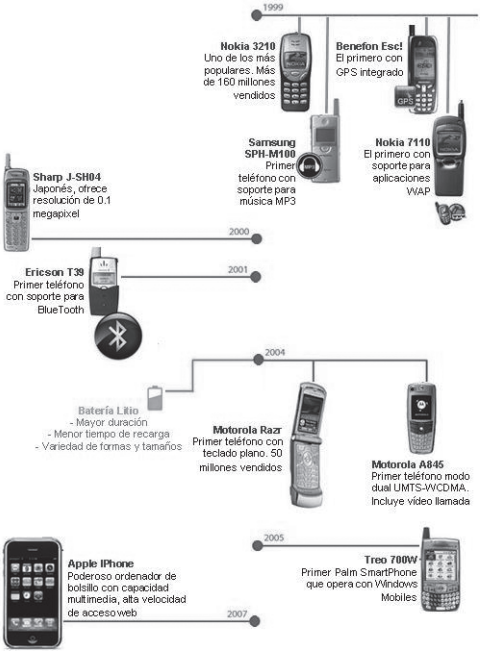

Figura 1.3. *Evolución de los dispositivos móviles (2)*

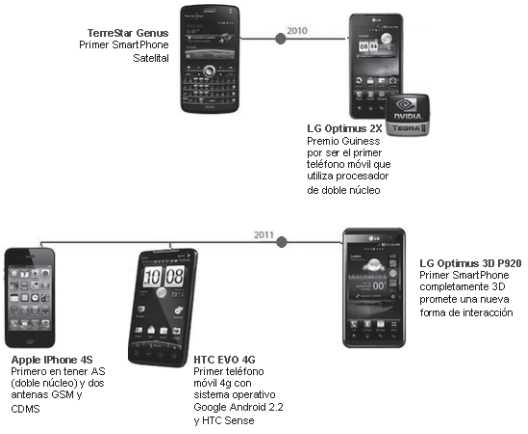

Figura 1.4. *Evolución de los dispositivos móviles*[1] *(3)*

[1] *http://webdesignerdepot.com/2009/05/the-evolution-of-cell-phone-design-between-1983-2009*
http://www.pcworld.com/article/131450/in_pictures_a_history_of_cell_phone.html
http://www.pcworld.com/article/173033/evolution_of_cell_phone.html
http://ginva.com/2011/05/the-evolution-of-cell-phone-design-between-1983-2011
http://blog.smartphonestab.com/evolution-of-cell-phone-batery
http://gizmodo.com/5500343/prints-htc-evo-the-firts-ever-4g-phone-meet-the-new-terrific

1.2 CARACTERÍSTICAS Y LIMITACIONES EN EL DESARROLLO DE APLICACIONES PARA DISPOSITIVOS MÓVILES

Las aplicaciones móviles no son aplicaciones de escritorio adaptadas para dispositivos con pantallas pequeñas, son por el contrario, aplicaciones diferentes por varias razones: *(i)* la capacidad para comunicarse desde cualquier lugar cambia la interacción del usuario con la aplicación, *(ii)* la interfaz de usuario para una pantalla y teclados pequeños difiere de forma significativa de la interfaz de una aplicación diseñada para un ordenador de sobremesa o un portátil, *(iii)* los tipos de canales de comunicación son diferentes, los dispositivos móviles incorporan capacidades de voz, mensajería, información de geolocalización y vídeo conferencia (en algunos teléfonos). Las mejores aplicaciones para móviles integran estas capacidades para optimizar la interacción del usuario con los datos y, por último, *(iv)* la naturaleza de las redes inalámbricas, aunque las redes ofrecen capacidades de datos de banda ancha, estas pueden variar, dependiendo de la calidad de la señal y de la disponibilidad de conexión de la red, en particular si se trata de usuarios móviles.

1.2.1 LIMITACIONES EN LA EJECUCIÓN DE APLICACIONES PARA DISPOSITIVOS MÓVILES

El desarrollo de aplicaciones para dispositivos móviles requiere tener en cuenta las limitaciones que podemos encontrar a la hora de ejecutarlas. Estas limitaciones están relacionadas, principalmente, con las características hardware y de conexión, asociadas a cada dispositivo móvil.

A pesar de que la capacidad de procesamiento y la memoria de los dispositivos móviles han ido mejorando con el paso del tiempo, las aplicaciones deben diseñarse evitando la sobrecarga de elementos multimedia, que exijan del dispositivo ciclos de procesamiento demasiado largos.

El tamaño de las pantallas y la iluminación también son factores determinantes en el diseño de aplicaciones. No hay que olvidar que la mayoría de los datos de entrada que proporciona el usuario son introducidos con una sola mano y en muchas ocasiones en movimiento. Por lo tanto, en la medida de lo posible se le debe facilitar al usuario la entrada de datos por pantalla, a través de los botones de navegación propios de cada dispositivo, de modo que pueda ir seleccionando opciones (desde un menú desplegable, por ejemplo), y que no tenga que rellenar campos de texto, de esta forma podrá realizar varias operaciones en poco tiempo.

La distribución de los elementos en la interfaz marca la diferencia, no solo en la forma en la que se presenta la información, sino también en cómo navega el usuario a través de ella. Lo fundamental es ofrecer interfaces a través de las cuales la entrada de los datos se realice de la forma más intuitiva y natural. Una buena opción es agrupar la información por funcionalidades o por jerarquías, de modo que el usuario acceda a la información que necesita en cada momento, sin tener que pasar por pantallas o información irrelevante para el tipo de operación que desea realizar.

Si además tenemos en cuenta que cada vez que accedemos desde nuestro teléfono móvil a Internet estamos pagando una tarifa por cantidad de bytes descargados y no por tiempo de uso de la red de datos, es necesario que el envío y la recepción de datos se realicen dentro de un tiempo de espera aceptable. Durante el proceso de envío y/o recepción de datos, debemos mantener informado al usuario sobre el progreso de esta operación y el tiempo estimado para la finalización de la misma. El tiempo de espera entre la petición de los datos y el momento en el que empieza a llegar la respuesta es lo que se conoce como latencia. Hay que tener en cuenta que en el caso de aplicaciones para dispositivos móviles la latencia es mucho mayor que la de una aplicación web normal.

También hay que tener presente que cuando se trata del desarrollo de aplicaciones para teléfonos móviles, la función de teléfono (es decir, realizar y/o recibir llamadas), tiene la prioridad más alta. Por lo tanto, en el momento en el que se reciba una llamada, la aplicación debe proporcionar la forma de mantener el estado en el que ha sido interrumpida, para volver a ella cuando la llamada termine y el usuario confirme que quiere volver al punto en el que se interrumpió su ejecución.

En cuanto a las conexiones, debemos tener presente que pueden fallar y de hecho es algo que sucede con relativa frecuencia, por ejemplo, por la falta de cobertura en determinadas áreas. Esto implica que no podemos dar por sentado que tendremos el acceso a Internet garantizado y que podremos obtener los datos necesarios para la ejecución de la aplicación.

Aunque el uso de emuladores es útil para simular el comportamiento de la aplicación, hay que tener presente que se ejecutan en equipos cuya capacidad de procesamiento es superior a la del dispositivo. Por lo cual, es muy recomendable probarla cuanto antes en el dispositivo para el cual ha sido diseñada y así evitar comportamientos indeseados.

1.2.2 ENFOQUES PARA EL DESARROLLO DE APLICACIONES MÓVILES

Para el diseño de aplicaciones para dispositivos móviles se consideran cuatro enfoques principales:

- **Clientes nativos**: las aplicaciones se escriben en lenguajes de bajo nivel como C o ensamblador y compilados en un lenguaje de máquina para un grupo específico de procesadores y configuraciones hardware, y luego se ejecuta como código nativo en esos dispositivos móviles. El principal beneficio de este enfoque es la capacidad para utilizar al completo todas las características de un hardware determinado. Este enfoque se debe utilizar si el dispositivo móvil tiene un hardware especializado al que solo se puede acceder utilizando una API de C. Otro beneficio de desarrollar en clientes nativos es que se pueden ajustar los bucles, la gestión de memoria y el acceso a datos para lograr un alto nivel de rendimiento de la aplicación. Sin embargo esta potencia y control tiene un precio. Los desarrollos se hacen para dispositivos que tienen hardware similar, por tanto es necesario mantener diferentes versiones del código fuente para todos los clientes. Además, los lenguajes de bajo nivel no son tan productivos y por tanto la cantidad de código que hay que escribir para que la aplicación realice una determinada operación es bastante grande. Por ejemplo, si se utiliza C, las asignaciones de memoria se deben hacer manualmente, lo cual añade no solo complejidad y líneas de código, sino también la posibilidad de errores. Resumiendo, se puede utilizar este enfoque cuando la utilización completa del hardware y el alto desempeño son primordiales. Cuando hay que dar soporte a un gran número de clientes móviles o cuando el tiempo de desarrollo es clave, este enfoque no es la mejor opción, a menos que sea la única opción.
- **Clientes JME** (*Java Platform Micro Edition*, Plataforma Java Micro Edición): las aplicaciones se escriben en Java y se compilan para ejecutarse contra una máquina virtual Java (JVM), diseñada específicamente para computadoras de mano y clientes móviles. Este enfoque proporciona dos beneficios principales: tiempo de desarrollo rápido y la posibilidad de utilizar el mismo código base en un gran número de dispositivos. La cantidad de código necesario escrito en Java es por lo general menor que si se escribe en un lenguaje de bajo nivel como C. Esto se debe a que la máquina virtual se encarga de manejar automáticamente muchas de las operaciones tediosas, entre ellas el manejo de memoria. Esto significa que para realizar la misma operación se necesitan pocas líneas de código y menos errores. Además, la JVM está disponible en varios dispositivos, por lo que la aplicación cliente trabajará en diferentes dispositivos sin necesidad de mantener múltiples versiones.

Sin embargo, debido a las diferencias en las implementaciones, será necesario probar la aplicación sobre cada plataforma. Los principales inconvenientes de este enfoque son el desempeño y la flexibilidad. Si la aplicación debe caber en un pequeño espacio de memoria o realizar operaciones que requieran mucha CPU (*Central Processing Unit*, Unidad Central de Procesamiento), entonces Java no es adecuado. Además, es posible que uno o más dispositivos no tengan la JVM. Resumiendo, si la aplicación a desarrollar tiene una interfaz estándar y no requiere acceso especial al hardware, entonces Java es una buena elección.

- **Clientes basados en web**: son similares a los clientes web estándar excepto porque se debe tener en cuenta el diseño de la página. Un cliente basado en web se ejecuta dentro del navegador web del dispositivo, desde el cual accede a la página que el servidor web envía utilizando las mismas facilidades que un cliente web de sobremesa. El principal beneficio de este enfoque es que simplifica el mantenimiento del cliente independientemente de que las características de las versiones cambien. Sin embargo, las aplicaciones deben desarrollarse para que sean compatibles con el navegador del dispositivo. De otra parte, con la proliferación de las características de la Web 2.0, es más sencillo crear clientes más ricos y dinámicos. Los principales inconvenientes para un cliente basado en web son las características, el desempeño y el modelo de conexión. Con un cliente web, el dispositivo requiere una conexión a un servidor web para que la aplicación móvil se pueda ejecutar. Esto significa que si la aplicación cliente necesita realizar trabajo fuera de línea, que se puede procesar por lotes y enviar a los servidores centrales solo unas cuantas veces al día, el enfoque de cliente basado en la web no es la opción adecuada. Además, si la aplicación requiere una interfaz de usuario dinámica o acceso a un hardware I/O (entrada/salida) especial, es necesario considerar una implementación alternativa. En resumen, el cliente basado en web es una buena opción si el cliente software tiene una interfaz de usuario simple, que puede mantener una conexión para realizar trabajo útil.

- **Clientes basados en middleware** (software que ayuda a una aplicación a interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos): el diseño de este tipo de aplicaciones utiliza un conjunto de herramientas y tiempos de ejecución para abstraer la aplicación y las tareas de adquisición de datos lejos de cualquier dispositivo. El principal beneficio de este tipo de implementación es el desarrollo rápido y el mantenimiento del código específico del dispositivo. En el enfoque midleware se crea una aplicación utilizando un conjunto de herramientas de diseño para terceros, en un sistema operativo propietario. Además controla aspectos como lo que se muestra en pantalla, los diálogos, los datos en el lado cliente, la gestión del estado de conexión del dispositivo y la gestión de los datos fuera de línea. El enfoque midleware funciona mejor cuando la aplicación necesita acceder y cambiar datos desde un servidor central. Por lo general, se puede gestionar y desplegar aplicaciones rápidamente. El inconveniente es el enfoque potencialmente estrecho de las capacidades y la incapacidad para utilizar funciones hardware especializadas sobre los dispositivos. Las plataformas midleware generan código que no es adecuado para situaciones de alto rendimiento. Finalmente, el enfoque midleware es muy similar al enfoque JME, pero más especializado y generalmente construido en torno a las bases de datos de las aplicaciones.

En conclusión, cada selección tiene sus beneficios e inconvenientes que determinan qué enfoque de desarrollo encaja mejor con las necesidades de desarrollo. Si la aplicación requiere un alto desempeño o un código para funcionalidades específicas del hardware, una aplicación nativa será la mejor opción. Si el cliente necesita ejecutarse sobre una amplia gama de dispositivos y manipulación de datos fuera de línea, el enfoque midleware es el mejor. Si es una aplicación sencilla en la que la operación puede depender del estado de conexión, entonces el cliente basado en web es una buena opción. Finalmente, si la aplicación requiere una interfaz de usuario especializada o un código del

lado cliente especializado, pero no requiere el poder de una aplicación nativa, entonces, el enfoque JME es una buena opción. La Tabla 1.1 recoge los beneficios y los inconvenientes de cada opción.

Tabla 1.1 Beneficios e inconvenientes de los diferentes enfoques para el desarrollo de aplicaciones para dispositivos móviles

Arquitectura	Beneficios	Inconvenientes
Cliente nativo	- Aplicaciones sofisticadas y control sobre el entorno local - Capacidades multitarea sobre muchas plataformas	- El más alto nivel de esfuerzo de desarrollo - Diferente código base para diferentes dispositivos
Java ME	- El mismo código base puede soportar múltiples aplicaciones - Aplicaciones más sofisticadas	- Algunos límites de capacidad (no multitarea) - Requiere prueba y adaptación para las plataformas
Navegador web	- Desarrollo rápido - No requiere mantenimiento del código cliente - Métodos Web 2.0 disponibles - Trabajo con un amplio rango de dispositivos móviles	- Menos sensible y capas que las aplicaciones nativas
Midleware móvil	- Alto nivel de capacidad con reducido esfuerzo de desarrollo	- Tasas adicionales de licencia - Curva de aprendizaje y esfuerzo de integración potencialmente grande

Los proveedores cuentan con centros de desarrollo a disposición de los usuarios, por tanto es de gran ayuda visitarlos y analizar las sugerencias en cada caso. Algunos de ellos se muestran en la Tabla 1.2.

Tabla 1.2 Centros de desarrollo

Proveedores	Centros de desarrollo
Sony Ericsson	*http://developers.sonyericsson.com*
Nokia	*http://forum.nokia.com*
Samsung	*http://developers.samsungmobile.com*
Motorola	*http://developer.motorola.com*
BlackBerry	*http://www.blackberry.com/developers*
IPhone	*http://developer.apple.com/iphone*
LG Mobile	*http://developer.lgmobile.com*

1.3 TECNOLOGÍAS DISPONIBLES

Se presenta en este punto una breve introducción a las principales tecnologías y sistemas operativos para dispositivos móviles que hay actualmente.

1.3.1 ANDROID

Android es una plataforma formada por un conjunto de software en estructura de pila (*software stack*) que incluye un sistema operativo, software para conectar aplicaciones (*middleware*) y aplicaciones base. El SDK (*Software Development Kit*, Kit de Desarrollo de Software) de Android proporciona varias herramientas y API (*Applications Programming Interface*, Interfaz de Programación de Aplicaciones) que son necesarias para desarrollar aplicaciones Android. Estas aplicaciones se desarrollan en lenguaje Java.

Android está desarrollado por *Open Handset Alliance* (OHA), una agrupación de 78 compañías para desarrollar estándares abiertos para dispositivos móviles y que está liderada por Google. Inicialmente Android fue desarrollado por la compañía Android Inc., que fue comprada en el año 2005 por Google. El sistema operativo se anunció el 5 de noviembre de 2007. Google libera la mayoría del código Android bajo una licencia Apache (licencia libre y de código abierto). Desde su creación ha ido pasando por diferentes versiones, desde la versión primera (1.0) hasta la actual (4.0, denominada también *Ice Cream Sandwich*).

Android se ha convertido de forma rápida en uno de los SO de móviles con mayor presencia. Actualmente[2] hay más de 200 millones de dispositivos móviles Android activados y cada día se activan más de 550.000 nuevos dispositivos en 137 países y regiones. Esto representa un 32,9% de la cuota del mercado a escala mundial de los teléfonos Smartphone, por delate de Symbian OS y iOS. Android presenta varias ventajas que han hecho posible su gran éxito: *(i)* el ser código abierto con licencia Apache, lo cual permite que un desarrollador pueda, no solo ver el código, sino mejorarlo y ampliarlo, *(ii)* dar libertad al usuario del dispositivo para instalar el software que crea oportuno sin imponer que sea software propietario, *(iii)* los desarrolladores tienen libertad para desarrollar cualquier software y ofertarlo a los usuarios (dispone de una amplia comunidad de desarrolladores), *(iv)* no está limitado a determinados proveedores, operadoras o fabricantes, etc.

La Figura 1.5 muestra la arquitectura del sistema operativo Android que está organizado en cuatro grandes bloques:

- **Aplicaciones** (*Applications*): proporciona un conjunto de aplicaciones de usuario como cliente de correo electrónico y de mensaje de texto SMS, calendario, mapas, navegador, agenda de contactos, etc.

[2] *http://www.android.com/developers/*

- **Framework de Aplicaciones** (*Applications Framework*): es un conjunto de aplicaciones en forma de servicios y sistemas disponibles al desarrollador que le facilitan aspectos como acceso al hardware del dispositivo, acceso a información de localización, servicios de *background*, lanzar alarmas y notificaciones, etc. Alguno de estos servicios o sistemas más importantes se describen a continuación. Sistema de Visualización (*View System*), que facilita el desarrollo de aplicaciones con potentes interfaces de usuario, Suministrador de Contenidos (*Content Providers*), gestiona el acceso a datos entre aplicaciones, Gestor de Fuentes (*Resource Manager*), gestiona el acceso a fuentes que no son de la aplicación como imágenes, texto, etc., Gestor de Actividades (*Activity Manager*), gestiona el ciclo de vida de las aplicaciones, etc.

- **Librerías** (*Libraries*): Android incluye un conjunto de librerías C/C++ que usan otros módulos del sistema operativo y están accesibles a los desarrolladores de aplicaciones a través del Framework de Aplicaciones. Algunas de estas librerías incluyen funcionalidades como la librería estándar de C para dispositivos embebidos Linux (*System C library*), funcionalidades multimedia para vídeo, audio e imagen (*Media Libraries*), visualización de gráficos 2D y 3D (*Surface Manager*), funcionalidades par optimizar el tratamiento de gráficos 3D (*3D Libraries*), motor de navegación web que da soporte al navegador web de Android (*LibWebCore*), motor de base de datos relacional (*SQLite*), etc.

- **Entorno de Ejecución** (*Android Runtime*): el entorno de ejecución está formado por las librerías del núcleo del sistema operativo (*Core Libraries*) y por la máquina virtual (*Dalvik Virtual Machine*). Es el motor que ejecuta los programas. El módulo *Core Libraries* proporciona muchas de las funciones disponibles en la librería base de Java así como funciones específicas de Android. La máquina virtual Dalvik se encarga de interpretar el código del programa y de ejecutarlo apoyándose en el Kernel del sistema operativo. Cada aplicación pone en ejecución su propio proceso, el cual tiene asociado una instancia propia y exclusiva de la máquina virtual Dalvik. Dalvik ha sido desarrollada de tal forma que puede haber múltiples máquinas virtuales (MV) ejecutándose a la vez en un mismo dispositivo (precisamente ésta fue la razón por la que no se utilizó la MV de Java ME, *Java Mobile Edition*). Dalvik es una máquina virtual basada en registros y ejecuta ficheros de código con un formato especial, denominado Ejecutable Dalvik (ficheros con extensión *.dex*). Este formato está especialmente pensado para optimizar el uso de la memoria. No obstante, esto no significa que no podamos usar un compilador de Java para desarrollar aplicaciones para esta MV. Dalvik es capaz de ejecutar clases compiladas por un compilador de Java siempre que hayan sido transformadas previamente al formato *.dex* por herramientas de poscompilación (*dx tools*).

- **Linux Kernel**: Android se basa en la versión 2.6 de Linux para implementar servicios bases de sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y varios drivers, etc., todos ellos integrados en el Kernel. El Kernel también constituye una capa de abstracción entre el hardware del dispositivo y el resto de los componentes y módulos de la pila de software del sistema operativo.

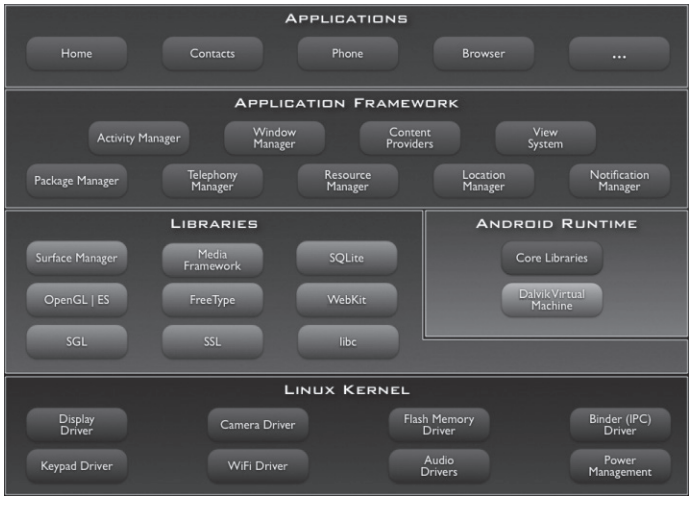

Figura 1.5. *Estructura de la pila de software del S.O Android (fuente: http://developer.android.com/guide/basics/what-is-android.html)*

1.3.2 BLACKBERRY

BlackBerry está desarrollado por la compañía RIM (*Research In Motion*). Los móviles BlackBerry destacan principalmente por su capacidad de enviar y recibir correo electrónico por Internet a través de los operadores que ofrecen este servicio. Actualmente, del mercado mundial, BlackBerry asume un 2,9% de los móviles vendidos, y el 11% de los *smartphone*, si bien su mayor cuota de mercado está en EE.UU.

Los dispositivos BlackBerry montan el sistema operativo BlackBerry OS, desarrollado por RIM. Este sistema operativo es propietario, con lo cual no hay información pública relevante sobre su diseño ni arquitectura. A día de hoy la última versión del sistema operativo es BlackBerry OS 7. Las versiones del sistema operativo tienen un Kernel que se basa en Java, montando la mayoría de los dispositivos arquitecturas ARM. ARM no construye sus propios chips CPU pero da licencia a terceros para que los fabriquen. El sistema operativo divide la memoria del dispositivo en tres secciones:

- **Memoria de aplicación**: espacio de memoria destinado para albergar las aplicaciones.
- **Memoria de dispositivo**: espacio para almacenar ficheros y otras fuentes.
- **Memoria para tarjeta**: espacio de almacenamiento adicional.

BlackBerry OS es un sistema operativo multitarea. Esto significa que puede ejecutar más de una aplicación a la vez. Por ejemplo, mientras que se está realizando una llamada, el usuario puede cambiar y consultar el calendario o los contactos sin cortar la llamada. Estas aplicaciones quedan en modo *background* y continúa así su ejecución dando la ejecución e interacción del usuario a otras aplicaciones. Hay que tener en cuenta que muchas aplicaciones ejecutándose en *background* puedan dar la sensación de que no hay mucha carga de ejecución para la máquina, pero lo cierto es que hay un alto consumo de memoria.

1.3.3 SYMBIAN

El sistema operativo Symbian viene como evolución del sistema operativo Epoc, este fue desarrollado por Psion en sus agendas electrónicas durante los 80. Symbian es el resultado de adaptar Psion a dispositivos móviles y tiene diferentes variantes según el dispositivo en el que se utilice.

Concretamente Symbian Inc desarrolló el sistema operativo base y les vendió la licencia a los distintos fabricantes de teléfonos móviles. Estos a su vez construyen interfaces de usuario sobre este sistema operativo base y personalizan el sistema para propósitos específicos.

De todos los fabricantes de móviles es Nokia el que más ampliamente ha utilizado este sistema operativo. Ha desarrollado varias versiones del sistema, que denomina Series, está la Serie60, Serie70, Serie80 y Serie90, siendo la Serie60 la que más amplio uso ha tenido.

Sony Ericcson por ejemplo también utiliza Symbian, con su plataforma UIQ, en sus dispositivos, por ejemplo, en el modelo P800 o Motorola en el A1000.

Como hay muchas combinaciones de SO y UI, habrá también muchos kits de desarrollo (SDK), por ejemplo para la Serie60 hay SDK versión 6.1, 7.0, 8.0 y 9.0. Si nos centramos en la Serie60, tenemos que para cada edición de un SDK hay ampliaciones con mejoras llamadas paquetes de características o FP (*Feature Pack*), así tenemos:

- SDK for 1st Edition.
- SDK for 1st Edition, FP1.
- SDK for 2st Edition.
- SDK for 2st Edition, FP1.
- SDK for 2st Edition, FP2.

Además, para cada kit de desarrollo hay una variante en función del IDE en el que se instale por lo que habrá variantes diferentes.

Normalmente el entorno de desarrollo viene preparado para estar programado en C o C++ y aporta el emulador de la Serie para la que está desarrollado. En las siguientes secciones comentaremos más acerca de la estructura de un programa en Symbian y las características de sus emuladores.

1.3.4 PALM OS (WEBOS)

Palm OS es el sistema operativo de los dispositivos móviles desarrollados por la empresa Palm Inc. Esta empresa tuvo un gran auge con sus agendas electrónicas o PDA. Comenzó su actividad en 1996, creando Palm OS, un sistema operativo fácil de utilizar con pantallas táctiles e interfaces de usuario gráficas.

Palm OS ha tenido una gran evolución de versiones desde la 1.0 hasta la 5.0, luego pasó a llamarse Palm OS Cobalt. Este último es un sistema operativo basado en Linux, que evolucionó para denominarse webOS en 2009, el primer dispositivo que lo incluía fue Palm Pre. La compañía Palm Inc, fue adquirida en 2010 por HP y, actualmente, HP utiliza webOS en sus dispositivos móviles y tablets, como Pixi, Veer y HP TouchPad.

En diciembre de 2011 la compañía Hewlet-Packard liberó el código de webOS y en la actualidad es un sistema de código abierto.

1.3.5 WINDOWS PHONE

Este sistema operativo fue lanzado a finales del año 2010 tras dos años de desarrollo. Entre las novedades se encuentra la denominada interfaz de usuario "Metro" basada en la utilización de mosaicos dinámicos que muestran información útil al usuario. Además se introduce el concepto de HUB, en donde se centralizan las acciones y las aplicaciones se agrupan por el tipo de actividad que representan. Por lo tanto, encontraremos diferentes HUB, por ejemplo, de Office, Xbox Live, Imágenes o Zune, desde los cuales tenemos acceso a tareas específicas. También incluye el motor de Internet Explorer 9, con soporte para HTML5, multitarea en aplicaciones de terceros e integración con Xbox 360 y Kinet.

Este sistema ha evolucionado en poco tiempo, desde la versión 7 (estable desde 2010), hasta la 7,5 y ya se habla de la versión 8. [1][3].

Analicemos la arquitectura de Windows Phone desde el punto de vista de su modelo hardware y software.

- **Modelo de Hardware**: para ejecutar Windows Phone y asegurar la consistencia de todos los usuarios del sistema, es necesario que el teléfono cuente con unas características mínimas, o lo que se conoce como Chassis 1, que constituyen las especificaciones que debe seguir todo fabricante que quiera crear terminales con soporte para este SO:
  - *Procesador*: ARMv7 Cortex/Scorpion a 1 GHz.
  - *Procesador gráfico*: soporte hardware completo de DirectX9.
  - *Memoria*: RAM 256 MB y ROM 8 GB.
  - *Sensores*: A-GPS, acelerómetro, brújula, iluminación y proximidad.
  - *Cámara*: 5 Mpx con *flash* y botón físico de disparo.
  - *Multimedia*: aceleración de audio y vídeo por hardware.
  - *Pantalla*: capacitiva, resolución 800 × 480.
  - *Botones físicos*: Inicio, Buscar y Atrás.

  Las especificaciones se irán adaptando a los nuevos tipos de teléfonos móviles, manteniendo la experiencia del usuario independientemente del dispositivo que utilice.

- **Modelo Software**: Windows Phone se basa en el Windows CE 6.0 R3, el cual tiene soporte para Silverlight Mobile, Internet Explorer Embedded, entre otras tecnologías. También soporta Flash Lite de forma nativa en el sistema (no disponible en la versión Woindows Phone 7.5).

  La Shell y la plataforma de aplicaciones residen en memoria de usuario, mientras que el Kernel, los drivers y el sistema de archivos, networking, el sistema de rendering y gráficos y el sistema de actualizaciones, residen en el espacio de Kernel. El sistema es de 32 bits, con lo cual maneja 4 GB de memoria, dos de los cuales son para procesos y los otros dos para el Kernel.

[3] Consultar bibliografía en la página web del libro.

La Figura 1.6 representa el modelo de software en el SO Windows Phone.

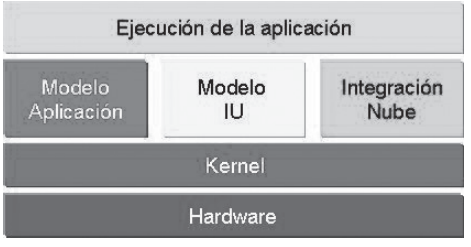

Figura 1.6. *Modelo de software en Windows Phone*

- **Modelo de aplicación**: las aplicaciones en Windows Phone se muestran en forma de paquetes XAP. Este es un archivo comprimido que contiene los recursos originales de la aplicación que estamos desarrollando. Para instalar una aplicación en el dispositivo, es necesario hacerlo a través del Marketplace (la tienda oficial de Microsoft), previo registro como desarrolladores, si nos interesa vender la aplicación. A cada aplicación se le asigna un ID único y un certificado de seguridad una vez la aplicación ha sido aprobada por el Marketplace. Si comprobamos el modo del certificado, podremos saber si estamos en modo de prueba o en modo completo, pudiendo cambiar la forma en la que la aplicación se comporta y evitando así la creación de versiones "lite" tal y como ocurre con otras plataformas.
- **Modelo de IU**: la interfaz de usuario está formada por: *(i)* elementos, es decir, todo control que se muestra al usuario, *(ii)* páginas, una asociación lógica de elementos y *(iii)* sesiones, conjunto de interacciones que realiza un usuario sobre la aplicación. También puede incluir otras aplicaciones.
- **Integración con la nube**: en la versión actual del kit de desarrollo no existen API que permitan a las aplicaciones acceder directamente a estos servicios, pero se espera que lo hagan en próximas actualizaciones.
- **Ejecución de aplicaciones**: Windows Phone proporciona dos *frameworks*: *(i)* Silverlight, permite crear aplicaciones multimedia que se ejecuten de forma nativa en Windows Phone, con una interfaz creada en XAML; *(ii)* XNA, es una solución Multiscreen (Xbox, Windows, Windows Phone) 2D y 3D que permite crear juegos de calidad profesional. Ambos *frameworks* se ejecutan sobre sandbox de .NET, lo cual facilita el acceso a hardware, sensores, almacenamiento, etc. Esto implica que la aplicación nunca tenga acceso nativo al sistema y que se ejecute aislada del sistema y de otras aplicaciones, por lo que no podrán compartir espacio de almacenamiento ni ningún otro tipo de información, a menos que utilicemos para ello servicios externos a la nube. La nueva versión de las herramientas de desarrollo para Mango (Windows Phone 7.5) permite tener aplicaciones mixtas, en las que se utilice Silverlight para la IU y XNA para los gráficos 3D, lo cual ofrece al usuario una experiencia interactiva más rica.

1.3.6 IOS

A mediados de 2007 la tecnología Apple nos ofreció iOS (inicialmente llamado IPhone OS), desarrollado originalmente para el IPhone y con él, una nueva definición del teléfono móvil. Más tarde fue introducido en el iPod Touch y actualmente en el IPad. Las actualizaciones de este S.O se enumeraron desde la 1.x hasta la 1.1.5. La versión 1.0 incorporaba aplicaciones como Mail, Fotos, iPod, Calculadora, entre otras, presentes en las versiones actuales y

que no han sido modificadas prácticamente ni en sus interfaces ni en sus funcionalidades. Un año después, en 2008, se lanzó el iPhone OS 2.0, cuyas actualizaciones llegaron hasta la 2.2. Comenzó entonces la revolución de las aplicaciones móviles y uno de los modelos de negocio más productivos existente hoy en día. En el año 2009 se lanzó el iPhone OS 3.0 que evolucionó hasta la versión 3.1.3, la cual incluía el Spotlight (para realizar búsquedas en el dispositivo), también ofrecía la posibilidad de incluir la API de Google Maps, las operaciones de copiar/cortar/pegar, interconexión por BlueTooth o P2P y librerías GPS. Esta versión fue soportada por los iPhones e iPads de primera generación. En 2010, empezó a llamarse iOS y se realizó el lanzamiento de la versión 4.0 del S.O. La versión 4.1.2 fue la última en los iPhone 3G y los iPod Touch de segunda generación. A finales de 2011 se lanzó la versión iOS 5, con una interfaz mejorada y funcionalidades como la presencia de un asistente personal Siri, facilidades para la sincronización sin cables, un centro de notificaciones mejorado, el servicio de iMessage, la navegación web con pestañas, entre otras. La versión iOS 5 solo se podrá instalar en los Apple TV (segunda generación), iPhone 3GS, iPhone 4, iPhone 4S, iPod Touch (tercera y cuarta generación) y en los dos modelos de IPad actuales (ver Figura 1.7).

Figura 1.7. *Evolución del sistema operativo iOS*
(fuente: http://sites.google.com/site/tecnologiaiostm/historia)

La arquitectura iOS se basa en capas, donde las capas más altas contienen los servicios y las tecnologías indispensables para el desarrollo de aplicaciones y las capas más bajas son las encargadas de controlar los servicios básicos.

- **Cocoa Touch**: posee tres *frameworks*: *(i)* UIKit, contiene todas las clases necesarias para el desarrollo de una interfaz de usuario; *(ii)* Foundation Framework, define las clases básicas de acceso, manejo de objetos y *(iii)* servicios del S.O. Estas *frameworks* proporcionan la API de Cocoa para desarrollar aplicaciones (ver Figura 1.8).

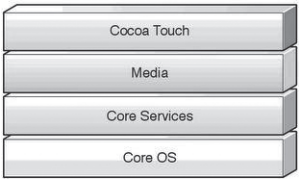

Figura 1.8. *Arquitectura del sistema operativo iOS*
(fuente: http://sites.google.com/site/tecnologiaiostm/desarrollo-de-aplicaciones/arquitectura-ios)

- **Media**: esta capa proporciona los servicios gráficos y multimedia a la capa superior.
- **Core Services**: esta capa proporciona los servicios fundamentales del sistema utilizados por todas las aplicaciones.
- **Core OS**: en esta capa se encuentran servicios de bajo nivel como los archivos de memoria, manejo de memoria, seguridad y *drivers* del dispositivo.

La Tabla 1.3 recoge algunas de las características más relevantes de los sistemas operativos.

Tabla 1.3 Sistemas Operativos

S.O Característica	Symbian	Windows Phone	IPhone	Palm	Android	BlackBerry
Versión 2012	Nokia Belle	Windows Phone 8 o Apollo	iOS 5.0.1	Garner OS 5.5	4.0.4 *Ice Cream Sandwich*	OS 7.0
Dispositivos	*Smartphone*	*Smartphone* PDA	IPhone 3 y 4S	PDA	*Smart-phones*	*Smart-phones*
Interfaz	Apuntador Teclado	Apuntador Teclado	Touch	Apuntador Teclado	Touch Apuntador	Teclado
SDK	Gratuito	Gratuito/IDE pagado	IPhone *Delevoper Program*	Gratuito	Gratuito	Gratuito
Aplicaciones	Nativas y JME	Nativas *Compact Framework*	Nativas y JME	Nativas y JME	Nativas y JME	JME
Firma	Obligatoria	Opcional	Obligatoria	Opcional	Opcional	Opcional
Proveedores	Nokia, Sony Ericsson, Samsung, Siemens	HP, HTC, Samsung, Dell	IPhone, IPhone 3G	Familia Palm	HTC, LG, Samsung	Familia BlackBerry

Tabla 1.4 Plataformas de desarrollo nativas

Sistema Operativo	IDE
Symbian	Eclipse + *Plugin* + SDK Visual Studio + SDK + *Plugin*
Windows Phone	Visual Studio + SDK
IPhone	XCode + SDK
Palm	Codewarrior + SDK

Tabla 1.5 Sistemas operativos

Proveedor	Eclipse (JME)	Netbeans (JME)	Visual Studio (CF)
Nokia	*Plugin* (SDK)	*Plugin* (SDK)	
Sony Ericsson	*Plugin* (SDK)	SDK	*Plugin* (SDK)
Samsung	SDK	*Plugin* (SDK)	SDK
Motorola	SDK	*Plugin* (SDK)	*Plugin* (SDK)
BlackBerry	*Plugin* (SDK)	SDK	
Android	*Plugin* (SDK)	SDK	
Palm	SDK	SDK	SDK
Windows based			SDK

1.4 DESARROLLO DE APLICACIONES MÓVILES

En esta sección se presenta una descripción de las herramientas que hay actualmente en el marco del desarrollo de aplicaciones para dispositivos móviles. El interés se centra en los lenguajes, entornos y emuladores más importantes y que están en uso.

1.4.1 LENGUAJES DE PROGRAMACIÓN

Existe una gran variedad de lenguajes para el desarrollo de aplicaciones para dispositivos móviles. A continuación se describen las principales características de cada uno de ellos.

Blackberry

Hay dos tipos de aplicaciones que se pueden desarrollar para dispositivos BlackBerry: aplicaciones Web y aplicaciones Java. El desarrollo Web de BlackBerry está pensado para usar estándares de web y herramientas my extendidas y conocidas como Eclipse o Microsoft Visual Studio. En el desarrollo de aplicaciones Java se utiliza el lenguaje de programación Java y se puede utilizar también varios entornos de desarrollo.

Las aplicaciones Java para BlackBerry se escriben en Java ME (*Java Micro Edition*). Los Smartphone de BlackBerry están diseñados para ejecutar aplicaciones Java. Como mínimo soportan compatibilidad para MIDP 1.0 y CLDC 1.0 (ver Sección 1.5). Además, los dispositivos que tienen la versión BlackBerry Device Software v4.0 o una versión superior son compatibles además con MIDP 2.0/CLDC1.1.

BlackBerry proporciona una API de Java que permite desarrollar aplicaciones con las siguientes características:

- ✓ Aplicaciones con interfaces de usuario personalizadas.
- ✓ Capacidad de almacenamiento y recuperación de datos en local en el dispositivo.
- ✓ Escucha y captura de eventos de la interfaz y del sistema.
- ✓ Comunicaciones seguras mediante HTTP y TCP.
- ✓ Cobertura de red y compatibilidad con el modo *roaming*.

Las aplicaciones para dispositivos BlackBerry deben tener un cierto equilibrio, por lo que es necesario considerar algunas cuestiones en el diseño: el tamaño de la pantalla es más pequeño, por lo que puede mostrar un número limitado de caracteres, tiene velocidades de procesador algo bajas, las redes inalámbricas tienen períodos de latencia más largos que las LAN estándar, tienen poca memoria, la batería no dura mucho y solo se puede mostrar una pantalla a la vez. Todas estas consideraciones afectan al diseño y se deben tener en cuenta no solo para dispositivos BlackBerry sino para cualquier otro dispositivo móvil.

Symbian

Los entornos de desarrollo para Symbian están pensados principalmente para el lenguaje de desarrollo C++, si bien siempre se puede programar en Java Micro Edition, ya que la mayoría de los dispositivos Symbian incluyen un gestor de aplicaciones para esta plataforma.

Si nos decidimos a programar en C++, deberemos de utilizar la SDK asociada a la serie a la que pertenece el dispositivo para el que vamos a programar, en este SDK se encontrarán todas las librerías que podemos utilizar para esta serie.

Si decidimos programar en J2ME, el CLDC en la versión que soporte la serie para la que vamos a desarrollar, que normalmente será la CLDC 1.1 y utilizar las MIDP 2.0 de la versión de la serie a la que pertenece nuestro dispositivo. Siempre es posible utilizar tanto CLDC como MIDP genéricos, pero en este caso no tendremos acceso al 100% de la funcionalidad que la serie de nuestro dispositivo nos ofrece.

Palm OS (webOS)

Los lenguajes de programación para la plataforma Palm han evolucionado desde el lenguaje C++, para los antiguos dispositivos monocromo, hasta la programación en HTML5 con JavaScript y CSS, para los nuevos sistemas webOS y Enyo. Enyo nos permite desarrollar aplicaciones no solo para dispositivos móviles, se puede programar para PC y tabletPC. La aplicación siempre se ejecutará dentro de un navegador, siendo compatible con la mayoría de ellos, desde Internet Explorer, Safary, FireFox, etc.

Enyo proporciona un potente modelo de encapsulación que permite construir y realizar funcionalidad. Estas piezas de encapsulación se denominan componentes. Por ejemplo resulta sencillo construir una marca entrada y una marca etiqueta para construir un componente `entradaEtiquetada`.

Veamos un sencillo ejemplo de programación con Enyo.

```
<!doctype html>
<html>
<head>
    <title>Enyo</title>
    <script src=”enyo-2.0b/enyo.js” type=”text/javascript”></script>
</head>
<body>
    <script type=”text/javascript”>
        new enyo.Control({content: “Hello From Enyo”}).write();
    </script>
</body>
</html>
```

En la marca de cabecera cargamos el archivo *enyo.js* que está en la carpeta *enyo-2.0.b*.

El código javascript del cuerpo genera un control que puede utilizarse como una marca HTML, puede tener clases, atributos, estilos, etc.

Por ejemplo el siguiente código:

```
new enyo.Control({content: “Hello From Enyo”, classes: “foo”,
    style: “color: red”, attributes: {tabIndex: 0}}).write();
```

genera la siguiente marca HTML en el navegador:

```
<div class="foo" style="color: red;" tabIndex="0">Hello From Enyo</div>
```

La potencia del *framework* viene cuando combinamos más de un control.

```
new enyo.Control({
    components: [
        {content: "Hello From Enyo"},
        {tag: "hr"}
    ]
}).write();
```

En el código anterior hemos definido dos controles, *content* y *tag* agrupados como un componente.

Este nuevo control encapsula dos controles, por esto se denomina componente. Realmente los controles son un tipo de componente. El control exterior es el responsable de gestionar sus controles internos, sus mensajes, sus referencias y ciclo de vida.

```
new enyo.Control({
    components: [
        {name: "hello", content: "Hello From Enyo", ontap: "helloTap"},
        {tag: "hr"}
    ],
    helloTap: function() {
        this.$.hello.addStyles("color: red");
    }
}).write();
```

El código anterior amplía el componente con eventos, hemos añadido el evento *ontap* que recoge tanto los clics del ratón como los toques de las pantallas táctiles e invoca al método *hellotap* que cambia el estilo del objeto a rojo, cuando se pulsa en él.

Por último, Enyo contiene un generador de constructores llamado *enyo.kind* que nos permite reutilizar el código, es decir, nuestros componentes, de manera muy sencilla.

```
enyo.kind({
    name: "Hello",
    kind: enyo.Control,
    components: [
        {name: "hello", content: "Hello From Enyo", ontap: "helloTap"},
        {tag: "hr"}
    ],
    helloTap: function() {
        this.$.hello.addStyles("color: red");
    }
});
```

```
// make two, they're small
new Hello().write();
new Hello().write();
```

El código anterior crea un control llamado *Hello* que posteriormente es instanciado dos veces, como si fuese una simple clase de JavaScript. A partir de ahora tendremos el control *Hello* que podremos utilizar dentro de nuestro código javascript sin preocuparnos de su implementación y sin tener que repetir código.

Objective-C

Objective-C es una extensión orientada a objetos de la sintaxis del lenguaje de programación C, utilizado en Mac OS X y GNUStep. Para desarrollar aplicaciones con este lenguaje, es recomendable saber programar en C, y si además sabes programar en C++, Java o C#, será más fácil. Es importante tener presente que en iPhone no existe recolector de basura, por lo que es recomendable estudiar en profundidad la documentación oficial de Apple sobre Objective-C y la gestión de memoria [2][4], [3][5], aunque en la versión Objective-C 2.0 se incluye un recolector de basura, este no es soportado por iPhone, por lo que cada instancia tiene un contador de referencias y cuando llega a cero el objeto es liberado de la memoria. Es decir, que los objetos de la API oficial de Apple a los que les pasemos objetos creados por nosotros, para almacenarlos en su interior, los retendrá aumentando el contador y los liberará cuando el objeto sea destruido.

Al estudiar Objective-C encontrarás que comparte elementos comunes con el lenguaje C como: sentencias de control de flujo (*if*, *for*, *while*, etc.); tipos de datos fundamentales; estructuras y punteros; conversiones implícitas y explícitas entre tipos; el ámbito de las variables (globales, estáticas o locales); las funciones y su sintaxis; las directivas del preprocesador (añadiendo Objective-C las suyas, así como las llamadas directivas del compilador).

Las clases en Objective-C se crean mediante dos archivos: un *.h* para definirla y un *.m* para implementarla. Por tanto, hay que utilizar *#import* para decirle al compilador las clases que vamos a utilizar en nuestro código. Se puede utilizar los mismos tipos que en C, más lo que añade el propio lenguaje.

En Objective-C se aplican los enlaces dinámicos donde las variables son identificadas por el runtime y no por el compilador y solo si existen punteros de objetos. De esta forma, comparando los lenguajes C y C++ con Objective-C veremos que los métodos y funciones deben ser llamados en alguna parte del código y se comprueba que efectivamente funcionen (en C++ por parte del compilador). En Objective-C el programa compila todos los datos y los objetos son los que deciden cuándo ejecutar los métodos.

Es un lenguaje interpretativo que permite crear clases, métodos y funciones, pero su sintaxis cambia al ser más dinámico. El lenguaje Objective-C es débilmente tipado, al contrario que C++, que es un lenguaje mayormente tipado y debe realizar conversiones de lenguaje para que sea más ordenado por defecto, pero al igual que el lenguaje C. Esto significa que una variable puede tener muchos valores distintos, por ejemplo, valores no específicos, cambiantes, que aunque estén relacionados no son los mismos.

[4] Consultar bibliografía en la página web del libro.

[5] Consultar bibliografía en la página web del libro.

Objective-C sobresale por su parte dinámica:

- **Memoria dinámica**. Objective-C crea sus objetos siempre en memoria dinámica para darles menos peso y favorecer la rapidez del proceso, contrario a lo que sucede con los lenguajes C y C++, los cuales deben adaptarse a la memoria dinámica y tienen un mayor peso.
- **Tipos dinámicos**. El tipado en Objective-C es mucho más amplio que en C++.
- **Introspección**. Se utiliza para preguntar acerca de los datos y su uso, por ejemplo, cuando se requiere información sobre a qué clase u objeto pertenece tal variable.
- **Enlace dinámico**. Objective-C busca en el propio objeto y clase la función de ejecución, a diferencia, por ejemplo, del lenguaje C++ donde en el compilador se decide qué funciones realizar para el objeto.
- **Carga dinámica**. Cuando se cargan las funciones del programa, el lenguaje busca nuevas formas de actualizarse y completar lo que le falta (por ejemplo, a través de *plugins*).

Desde el punto de vista de la conexión, en Objective-C se distinguen tres tipos: *(i)* conexión *outlet*, punteros de un objeto a otro; *(ii)* conexión *target action*, mensaje *action* hacia un *target* y *(iii) Bindings*, conexión de objetos distintos.

A continuación, un ejemplo de "`Hola mundo`" para iPhone:

Makefile (GNU make):

```
CC=arm-apple-darwin-cc
LD=$(cc)
LDFLAGS=-lobj –framework CoreFoundation –framework Foundation –framework UIKit –framework
LayerKit –framework CoreGraphics
all:   SampleApp
SampleApp: mainapp.o SampleApp.o
    $(LD) $(LDFLAGS) –v  -o $@ S^
%.o    %.m
       $(CC)  -c $(CFLAGS) $(CPPFLAGS) $< -o $@
clean:
       rm –f *.o SampleApp
```

mainapp.m

```
#import <UIKit/Uikit.h>
#import "SampleApp.h"
Int main(int argc, char **argv)
{
    NSAutoreleasePool *poll = [[NSAutoreleasePool alloc init];
    Return UIApplicationMain(argc, argv, [SampleApp Class]);
}
```

SampleApp.h

```
#import <CoreFoundation/CoreFoundation/h>
#import <Foundation/Foundation.h>
#import <UIkit/CDStructures.h>
#import <UIkit/UIWindow.h>
#import <UIkit/UIWView-Hierarchy.h>
#import <UIkit/UIHardware.h>
#import <UIkit/UIKit.h>
#import <UIkit/UIApplicacition.h>
#import <UIkit/UITextView.h>
#import <UIkit/UIView.h>
@interface SampleApp : UIApplication {
    UIView      *mainView;
    UITextview  *textView;
}
@end
```

SampleApp.m

```
#import "SampleApp.h"
@implementation SampleApp
        - (void) applicationDidFininshLaunching: (id) unused
    {
         UIWindow *window;
         structCGRect rect 0 [UIHardware
               fullScreenApplicationContentRect];
         rect.orgin.x = rect.origin.y = 0.0f;
         window = [[UIWindow alloc] iniWithContentRect: rect];
         mainview = [[UIView alloc] iniwithFrame: rect];
         textView = [[UITextView alloc]
             iniWithFrame: CGRectMake(0.0f, 0.0f, 320.0f, 480.0f)];
         [textView setEditable:YES];
     [textView setTextSize:14];
     [window orderFront: self];
     [window makeKey: self];
     [window_setHidden: NO];
     [window setContentView: mainView];
     [mainView addSubview:textView];
     [textView setText:@"Hello World"]
}
```

Podríamos concluir que Objective-C y el lenguaje C realizan procesos similares pero mejorados, y que en C++ hay mayores diferencias [4][6]. Además es un lenguaje muy limpio, pequeño y por ende, mucho más rápido y fácil de aprender que C++. Queda claro que la evolución de estos lenguajes ha permitido dejar de lado lo estático y pasar a lo dinámico.

Lenguajes .net (Windows Phone)

NET es una plataforma de desarrollo de aplicaciones que se ejecutan en un entorno aislado del S.O y que recibe el nombre de *runtime*. Entre las características que ofrece esta plataforma encontramos que: *(i)* es orientada a objetos, *(ii)* permite desarrollar aplicaciones en varios lenguajes, *(iii)* proporciona un modelo de programación consistente que permite el desarrollo de diferentes modelos de aplicaciones (por ejemplo: web, Windows, consolas, móviles, etc.) orientadas a diversos dispositivos hardware (PC, *tablet*, *pocket PC*, etc.), *(iv)* es compatible con aplicaciones desarrolladas con modelo COM, *(v)* permite integrar aplicaciones de otras plataformas y S.O, ya que implementa estándares como XML, WSDL, SOAP, entre otros [5][7].

El componente fundamental y que contiene todos los elementos necesarios para el desarrollo de aplicaciones es .NET Framework, que está formado por:

- **Entorno de ejecución CLR (***Common Language Runtime*, Tiempo de Ejecución en Lenguaje Común): encargado de administrar la memoria, la seguridad de la ejecución y la generación del código nativo a través del compilador **JIT** (*Just In Time*, Justo a tiempo) encargado de traducir a código nativo los componentes, dependiendo de la CPU en la que se encuentren.
- **Bibliotecas base BCL (***Base Class Library*, Biblioteca de Clase Base): encargada de proporcionar los componentes y clases necesarios para el desarrollo de aplicaciones en la plataforma. Está compuesta por Windows Forms, ASP.NET y Servicios Web XML y ADO.NET.

En cuanto a los lenguajes de desarrollo, estos siguen las especificaciones del estándar CLS (*Common Language Specification*, Especificación de Lenguaje Común), por lo que son compatibles entre sí, lo que hace que la elección del lenguaje se ajuste más al gusto del desarrollador que a motivos tecnológicos.

Existe una gran variedad de lenguajes de programación en la plataforma .NET, pero destacan dos de ellos:

- **VB.NET**. Visual Basic .NET es la evolución de Visual Basic 6.0, es el lenguaje natural para utilizar .NET pues su sintaxis es igual que la de su predecesor, solo que con pequeños cambios que le han permitido adaptarse a la plataforma.
- **C#**. Diseñado específicamente para la plataforma .NET. Combina la complejidad de C y la flexibilidad de Java. Su sintaxis resulta sencilla para los desarrolladores que están familiarizados con ambos lenguajes.

Debido a que cada lenguaje utiliza su propia sintaxis de implementación de las directivas del CLS, es necesario que los compiladores traduzcan su código fuente a un lenguaje entendible por el CLR. Ese lenguaje es el **MSIL** (*Microsoft Intermediate Language*, Lenguaje Intermedio de Microsoft). De esta forma, los componentes y aplicaciones resultantes de la compilación reciben el nombre de *assemblies* o ensamblados. La naturaleza de estos archivos puede

6 Consultar bibliografía en la página web del libro.

7 Consultar bibliografía en la página web del libro.

ser ejecutable (*.exe*) o bibliotecas de clases y componentes (*.dll*). Los *assemblies* no son ejecutables directamente, sino que se compilan al código nativo de la CPU en la que se ejecuta el **JIT** del **CLR**, para ejecutarse en el dominio de la aplicación. Pero para que el **CLR** entienda cómo se deben ejecutar esas aplicaciones, los compiladores de los lenguajes .NET incluyen los *metadata* o metadatos, los cuales proporcionan información sobre los objetos que conforman la aplicación o el componente que se halla generado.

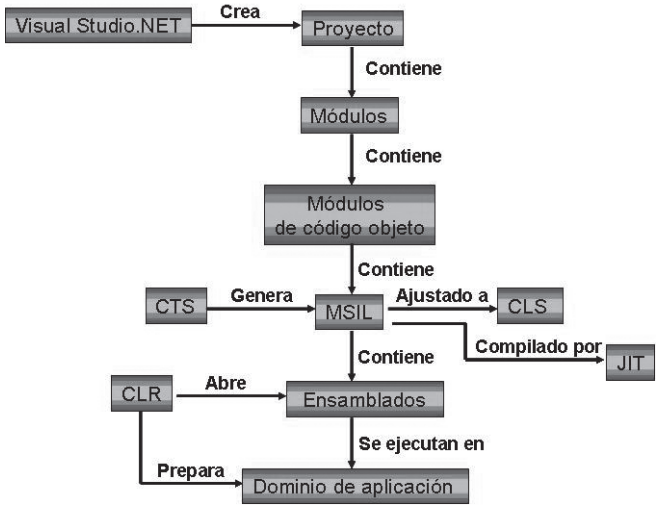

Figura 1.9. *Proceso de creación y ejecución de aplicaciones .NET*

La Figura 1.9 ilustra el proceso de creación y ejecución de aplicaciones con .NET.

Escribir la aplicación "Hola Mundo" en el entorno de desarrollo de aplicaciones para Windows Phone es tan sencillo como arrastrar un *textblock* hacia la zona del formulario central y hacer doble clic sobre el form. Así se cargará automáticamente el código del formulario y se generará el método para el manejo del evento **PhoneApplicationPage_Load**, que se ha de ejecutar una vez se cargue el formulario. En la sección de este evento escribiremos el código para que aparezca el texto "`Hola mundo`" en el **textblock** que previamente arrastramos al formulario.

```
Using System.Windows.Media.Animation;
Using System. Windows.Shapes;
Using Microsoft.Phone.Controls;
Namespace HolaWP7
{
        public parcial class Mainpage : PhoneApplicationPage
   {
        // Constructor
        Public MainPage ()
        {
             InitializeComponent();
               }
        Private void PhoneApplicationPage_Loaded(object
```

```
        sender, RoutedEventArgs e)
        {
              xtBlock1.Text = "Hola Mundo WP7";
        }
    }
}
```

Finalmente, para compilar y ejecutar la aplicación, basta con hacer clic sobre el botón **Play** o sobre la tecla **F5**.

1.4.2 ENTORNOS INTEGRADOS DE TRABAJO Y COMPILACIÓN

Una vez presentados los lenguajes para programación de aplicaciones móviles se pasará a presentar las herramientas de desarrollo y compilación para estos lenguajes.

Entorno para Blackberry

Para desarrollar aplicaciones Java para BlackBerry lo más recomendable es utilizar un entorno de desarrollo integrado. Se puede utilizar IDE generales como Eclipse y Netbean, configurando el entorno mediante el plugin de BlackBerry, o bien utilizando un IDE específico: BlackBerry proporciona JDE (*BlackBerry Java Development Enviroment*). El JDE es un entorno de desarrollo integrado que permite desarrollar y simular aplicaciones Java especialmente pensadas para teléfonos BlackBerry. Gracias a BlackBerry JDE, los desarrolladores pueden crear aplicaciones con el lenguaje de programación Java® ME y las API extendidas de Java para BlackBerry. El JDE está compuesto de los siguientes componentes:

- **BlackBerry Integrated Development Environment**: el entorno integrado con capacidad de edición de código, gestión de proyectos, depuración y traza de programas (Figura 1.10).
- **BlackBerry Smartphone Simulator**: un simulador de teléfonos BlackBerry. BlackBerry Smartphone Simulator ofrece un entorno tipo Windows completo y permite simular interfaces de usuario y la interacción del usuario, conexiones de red, servicios de correo electrónico y sincronización inalámbrica de datos.
- **Java ME y API de BlackBerry**: los paquetes de la especificación de Micro Edition de Java y el paquete que contiene la API específica de BlackBerry para sacarle todo el partido a los dispositivos BlackBerry.
- **Aplicaciones de ejemplo**: un conjunto de ejemplos de aplicaciones muy recomendables para iniciarse en el desarrollo de esta tecnología.

Además de lo anterior, el paquete JDE de BlackBerry incluye estas herramientas de soporte al desarrollo adicionales. Algunas de ellas permiten desarrollar sin necesidad de un IDE ya que son para su ejecución desde el símbolo del sistema y otras son para integrar en IDE de terceros:

- **RAPC**: es un compilador de línea de comando (a ejecutar desde el símbolo del sistema) para compilar archivos *.java* y *.jar* en archivos *.cod*. Los ficheros *.cod* se ejecutan en BlackBerry Smartphone Simulator o en un dispositivo BlackBerry.
- **JavaLoader**: esta herramienta permite agregar o actualizar una aplicación en un dispositivo BlackBerry. De esta forma se pueden probar los archivos *.cod* de la aplicación.

- **BlackBerry Signature Tool**: esta utilidad permite enviar solicitudes de firma de código a BlackBerry Signing Authority Tool.
- **Herramienta de verificación previa**: permite comprobar parcialmente las clases que se han desarrollado antes de cargar la aplicación en un dispositivo BlackBerry.
- **JDWP**: facilita la depuración de aplicaciones utilizando entornos de desarrollo integrados de terceros, como Eclipse o Netbeen.

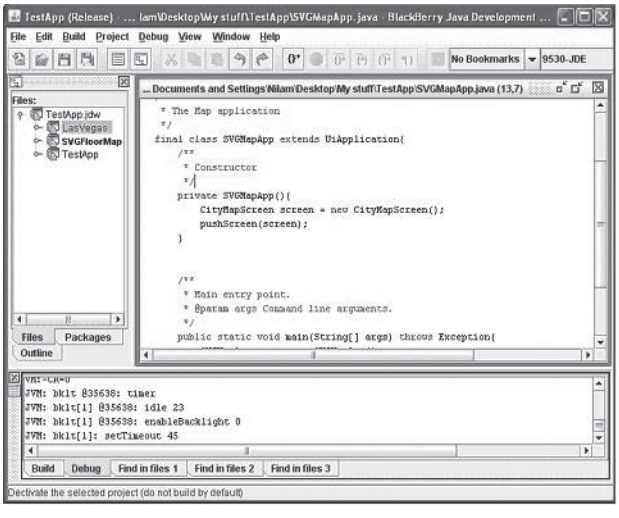

Figura 1.10. *El IDE de JDE de BlackBerry*
(fuente: http://svgopen.org/2009/papers/77Using_SVG_to_render_content_rich_maps_on_mobile_devices/)

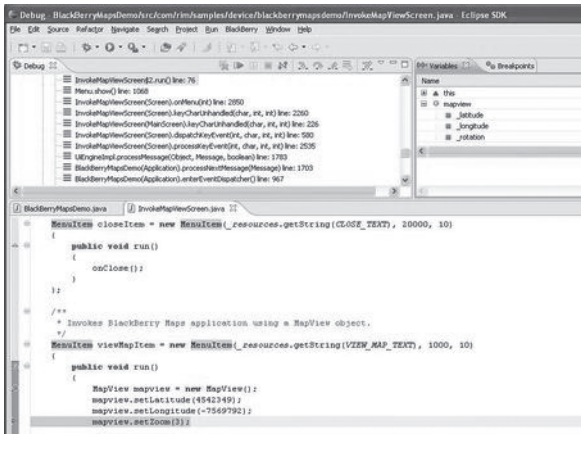

Figura 1.11. *Eclipse con el plugin de BlackBerry*
(fuente: http://www.visualbeta.es/12351/software/blackberry-lanza-su-plugin-para-eclipse-y-visual-studio/)

La Figura 1.11 muestra el Eclipse con el *plugin* de BlackBerry.

Entorno Para Android

El desarrollo de aplicaciones Android se realiza con un grupo de herramientas que son suministradas en el SDK. La utilización de este grupo de herramientas puede ser de dos formas: utilizando un Entorno de Desarrollo Integrado (IDE) en combinación con un plugin llamado ADT (*Android Development Tools*, Herramientas de Desarrollo para Android) o bien desde la línea de comandos. Se puede utilizar cualquier IDE, si bien lo más común es usar Eclipse. Si se decide prescindir de un IDE se necesita únicamente un editor de texto para escribir el código fuente e invocar las herramientas de compilación, depuración, etc., desde la línea de comandos o mediante *scripts*. Algunas de las herramientas más importantes de línea de comandos son:

- **Android**: crea y actualiza proyectos Android y crea y elimina AVD.
- **Android Emulator**: ejecuta aplicaciones en una plataforma Android emulada.
- **Android Debu Bridge**: es una interfaz para conectar la aplicación con un emulador o con un móvil Android. Permite instalar aplicaciones, ejecutar comandos en línea, etc.
- **Ant**: permite compilar y construir proyectos generando el fichero *.apk* instalable de la aplicación.
- **Keytool**: permite generar una clave privada que se usa para firmar y autenticar el fichero *.apk*. Esta herramienta forma parte del JDK.
- **Jarsigner**: es una herramienta para firmar el fichero *.apk* con una clave privada generada con la herramienta Keytool. Esta herramienta también forma parte del JDK.

La Figura 1.12 muestra los pasos básicos para desarrollar aplicaciones Android. Estos pasos básicos están agrupados en cuatro etapas:

- **Instalación**: en esta etapa se instala el entorno de desarrollo completo incluyendo el EDI y el SDK de Android, y se crean AVD (*Android Virtual Device*, Dispositivos Virtuales Android).
- **Desarrollo**: en esta etapa se crea y desarrolla el proyecto Android, creando el código fuente de la aplicación y añadiendo todos los ficheros fuentes que pueda necesitar como imágenes o demás recursos.
- **Depuración y pruebas**: en esta etapa, en primer lugar se genera un paquete de depuración *.apk* que contiene el proyecto desarrollado en la etapa anterior. Este paquete se puede instalar y arrancar en cualquier emulador o teléfono Android. Si se usa Eclipse, cada vez que se guarda el proyecto automáticamente se genera el *.apk* correspondiente. A continuación se depura la aplicación usando un depurador JDWP y las herramientas debug del SDK Android. Eclipse proporciona su propio depurador. Por último, se comprueba el correcto funcionamiento de la aplicación usando varias herramientas del SDK como emuladores y herramientas de traza.
- **Publicación**: en esta última etapa se configura y se construye la aplicación para generarse una versión *release* (una versión de entrega) para distribuir entre los usuarios.

Como ya se ha comentado anteriormente se puede utilizar cualquier IDE para desarrollar con el SDK Android. El más extendido es Eclipse. Veamos con detalle cómo instalar y configurar Eclipse para desarrollar aplicaciones Android.

1 Instalar Eclipse

El entorno Eclipse se puede descargar desde el sitio web de descargas de Eclipse[8]. La versión Eclipse *IDE for Java Developers* es una buena opción. La instalación consiste en descomprimir el fichero *.zip* descargado. Esto crea una carpeta llamada *eclipse* dentro de la cual hay, entre otros, el programa Eclipse, el cual inicia el entorno.

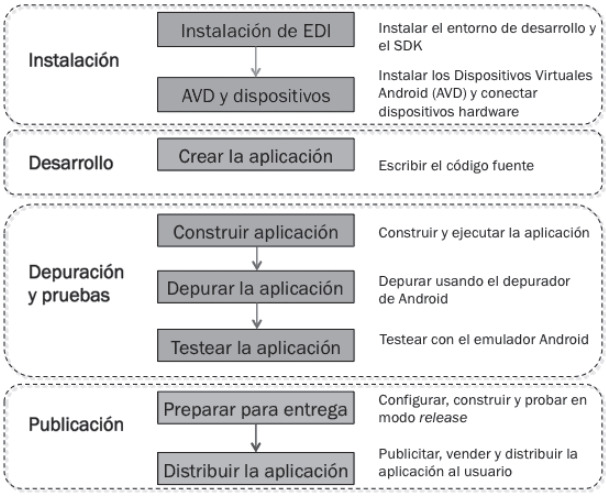

Figura 1.12. *Pasos básicos para desarrollar aplicaciones Android*

2 Instalar el SDK Android

El paquete SDK se puede descargar en un fichero *.zip* o en una versión de instalador (solo para Windows). Si se descarga un fichero *.zip*, la instalación consiste únicamente en descomprimirlo (se descomprime automáticamente en el directorio *android-sdk-<machine-platform>*, donde *<machine-platform>* es el nombre de la plataforma donde se instala). Si se descarga la versión de instalador para Windows únicamente se debe ejecutar el programa y seguir las indicaciones del mismo.

3 Instalar el Plugin ADT para Eclipse

Para instalar el plugin ADT se deben realizar las siguientes acciones:

3.1 Iniciar Eclipse y seleccionar del menú **Help** la opción de instalación de nuevo software (menú **Help > Install New Software...**).

3.2 Hacer clic en el botón **Add...** y aparecerá un cuadro de diálogo en el que se solicita introducir el nombre del repositorio donde está el *plugin* (introducir **ADT Plugin**) y la URL de localización (introducir *https://dl-ssl.google.com/android/eclipse/*). La Figura 1.13 muestra este cuadro de diálogo.

[8] *http://www.eclipse.org/downloads/*

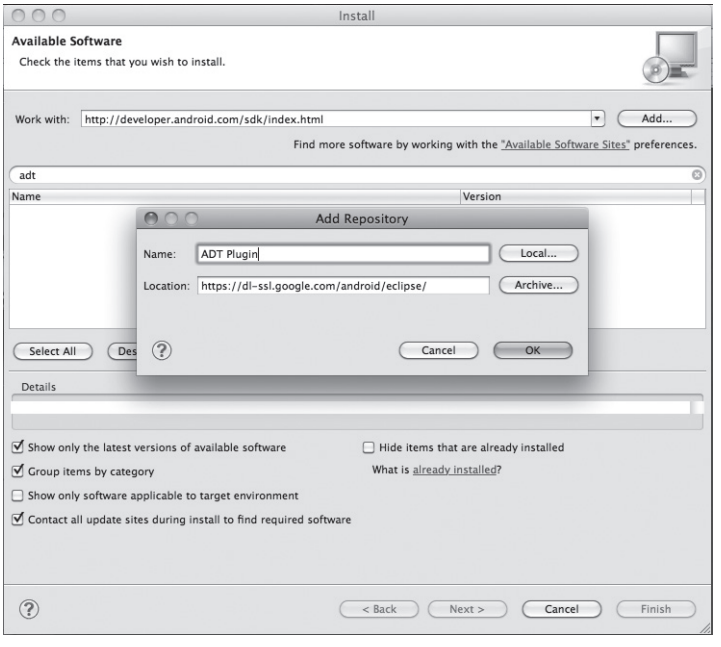

Figura 1.13. *Añadiendo el plugin ADT en Eclipse*

3.3 El paso anterior origina que en el cuadro de diálogo aparezca una lista de software instalable de ese repositorio. En concreto solo aparece un paquete. Se selecciona el único que aparece haciendo clic en la correspondiente *check box* (**Developer Tools**) y con el botón **Next** se pasa al siguiente cuadro de diálogo.

3.4 En la nueva ventana se muestra una lista de herramientas que van a ser descargadas. Hacer clic en el botón **Next** y a continuación aceptar los acuerdos de licencia. Por último hacer clic en el botón **Finish** finalizando el proceso de instalación. Una vez que la instalación ha sido completada es necesario restaurar Eclipse para que la nueva instalación tenga efecto.

Si ya se tiene instalado el *plugin* ADT conviene hacer actualizaciones cada cierto tiempo de las versiones más recientes. Para ello se utiliza el gestor de actualizaciones de Eclipse (*Update Manger*, accesible mediante el menú **Help > Check for updates**).

4 Configurar el plugin ADT

En este paso se configuran algunas preferencias del plugin. En concreto se debe asociar el plugin con el SDK de Android. Para ello seleccionar el menú **Eclipse > Preferences** y en el panel de la izquierda seleccionar **Android**. A continuación, en el panel principal se podrá indicar mediante el botón **Browse** la localización del SDK que fue descargado en su momento. Por último, hacer clic en **Apply** y **OK**.

5 Añadir Componentes

El SDK de Android está organizado en varias partes: versiones de plataformas Android, herramientas, ejemplos y documentación, las cuales se pueden instalar separadamente. El paquete SDK instalado que ha sido descargado incluye solamente las herramientas. Para desarrollar una aplicación Android necesitamos descargar e instalar la plataforma Android. Este proceso se realiza mediante el gestor AVD (*Android Virtual Device*, Dispositivo Virtual

Android). El AVD se puede ejecutar de varias formas. Una de ellas es desde Eclipse con el menú **Window > Android SDK Manager**. Este menú proporciona la ventana del gestor de AVD y muestra todos los componentes del SDK que se puede descargar e instalarse en el SDK local de la máquina en la que se trabaja. ¿Qué componentes son los que se deben descargar? Los componentes básicos son *SDK Tools, SDK Platform Tools* y *SDK platform*. La Figura 1.14 muestra el gestor AVD. Como se puede ver se muestran todos los paquetes asociados al SDK que se pueden descargar, indicando cuáles están ya instalados.

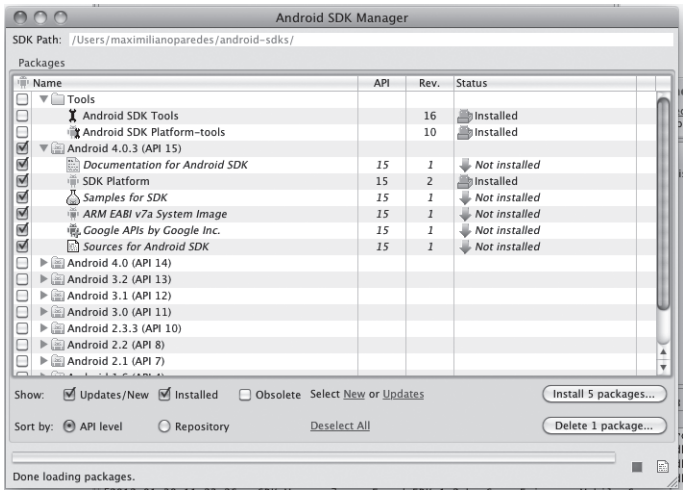

Figura 1.14. *AVD Manager*

Con este último paso ya queda configurado el entorno de desarrollo listo para programar a falta de crear dispositivos emuladores (ver apartado 1.4.3).

Entorno para Symbian

Este sistema puede programarse con gran variedad de entornos de desarrollo dependiendo de la tecnología de programación que queramos utilizar. Las principales tecnologías son Java Micro Edition y C++, para la primera es necesario disponer del SDK para la serie sobre la que queramos programar y luego utilizarla junto con un entorno de desarrollo como puede ser Eclipse o NetBeans, al ser J2ME una tecnología muy extendida es posible construir gran cantidad de aplicaciones sin los SDK, pero si queremos utilizar el cien por cien de la funcionalidad de los dispositivos finalmente tendremos que utilizarlo. Si bien instalando, por ejemplo, NetBeans con soporte para J2ME podremos crear aplicaciones que correrán directamente en todos los dispositivos Symbian.

A pesar de la facilidad de desarrollo de aplicaciones J2ME, para desarrollar aplicaciones realmente potentes en la plataforma Symbian hay que utilizar la tecnología basada en C++, para ello tendremos que localizar el SDK para C++ de la serie para la que vamos a construir nuestra aplicación. Para C++ existe un entorno basado en Eclipse y en el SDK Qt llamado Carbide c++, desarrollado expresamente por Nokia para el desarrollo de aplicaciones de todas las series. También existe un plugin para Visual Studio C++ de Carbide c++ con el que es posible desarrollar aplicaciones C++.

Una de las maneras más sencillas de programar aplicaciones para la serie S60 sería utilizando PythonForSerie60 que directamente nos permite crear las aplicaciones dentro de nuestro propio teléfono en el lenguaje Python 2.5.

Entorno para Palm OS (WebOs)

Para los antiguos entornos de desarrollo de Palm se requería el SDK y los compiladores cruzados gcc dentro de la plataforma Linux o Cygwin que es Linux emulado en Windows.

Para el desarrollo de sistemas en webOS se utiliza el entorno Ares, que es el primer entorno de desarrollo para móviles que funciona íntegramente dentro de un navegador. Para utilizarlo simplemente hay que registrarse en la web *http://ares.palm.com* y posteriormente introducir el usuario y clave con los que nos hemos registrado, no es necesario instalar nada, desde cualquier ordenador que disponga de navegador y conexión a Internet se puede hacer uso del entorno de desarrollo. La filosofía de este entorno al igual que la del sistema webOS es que el futuro de la tecnología móvil, y del resto de tecnologías se va a construir dentro de la web. De esta manera, resulta muy sencillo desarrollar en la web, depurar en un dispositivo concreto conectado a la web y lanzar la aplicación desarrollada en el catálogo de aplicaciones de webOS en la web.

Este entorno tiene dos vistas principales, una vista de gestión de ficheros, como se muestra en la Figura 1.15 y una vista de diseño visual de pantallas y controles, tal y como muestra la Figura 1.16.

El sistema webOS también nos proporciona un SDK y un *plugin* PDk que permite desarrollar en C++ o en JavaScript aplicaciones para móviles. Puede instalarse en cualquier plataforma Windows, Linux o iOS y se integra dentro del IDE de Eclipse.

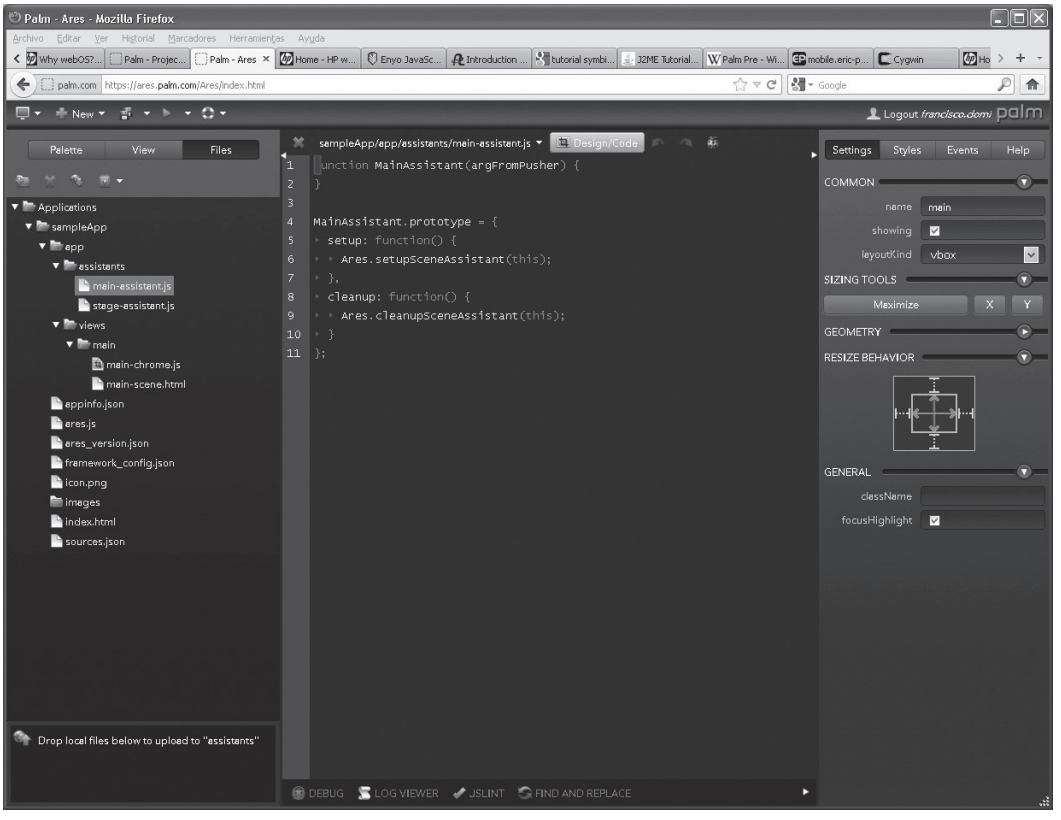

Figura 1.15. *Entorno Ares en vista gestión de ficheros*

Entorno para Iphone

El desarrollo de aplicaciones para iPhone se realizó en un principio a través de páginas o proyectos web, sin embargo, Apple vio la necesidad de controlar y unificar la programación de las aplicaciones para sus diferentes dispositivos. Es así como en el año 2008 decidió crear el SDK (*Software Development Kit*, *Kit* de Desarrollo de Software) de Apple, un conjunto de herramientas y tecnologías de desarrollo para crear aplicaciones para iPhone e iPod Touch.

Para obtener el SDK es necesario inscribirse en el *iOS Dev Center* (el sitio oficial de desarrolladores de Apple)[9], desde donde se puede descargar gratuitamente. Si una vez creada la aplicación se quiere comercializar, es necesario enviarla al App Store, lo cual requiere registrarse en el programa de desarrolladores de Apple y pagar una cuota de $99. El desarrollador fija el precio de la aplicación y obtiene el 70% de las ventas realizadas a través del App Store. Un dispositivo Apple solo ejecuta aplicaciones firmadas por Apple, por lo que después de asignado el permiso de desarrollador, Apple expide un certificado que permite hacer pruebas en nuestro iPhone.

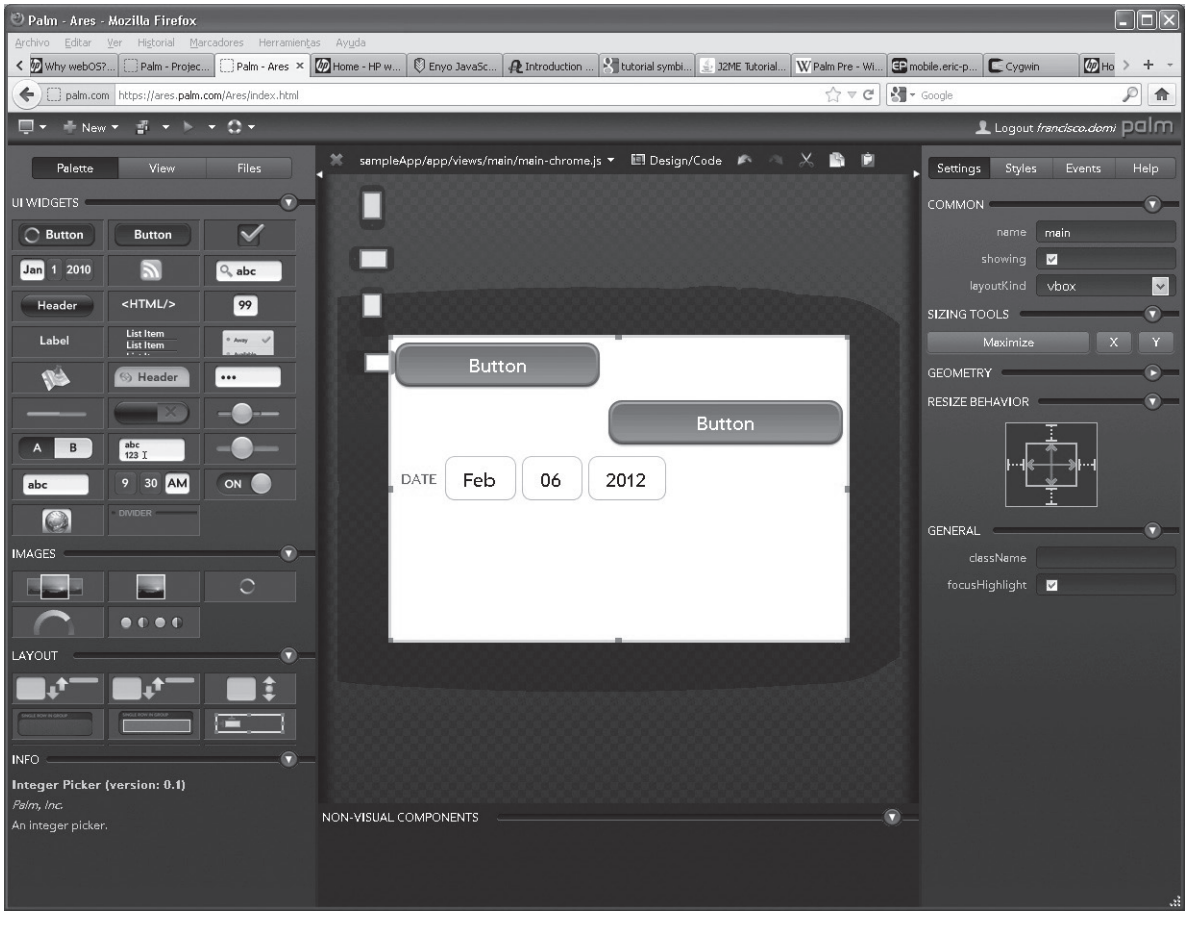

Figura 1.16. *Entorno Ares con vista de controles y diseño*

[9] iOS Dev Center: *https://developer.apple.com/devcenter/ios/index.action*

Hay dos aspectos clave a la hora de desarrollar una aplicación: *(i)* las características del dispositivo para el cual se va a desarrollar la aplicación (tamaño de la pantalla, memoria, etc.) pues no es lo mismo programar para un iPhone que para un Mac y *(ii)* el tipo de aplicación, iOS permite tres tipos de aplicaciones:

- **Aplicación web**: se basa en contenido web, se ejecutan en un navegador y no hacen uso de las funcionalidades del dispositivo.
- **Aplicación nativa**: este tipo de aplicación utiliza las funcionalidades del dispositivo y se instala en el mismo, de modo que para acceder a ella no hace falta conexión/acceso a Internet.
- **Aplicación híbrida**: es posible desarrollar aplicaciones combinando los dos tipos anteriores, de modo que pueda instalarse en el dispositivo y además requiera de un navegador para utilizarla.

Además de los tipos de aplicaciones, también podemos hablar del estilo, por ejemplo: *(i)* aplicaciones productivas, en las que lo que prima es la utilidad que pueden ofrecer al usuario en la realización de una determinada tarea; *(ii)* aplicaciones de utilidad, aportan información al usuario sin que tenga que realizar muchas acciones para obtenerla, por ejemplo, una aplicación que informe sobre el estado del tiempo; *(iii)* aplicaciones de pantalla completa, útiles sobre todo en el soporte de juegos, música y vídeo.

Herramientas de desarrollo

El SDK incluye las herramientas necesarias para que la aplicación pueda incluirse en el App Store.

- **Herramientas Xcode**. Utilizadas para la codificación, desarrollo y depuración. Es un IDE (*Integrated Development Environmet*, Entorno Integrado de Desarrollo) que permite escribir, compilar, ejecutar y depurar el código de la aplicación que se está desarrollando. También es útil para organizar los archivos por proyectos y facilitar su importación/exportación (ver Figura 1.17).

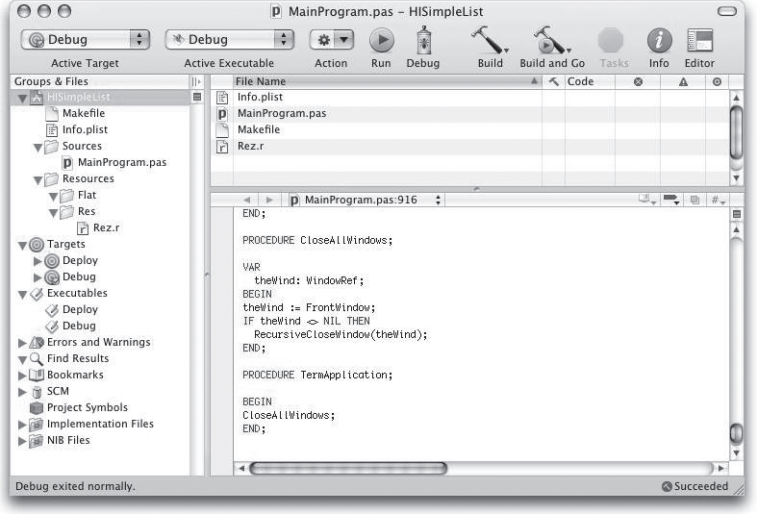

Figura 1.17. *Herramientas Xcode*

También proporciona un simulador para probar la aplicación antes de ejecutarla en el dispositivo. Es posible desarrollar una aplicación a partir de un patrón establecido, ya que Xcode proporciona una serie de plantillas para tal fin.

- **Interfaz Builder**. Permite diseñar la interfaz gráfica de la aplicación. Es básicamente una herramienta gráfica que trabaja en conjunto con Xcode. Desde la versión 4 de Xcode se puede programar el código y diseñar la interfaz de la aplicación en la misma ventana, simplemente arrastrando y soltando elementos como controles o componentes. Los archivos generados con Builder tienen extensión *.nib* y describen la componente gráfica creada. El diseño se puede cargar mediante Cocoa Touch (ver Figura 1.18).

Figura 1.18. *Herramienta Builder*

- **Instruments**. Esta herramienta permite analizar el comportamiento de la aplicación mediante el análisis de parámetros como: *(i)* memoria consumida por la aplicación, *(ii)* ancho de banda consumido, *(iii)* recursos utilizados y energía requerida, *(iv)* datos acerca de las animaciones o actividad relacionada con archivos y *sockets* (ver Figura 1.19).

Dentro de las herramientas de desarrollo también encontramos el emulador, pero nos referiremos a él en otra sección dedicada a emuladores para iPhone.

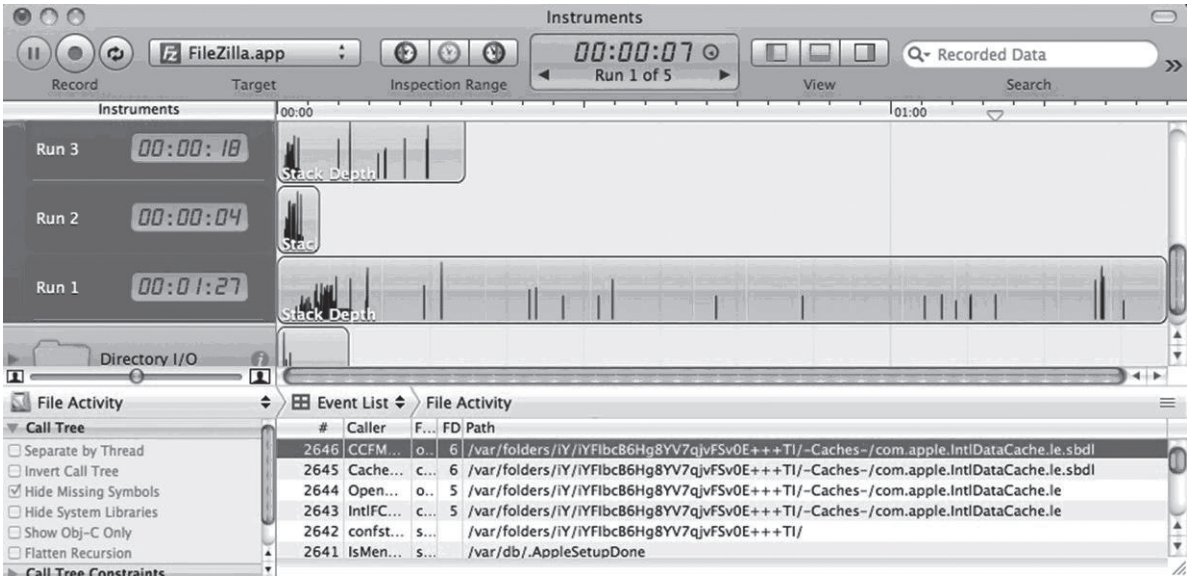

Figura 1.19. *Herramienta Instruments*

En resumen, los pasos para el desarrollo de aplicaciones para iPhone son los siguientes:

1 Inscribirse en el iOS Dev Center y descarga del SDK.

2 Registrarse como desarrollador de Apple (este paso es necesario solo para desarrolladores que quieran comercializar la aplicación desarrollada).

3 Elegir el tipo de aplicación a desarrollar.

4 Utilizar las herramientas de desarrollo para el diseño y puesta a punto de la aplicación.

5 Probar la aplicación, utilizando el simulador de iOS y luego sobre el dispositivo de Apple.

6 Distribuir la aplicación a través del App Store (una vez obtenida la firma de Apple).

Entorno para Windows Phone

Para desarrollar aplicaciones para teléfonos con S.O Windows Phone, lo primero que necesitamos es descargar el SDK que se encuentra disponible en la web oficial del App HUB: *http://create.msdn.com/en-us/home/getting_started* y que contiene las siguientes herramientas:

- **Visual Studio 2010 para Windows Phone**. Es una de las herramientas que utilizaremos en el proceso de desarrollo de las aplicaciones. Esta versión es gratuita y completamente funcional. Desde la ventana principal podremos definir dos tipos de proyectos diferentes, Silverlight para Windows Phone 7.5 o XNA 4.0 para Windows Phone 7.5, en ambos casos utilizando el lenguaje C#, seleccionando **New Project** o desde el menú **File > New > Project**, tal como se muestra en la Figura 1.20.

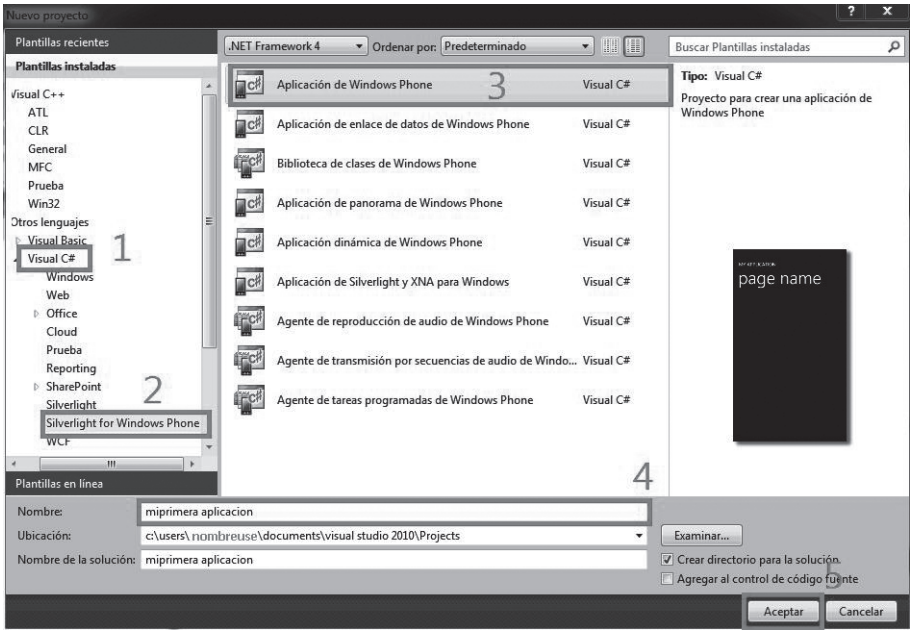

Figura 1.20. *Creación de un proyecto en Visual Studio 2010 (fuente: http://www.carlosluengo.es/2011/11/)*

- **Microsoft Expression Blend 4 para Windows Phone**. Orientada al diseño de interfaces de usuario, esta herramienta trabaja en clave con Visual Studio 2010. Como desarrolladores tendremos control sobre el aspecto de la aplicación, pudiendo definir animaciones, transiciones o personalizando las plantillas de control, los estilos o plantillas de datos y también, podremos enlazar los datos visualmente. Los proyectos que tenemos en Visual Studio 2010 son accesibles desde Expression Blend 4, por lo que es posible trabajar sobre un mismo proyecto en los dos entornos. Así por ejemplo, al abrir un archivo XAML, CS o VB podremos seleccionar, a través del menú contextual, la opción **Edit in Visual Studio** para editar el archivo seleccionado en Visual Studio y después de realizar los cambios y guardar, podremos volver a Expression Blend 4 donde los cambios se cargarán automáticamente (ver Figura 1.21).

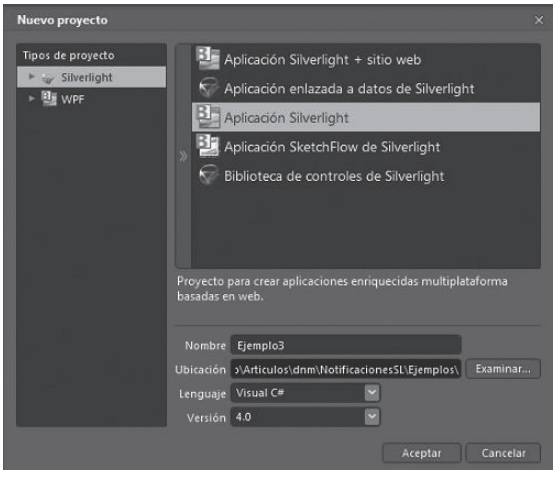

Figura 1.21. *Selección de un proyecto en Expression Blend 4 (fuente: http://www.dnmplus.net/articulos/notificaciones-en-silverlight.aspx)*

- **Application Development Tool**. Esta herramienta permite mostrar aplicaciones en formato XAP al emulador o a un dispositivo desbloqueado para desarrollo, con lo cual, podremos probar aplicaciones fuera de Visual Studio. Podemos hacer lo mismo desde Visual Studio, haciendo clic sobre el proyecto y seleccionando la opción **Deploy** (Desplegar), con lo cual se instalará la aplicación en el dispositivo seleccionado en la barra de herramientas (ver la Figura 1.22).

Figura 1.22. *Herramienta de despliegue de aplicaciones*

- **Developer Registration Tool**. Con esta herramienta podremos registrar un dispositivo físico Windows Phone con nuestra cuenta de desarrollador de Marketplace. De esta manera, el dispositivo quedará desbloqueado para desarrollar, depurar y desplegar aplicaciones en él (ver Figura 1.23).

Figura 1.23. *Herramienta de registro de dispositivos*

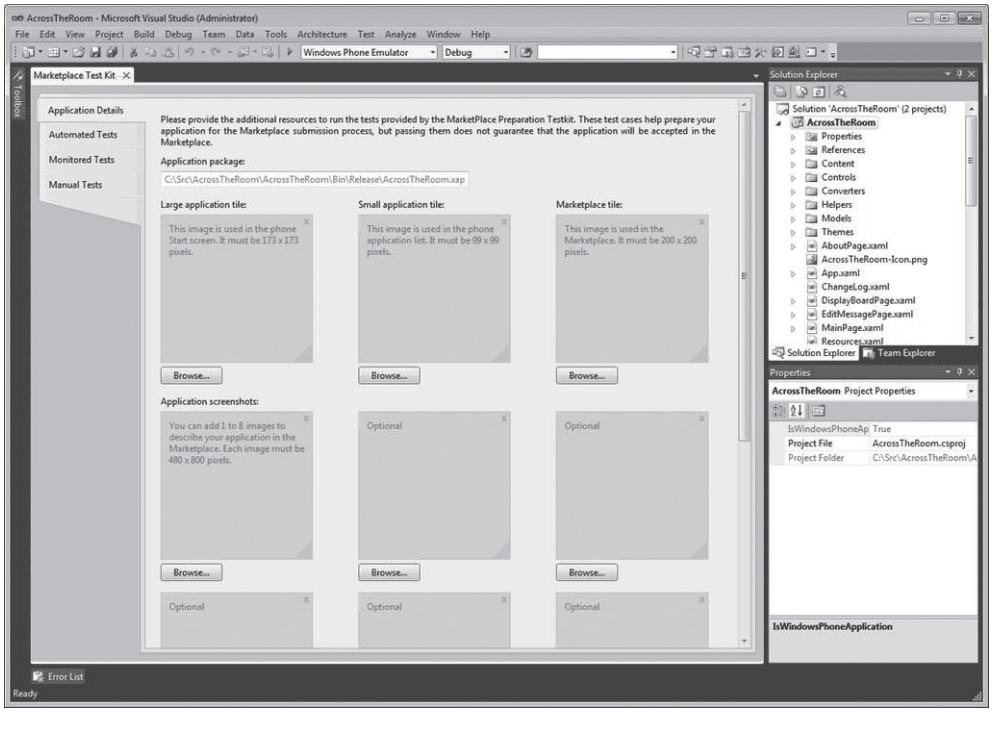

Figura 1.24. *Marketplace Test Tool*

- **Windows Phone Marketplace Test Tool**. Esta herramienta permite realizar pruebas automáticas y manuales de la aplicación. Está integrada en Visual Studio 2010, basta con hacer clic con el botón derecho sobre el nombre del proyecto y escoger la opción **Open Market Test Kit** (ver Figura 1.24).

Las opciones de esta herramienta son:

- **Application Details**: carga la iconografía y capturas que vamos a enviar al Marketplace.
- **Automated Test**: ejecuta pruebas automáticas para comprobar el funcionamiento de la aplicación. Es necesario compilar la aplicación en modo *Realise*.
- **Monitored Test**: permite hacer pruebas teniendo en cuenta parámetros específicos como tiempo de arranque, consumo de memoria, etc.
- **Manual Test**: proporciona una lista de comprobación para realizar 50 pruebas de forma manual sobre la aplicación, marcando aquellas en las que la aplicación ha fallado.

De esta forma nos aseguramos de que la aplicación que vamos a mandar al Marketplace funciona correctamente.

1.4.3 EMULADORES

Los entornos de desarrollo presentados en el apartado 1.4.2, utilizan emuladores para la simulación de las aplicaciones que desarrollan sin necesidad de utilizar un dispositivo móvil real. Estos emuladores suelen ir integrados en los entornos de desarrollo, si bien se pueden instalar y utilizar en algunos casos independientemente.

Emuladores para Android

El SDK Android incluye un emulador de dispositivos móviles virtuales. El emulador puede invocar a otras aplicaciones, acceder a red, reproducir audio y vídeo, almacenar y recuperar datos, etc., usando servicios de la plataforma Android. Además, también proporciona servicios de depuración y permite realizar prototipos de aplicaciones, desarrollar y testear aplicaciones Android sin necesidad de un dispositivo físico. El emulador proporciona una ventana en la que se visualiza en ejecución la aplicación que se está desarrollando junto con otras aplicaciones Android. El emulador utiliza configuraciones AVD (*Android Virtual Device*, Dispositivo Virtual para Android). Un AVD permite definir ciertas características hardware del teléfono a emular. Se pueden crear varias configuraciones para diferentes plataformas Android mediante los AVD.

Un AVD está formado por:

- ✓ **Perfil hardware**: define las características hardware de dispositivos virtuales. Por ejemplo, se puede definir si el dispositivo tiene cámara, si usa teclado físico QWERTY, cuánta memoria tiene, etc.
- ✓ **Mapping del sistema**: se puede definir para qué versión de la plataforma Android se ejecutará el emulador.
- ✓ **Otras opciones**: especifica otras características del emulador, como la dimensión de la pantalla, la apariencia, si se quiere emular una tarjeta de almacenamiento SD, etc.
- ✓ **Área de almacenamiento**: es necesario un espacio de almacenamiento de la máquina de desarrollo para almacenar los datos del usuario del emulador (como aplicaciones que instala el usuario en el emulador) y la tarjeta SD emulada.

Lo más común es integrar y utilizar el emulador desde el EDI que se esté utilizando para desarrollar aplicaciones Android. Eclipse proporciona con el menú **Window > AVD Manager** un gestor de dispositivos virtuales. Este gestor muestra una lista de dispositivos virtuales que están instalados en el entorno de desarrollo y diferentes opciones para gestionarlos (añadir nuevos, editarlo, borrarlo, etc.) (ver Figura 1.25). Cuando se instala Eclipse y el SDK de Android, no se crean automáticamente los dispositivos virtuales. Esto se tiene que hacer explícitamente, mediante el botón **New**, el cual muestra un cuadro de diálogo que permite configurar el emulador (ver Figura 1.26).

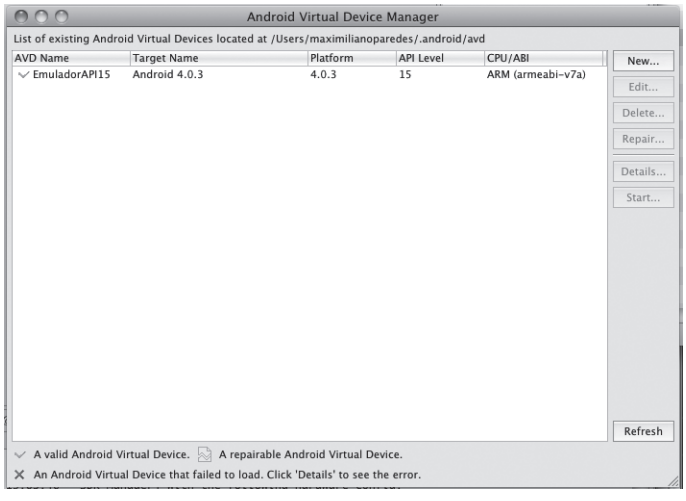

Figura 1.25. *Gestor de dispositivos virtuales*

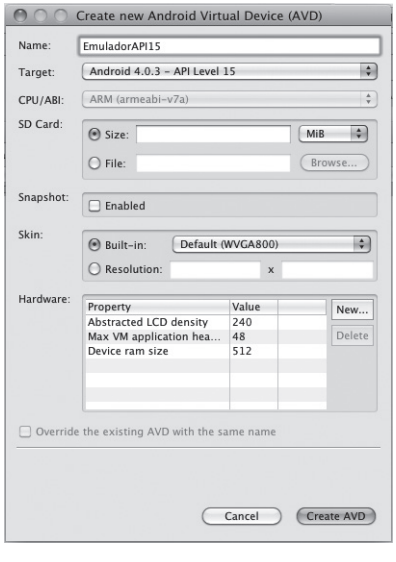

Figura 1.26. *Crear un nuevo emulador Android*

Para completar la instalación y configuración del entorno de desarrollo Eclipse es necesario crear un dispositivo virtual. Como ya se ha comentado esto se realiza mediante el botón **New** del menú **Window > AVD Manager**. En el nuevo cuadro de diálogo que aparece es necesario indicar un nombre y seleccionar el *target* (la API de la plataforma Android en la que se desarrolla) (ver Figura 1.26). Cuando se crea un dispositivo virtual o emulador el desarrollador puede también configurar algunos aspectos básicos de hardware (como LCD, tamaño de la máquina virtual en el emulador, tamaño de la RAM, etc.) y además puede añadir más características hardware al dispositivo virtual y configurarlas mediante el botón **New** como son pantalla sensible al tacto, teclado, GPS, batería, etc. Es importante seleccionar el *target* apropiado ya que si se está desarrollando una aplicación con una cierta API, esta aplicación no se podrá ejecutar en un dispositivo virtual que tenga un *target* con una versión de API inferior. La Figura 1.27 muestra la ejecución de un emulador.

Figura 1.27. *Apariencia de un emulador Android (fuente: http://developer.android.com/guide/developing/devices/emulator.html)*

El emulador de la Figura 1.27 es el proporcionado por el SDK de Android. Sin embargo, hay otros emuladores que se pueden descargar independientemente y ejecutarlos, instalando aplicaciones en los mismos. En estos casos las capacidades de depuración del emulador pueden mermar o ser inexistentes. Algunos de estos emuladores son Google Android Emulator (ver Figura 1.28). En general este tipo de emuladores no tienen demasiada presencia en el mundo de los desarrolladores.

Figura 1.28. *Emulador Google Android (fuente: http://www.addictivetips.com/windows-tips/download-google-android-emulator/)*

Emulador para Blackberry

Hay varios simuladores BlackBerry disponibles para emular la funcionalidad de varios productos como dispositivos BlackBerry y BlackBerry Enterprise Server. Los simuladores BlackBerry Enterprise Server simulan ciertos servicios (como MDS services y *email*) del servidor de BlackBerry (*Enterprise Service*), que pueden ser usados en combinación con un simulador de dispositivos BlackBerry (simular contenidos web). Dentro de los simuladores para dispositivos BlackBerry se pueden encontrar dos principales emuladores: BlackBerry PlayBook Simulators, para desarrollos de aplicaciones para dispositivos PalyBook, y BlackBerry Smartphone Simulators, para dispositivos móviles *smartphone*. Estos simuladores permiten testear el software, pantallas, teclados, etc., con los que las aplicaciones que se desarrollan trabajarán en dispositivos reales. Los simuladores también simulan el entorno de varias condiciones de redes *wireless*. El BlackBerry Smarthpone Simulators puede ser descargado e instalado como un programa *.exe* en una máquina Windows, si bien, y es lo más común, suele estar integrado en el JDE de BlackBerry, con lo cual al descargar el entorno de desarrollo ya se está descargando también el simulador. La Figura 1.29 muestra una ventana de un emulador BlackBerry para *smartphone*.

Figura 1.29. *Emulador BlackBerry (fuente: http://techtastico.com/post/simulador-blackberry/blackberry-device-simulator/)*

Emulador para Symbian

Cada uno de los entornos de desarrollo suele llevar incorporado su propio emulador o utiliza los que proporcionan los SDK de las diversas series de Symbian. Aunque algunos lenguajes como pyS60, que es Python para la serie S60, no necesitan emulador ya que este lenguaje puede ser ejecutado en cualquier plataforma, y como comentamos anteriormente puede programarse incluso directamente en el propio dispositivo, sin necesidad de realizar compilaciones cruzadas.

Los emuladores permiten seleccionar las características del modelo a emular, de esta forma podemos ver como se adapta, sobre todo la interfaz gráfica a las diversas resoluciones y tamaños de pantalla de los diferentes dispositivos. A continuación, en la Figura 1.30 se muestra un ejemplo de diversos modelos de dispositivos para emuladores en la plataforma de desarrollo Netbeans para J2ME.

Figura 1.30. *Diversos dispositivos emulados en J2ME*

Como podemos observar en el tamaño de pantalla, vemos que en todos hay diferentes elementos gráficos del mismo juego, como la interacción, pero el último emula un dispositivo táctil, son diferentes. De hecho una de las tareas más laboriosas del desarrollo de aplicaciones para móviles es verificar su correcto funcionamiento con el máximo número posible de configuraciones de dispositivos móviles.

Emulador para Palm Os (WebOS)

Los antiguos emuladores para Palm eran monocromo, sin pantalla táctil y con muy baja resolución, los proporcionaba la empresa Palm como herramientas independientes y se disponía de una versión emulador y una versión simulador. La primera emulaba la aplicación que queríamos depurar mientras que la segunda simulaba un dispositivo completo con todas las aplicaciones nativas que llevaba instaladas.

La Figura 1.31 muestra el resultado de una emulación para una Palm antigua.

Figura 1.31. *Emulación en Palm antigua*

Los emuladores actuales para la plataforma webOS van incluidos dentro del entorno de desarrollo SDK, permiten emular diversas configuraciones y resoluciones de pantalla. Se pueden ejecutar desde el propio entorno de desarrollo Ares pero es necesario tener instalado en local el SDK.

La Figura 1.32 muestra las cuatro configuraciones de pantalla posibles en el entorno webOS.

Figura 1.32. *Emulación en webOS*

Emulador para IPhone

El emulador iOS permite comprobar el correcto funcionamiento de la aplicación. Ofrece funcionalidades como: *(i)* rotación, cuando gira el dispositivo la aplicación rota con él; *(ii)* cambio de dispositivo, permite comprobar el funcionamiento de la aplicación tanto en un iPhone como en un iPad; *(iii)* comprobar el funcionamiento de la aplicación para diferentes versiones del iOS; *(iv)* avisos de memoria, permite comprobar el funcionamiento de la aplicación dependiendo del consumo de recursos (ver Figura 1.33).

Figura 1.33. *Simulador en modo iPhone e iPad*

Windows Phone

El kit de desarrollo de Windows Phone incluye un emulador que nos permitirá simular las aplicaciones en condiciones similares a las que encontraremos al ejecutarlas en nuestro dispositivo.

El emulador se ejecuta desde **Inicio > Programas > Windows Phone Developers Tools > Emulador de Windows Phone**. También lo podemos lanzar desde Visual Studio 2010. El emulador nos permitirá también probar aplicaciones que utilicen sensores (acelerómetro) o que dependan de la localización (ver las Figuras 1.34 y 1.35).

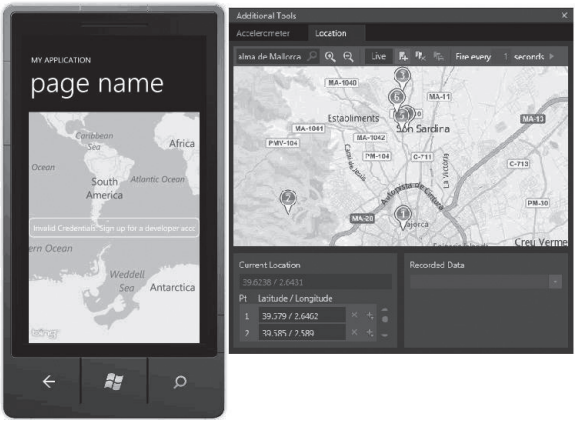

Figura 1.34. *Emulador de GPS*

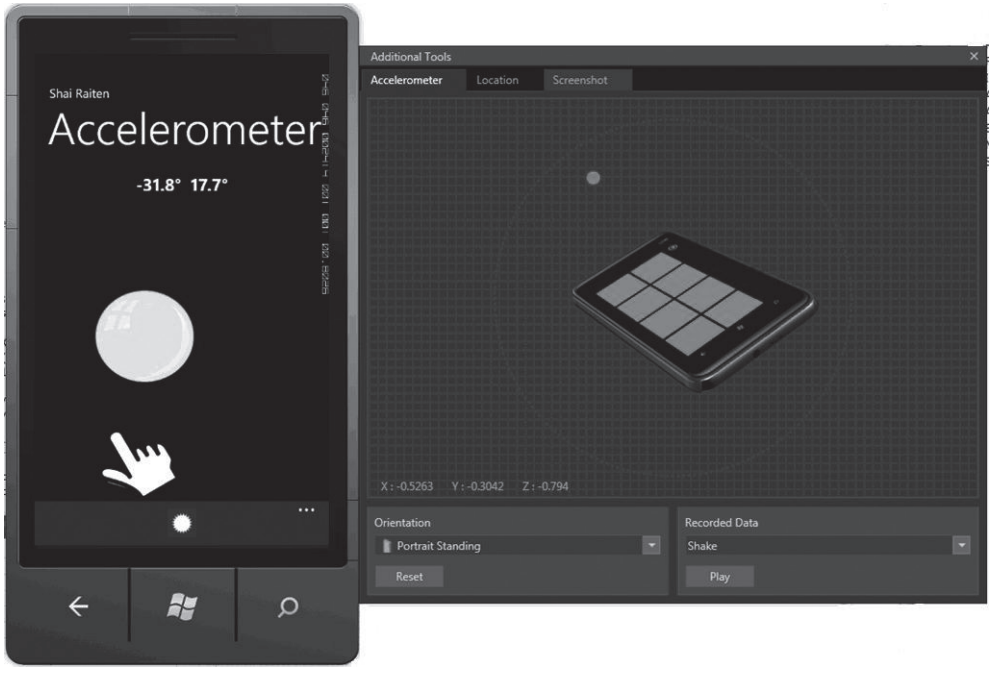

Figura 1.35. *Emulador del comportamiento de un acelerómetro (fuente: http://blogs.microsoft.co.il/blogs/shair/archive/2012/01/28/accelerometer-sensor-for-windows-phone-7.aspx)*

1.5 ARQUITECTURA J2ME

Como ya se ha comentado en los apartados anteriores, J2ME o Java 2 Micro Edition es la versión Java para dispositivos móviles y sistemas de pequeño tamaño. J2ME pertenece a la versión 2 de Java que está compuesta por tres ediciones distintas:

- ✓ **Edición Estándar** (J2SE): este es el entorno básico de Java, está compuesto por las clases básicas de Java, que nos permiten desarrollar y ejecutar aplicaciones cliente, servidor y *applets*.
- ✓ **Edición Empresarial** (J2EE): es una ampliación de la edición anterior y está pensada para el desarrollo de aplicaciones servidor.
- ✓ **Edición Micro** (J2ME): es una versión reducida de Java que permite desarrollar aplicaciones para sistemas móviles, empotrados y electrónicos de características especiales. Las aplicaciones en esta edición se denominan MIDlets por analogía con los *applets*.

La Figura 1.36 muestra la relación entre las tres ediciones de Java 2.

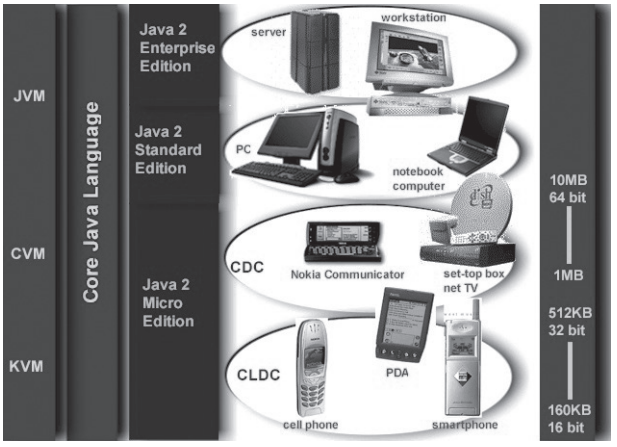

Figura 1.36. *Entorno Java 2*

J2ME se divide en tres partes:

- **Configuración**: contiene la máquina virtual, normalmente es una versión reducida de la máquina normal de Java, junto con algunas librerías básicas. A la máquina virtual de la configuración se la suele llamar KVM, ya que requiere unos pocos Kilobytes para ejecutar el *bytecode*.
- **Perfil**: está desarrollado sobre una configuración y proporciona un conjunto de librerías útiles para poder desarrollar aplicaciones completas con ellas.
- **Paquetes opcionales**: son paquetes especiales que no pueden ser ejecutados en todos los dispositivos que permiten obtener un acceso a todas las funcionalidades que nos aportan los dispositivos, por ejemplo, un paquete opcional es la API para acceso al módulo GPS, otro paquete opcional es una API para programación con gráficos 3D. No todos los dispositivos permiten utilizar GPS o gráficos 3D.

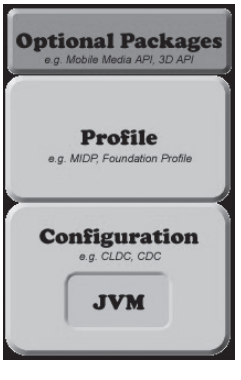

Figura 1.37. *Partes de J2ME*

1.5.1 CONFIGURACIONES Y PERFILES

Podemos ver en la Figura 1.37 que Java presenta dos configuraciones CLDC y CDC. La primera es la utilizada en dispositivos móviles mientras que la segunda está pensada para dispositivos con más recursos como PDA o *tablets*. Dentro de estas configuraciones existen versiones.

Los dispositivos donde se ejecuten las aplicaciones con configuración CLDC deben cumplir las siguientes restricciones:

- ✓ Disponer de entre 128 KB y 512 KB de memoria ROM o *flash* donde se almacena la propia máquina virtual y de 32 KB de memoria volátil para guardar información en tiempo de ejecución.
- ✓ Procesador de 16 ó 32 bits con al menos 25 MHz de velocidad.
- ✓ Bajo consumo, debido a que suelen estar alimentados por batería.
- ✓ Tener algún tipo de conectividad como Wi-Fi, 3G, etc.

Sobre las configuraciones se definen perfiles, en el caso de la configuración CLDC, se definen los perfiles para dispositivos de información móvil o MIDP. Están definidos sobre las siguientes restricciones mínimas:

- ✓ Pantalla de tamaño 96 × 54, con un solo bit de color y una forma de píxel cuadrada.
- ✓ Entrada por teclado o pantalla táctil.
- ✓ Memoria, 256 KB para el CLDC, 8 KB para guardar datos y 128 KB de memoria volátil para la ejecución de la aplicación.
- ✓ Sonido, para reproducir tonos musicales.
- ✓ Conectividad, bidireccional e inalámbrica, con un ancho de banda limitado.

Los perfiles MIDP nos permiten definir el ciclo de vida de los MIDlets, construir interfaces de usuario, soporte para la red, juegos, sonido, criptografía, etc.

1.5.2 MODELO DE ESTADOS

Los MIDlets se ejecutan dentro de un entorno proporcionado por el Gestor de Aplicaciones o AMS (*Application Management Software*), este es un programa que además de guardar las aplicaciones desarrolladas para J2ME contiene la implementación de la KVM, CLDC y el MIDP. Todos los dispositivos que permiten J2ME tienen un Gestor de Aplicaciones, que suele venir como una aplicación independiente.

El Gestor de Aplicaciones no puede ejecutar los MDlet directamente, estos deben venir empaquetados en archivos *.jar* que contienen uno o varios MIDlets así como un archivo *.jad* que describe cada uno de ellos, indicando nombre, recursos que necesita, fabricante, fecha de creación, etc.

El AMS define el ciclo de vida de los MIDlets, que pasa por 5 fases: *localización*, *instalación*, *ejecución*, *actualización* y *borrado*.

El MIDP define los tres estados en los que se puede encontrar un MIDlet cuando está en ejecución: *Activo*, *Pausa* y *Destruido*.

1.5.3 CICLO DE VIDA DE UNA APLICACIÓN

A continuación vamos a mostrar paso a paso el desarrollo de la primera aplicación para móviles en el entorno J2ME con el IDE NetBeans 7.1.

Vamos a la opción de menú **File > New Project** y seleccionamos **Java ME** como muestra la Figura 1.38.

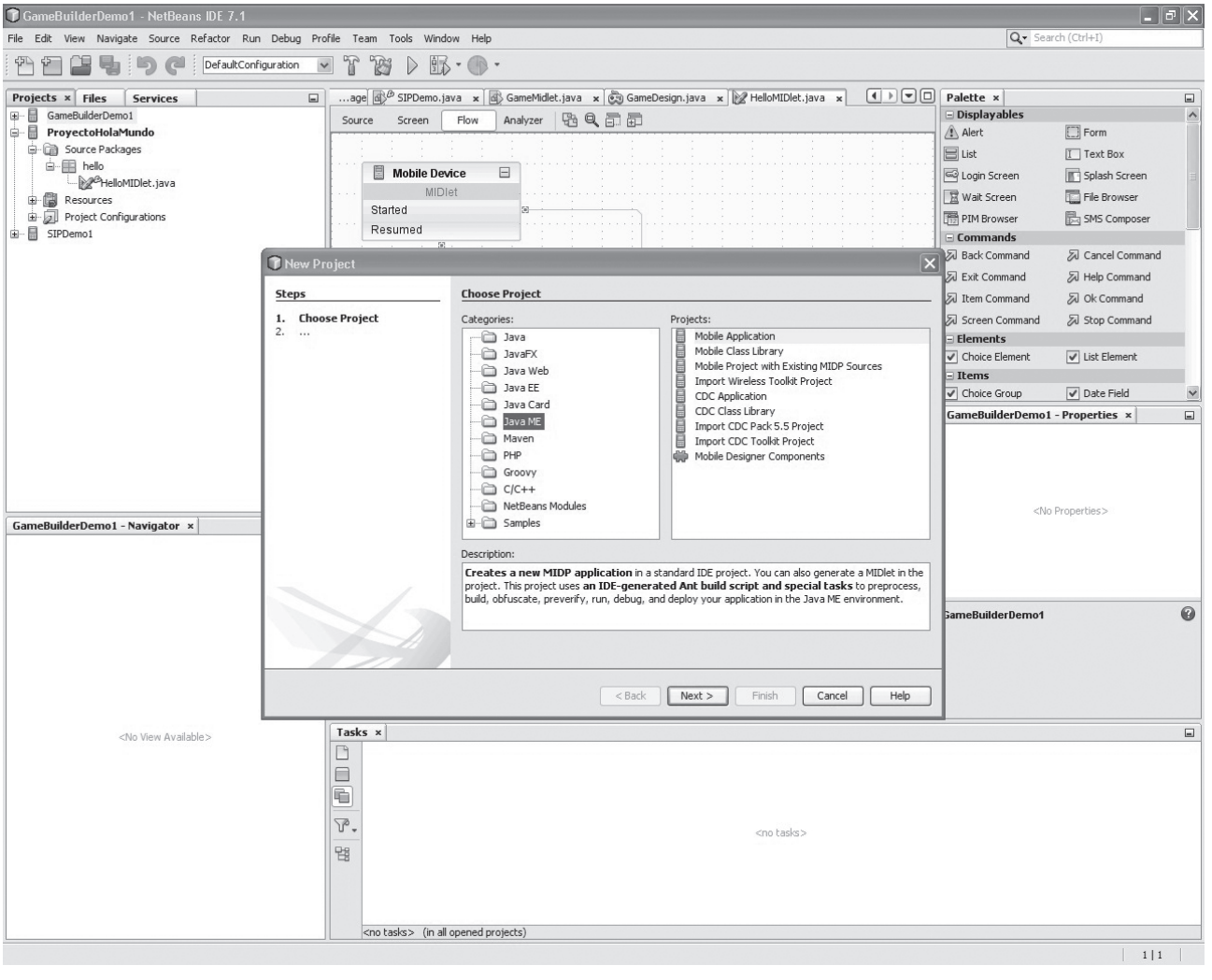

Figura 1.38. *Elección de proyecto Java ME*

Hacemos clic sobre el botón **Next** y a continuación nos sale la siguiente ventana de diálogo donde vamos a indicarle el nombre del proyecto en el campo de texto **Project Name**, ahí pondremos **ProyectoHolaMundo** y haremos clic en el botón **Next** de esta ventana de diálogo.

La siguiente ventana llamada **Default Platform Selection** la dejamos tal cual, y hacemos clic en el botón **Next** de nuevo. Pasamos a la pantalla **More Configuration Selection**, donde tampoco cambiamos nada y hacemos clic en el botón **Finish**. A partir de este momento el entorno creará un proyecto funcional pero vacío y con nuestra aplicación como aplicación principal, como muestra la Figura 1.39.

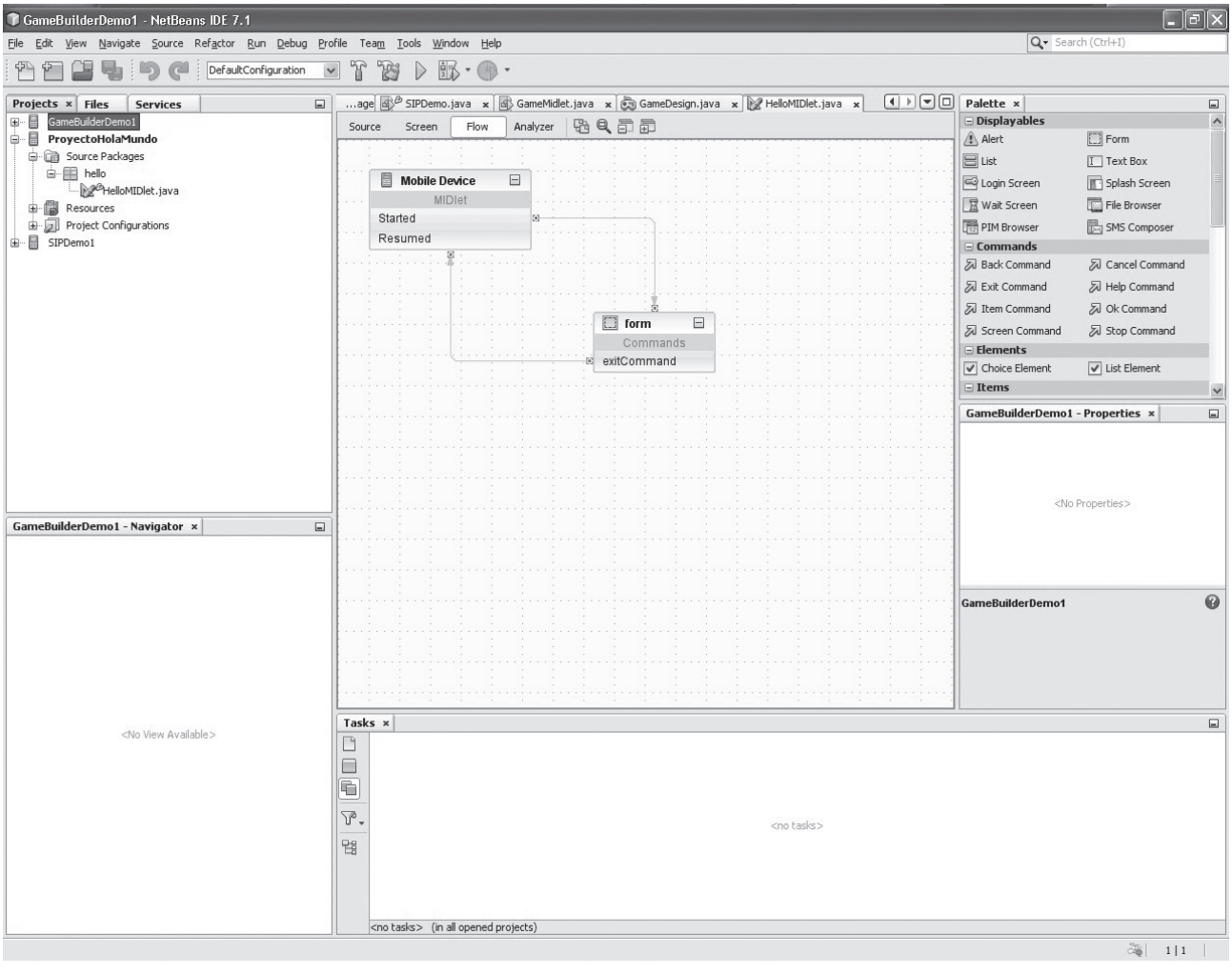

Figura 1.39. *Entorno Java ME vista flujos*

Nos aparece seleccionado el editor de flujos de ejecución con la etiqueta **Flow** resaltada (véase la Figura 1.39). Este editor no lo vamos a utilizar en este ejemplo. Vamos a pulsar sobre la etiqueta **Screen** y nos aparecerá un editor que nos permite cambiar elementos visuales del dispositivo, vamos a cambiar el texto **Hello** por **Hola** y **Helo World!** por **Hola Mundo**. Para ello nos vamos a los campos **Label** y **Text** del diálogo de propiedades que es el que hay abajo a la derecha llamado **Properties**. En la Figura 1.40 donde ya han sido realizadas las sustituciones.

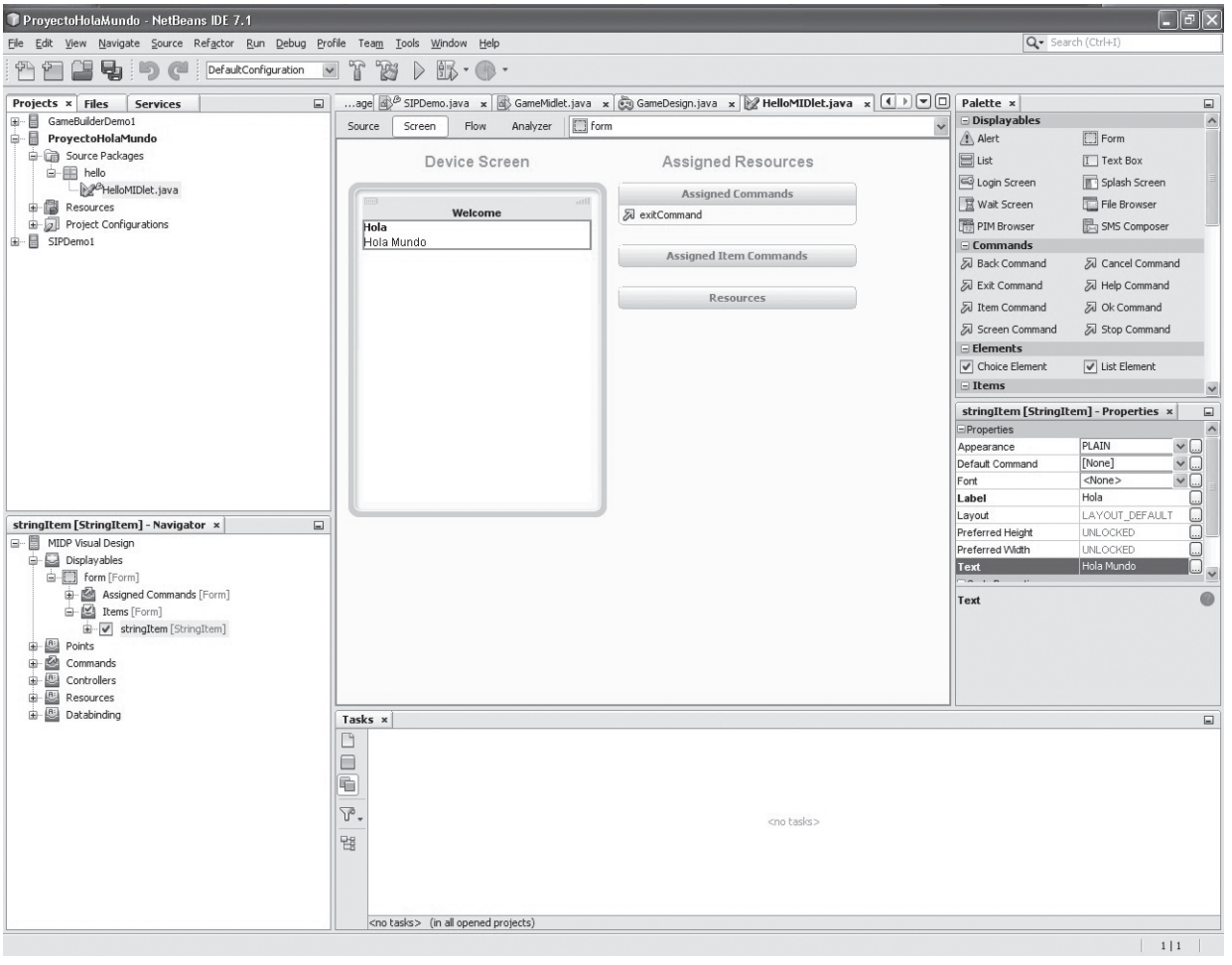

Figura 1.40. *Entorno Java ME vista diseño*

Ahora solo falta ejecutar nuestro primer proyecto, para ello solo hay que pulsar la tecla **F6** o ir a la opción de menú **Run > Run Main Project** o hacer clic en el icono verde de **Play** que hay en la barra de herramientas en el centro superior de la pantalla. En cualquiera de estos casos comenzará a ejecutarse el emulador y nos mostrará nuestra primera aplicación para móviles. Todo ello sin teclear una sola línea de código Java (ver Figura 1.41).

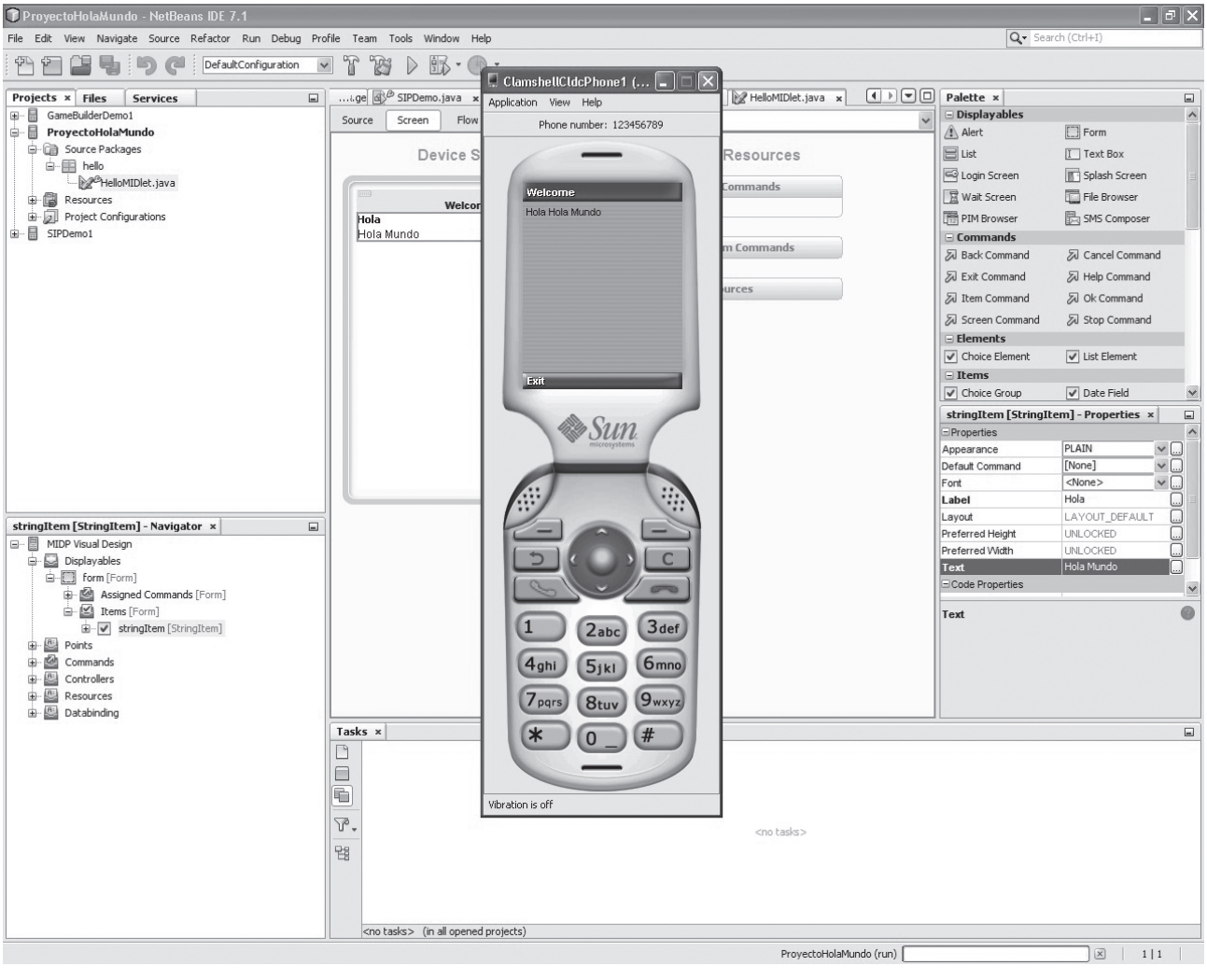

Figura 1.41. *Ejecución del proyecto*

Por último, si dispone de un teléfono móvil con gestor de aplicaciones para J2ME puede pulsar sobre el proyecto con el botón de la derecha del ratón, seleccionar la opción **Batch Delploy...** y se generará un archivo llamado **ProyectoHolaMundo.jar** que puede enviar a su móvil y se instalará automáticamente con el gestor de aplicación y podrá ejecutarlo cuando desee.

ACTIVIDADES 1.1

- Suponga que está instalado el *plugin* ADT en un entorno Eclipse. ¿Cómo se puede actualizar la última versión de este *plugin* desde Eclipse? Indique el camino de menú a realizar.
- Si se tiene la instalación por defecto del SDK de Android bajo Eclipse y se quisiere descargar e instalar el componente *Documentation for Android SDK* de dicho paquete, ¿qué utilidad se debe usar? Indique el camino de menú de Eclipse para arrancar dicha utilidad.
- Realice una revisión de las características de los últimos modelos de teléfonos móviles con S.O Android, iPhone y Windows Phone. ¿Qué funcionalidad añadiría en cada caso, distinta a las ya existentes?

RESUMEN DEL CAPÍTULO

En este capítulo se ha proporcionado una visión de los principales hitos que han enmarcado la historia y evolución de los dispositivos móviles. También se ha presentado un resumen de las principales limitaciones a las que se enfrentan los desarrolladores, así como también los beneficios e inconvenientes de los principales enfoques que puede utilizar, según las necesidades de la aplicación a desarrollar. Además, se han descrito las tecnologías disponibles en la actualidad, los lenguajes de desarrollo, los entornos específicos y las características de los sistemas de simulación asociados a cada tecnología. Finalmente se ha descrito la arquitectura J2ME y sus principales componentes.

EJERCICIOS PROPUESTOS

- **1.** Descargar desde *http://www.eclipse.org/downloads/* y realizar la instalación de Eclipse IDE for Java Developers en el disco duro del ordenador.
- **2.** Instalación del SDK Android. Se quiere instalar el paquete SDK de Android desde el formato comprimido *.zip*, para ello se debe conectar a *http://developer.android.com/sdk/index.html* y realizar la instalación por defecto en el disco duro de la máquina.
- **3.** Se desea instalar la última versión del *plugin* ADT para Eclipse desde el propio IDE Eclipse. El nombre que se quiere dar al repositorio que se añade es "ADT Plugin".
- **4.** Cree un dispositivo virtual Android con AVD Manager con las siguientes características para un *target* Android 4.0.3: 2 GB de tarjeta SD Card y un tamaño de RAM de 1.024 MB.
- **5.** Descargue e instale *Java Development Enviroment* de BlackBerry en el disco duro del PC.
- **6.** Descargue el SDK de Windows Phone y cree la aplicación "Hola Mundo". Verifique su ejecución utilizando el emulador.
- **7.** Descargue el SDK de iPhone y cree la aplicación "Hola Mundo". Verifique su ejecución, utilizando el emulador.

TEST DE CONOCIMIENTOS

1 ¿Qué es la Computación Ubicua y cuáles son sus pilares fundamentales?
- **a)** Descentralización, diversificación, conectividad y simplicidad.
- **b)** Rapidez, sencillez y disponibilidad.
- **c)** Descentralización, diversificación y rapidez.
- **d)** Las respuestas *b* y *c* son correctas.

2 ¿Cuál es el cambio más relevante en las baterías de los teléfonos móviles?
- **a)** Livianas, sin materiales tóxicos, control de la temperatura.
- **b)** Mayor duración, menor tiempo de carga.
- **c)** Más económicas.
- **d)** Las respuestas *a* y *c* son correctas.

3 ¿Cuáles son las limitaciones para el desarrollo de aplicaciones en teléfonos móviles?
- **a)** Latencia.
- **b)** Deficiencia en las conexiones.
- **c)** Tamaño de las pantallas.
- **d)** Las respuestas anteriores son correctas.

4 ¿Cuáles son los enfoques principales para el desarrollo de aplicaciones para dispositivos móviles?
- **a)** Clientes nativos, clientes JME.
- **b)** Clientes basados en la web, clientes basados en Midleware.
- **c)** Clientes nativos, clientes JME, clientes basados en la web y clientes basados en Midleware.
- **d)** Clientes nativos y clientes Midleware.

5 ¿Cuáles son los principales componentes del modelo software de Windows Phone?
- **a)** Ejecución, modelo aplicación, modelo IU, Kernel, hardware.
- **b)** Ejecución, modelo aplicación, modelo IU, integración nube, Kernel, hardware.
- **c)** Modelo aplicación, modelo IU, integración nube.
- **d)** Ejecución, modelo IU, hardware.

6 ¿Cuáles son los componentes de la arquitectura del sistema operativo iOS?
- **a)** Cocoa Touch, Core Services.
- **b)** Cocoa Touch, Media, Core Services, Core OS.
- **c)** Media, Core Services, Core OS.
- **d)** Applications, Applications framework, Libraries, Kernel.

7 ¿Cuál es la principal característica del lenguaje Objective-C?
- **a)** Su facilidad de uso.
- **b)** Sus numerosas librerías.
- **c)** Su parte dinámica.
- **d)** Su facilidad para integrarse con otros lenguajes.

8 ¿Por qué es necesario que los compiladores en .NET traduzcan su código fuente a uno entendible por el CLR y cómo se llama ese lenguaje?
- **a)** Porque se ejecutan en la CPU del dispositivo.
- **b)** Porque necesitan generar información sobre la aplicación.
- **c)** Porque cada lenguaje utiliza su propia sintaxis de implementación de las directivas CLS.
- **d)** Porque hace más rápida la ejecución de la aplicación.

9 ¿Qué función realizan los *assemblies*?
- **a)** Proporcionan información sobre los componentes de la aplicación.
- **b)** Compilan al código nativo de la CPU en la que se ejecuta el JIT del CLR para ejecutarse en el dominio de la aplicación.
- **c)** Definen las directivas de implementación de los lenguajes.
- **d)** Las respuestas *a* y *c* son correctas.

10 ¿De dónde se obtiene la información de los objetos que conforman una aplicación desarrollada en la plataforma .NET?
- **a)** De los *assemblies*.
- **b)** Del JIT.
- **c)** Del CLR.
- **d)** De los metadata.

11 ¿Qué es un AVD de Android?
- **a)** Un gestor de *plugins* Android.
- **b)** Un dispositivo real Android conectado por USB.
- **c)** Un dispositivo virtual Android.
- **d)** El paquete de instalación del SDK Android.

12 ¿Cuál es la principal ventaja de BlackBerry?
- **a)** El sistema operativo es código abierto y esto permite que haya muchas aplicaciones.
- **b)** El sistema de gestión de memoria.
- **c)** Hay muy pocos modelos en el mercado y esto hace que sea muy simple manejarlo.
- **d)** Su capacidad para enviar y recibir correos electrónicos por Internet.

13 Señale la afirmación correcta sobre el sistema operativo Android:
- **a)** La máquina virtual de Android (*Dalvik Virtual Machine*) forma parte del Entorno de Ejecución.
- **b)** La capa de Aplicaciones proporciona un conjunto de servicios y sistemas para el desarrollador.
- **c)** La capa de Librerías proporciona un conjunto de aplicaciones de usuario como correo electrónico, calendario y mapas.
- **d)** La máquina virtual de Android (*Dalvik Virtual Machine*) puede ejecutar cualquier código Java directamente.