Capítulo 6

JAVA 2 MICRO EDITION (J2ME)

6.1 Introducción

La empresa Sun Microsystems [30] lanzó a mediados de los años 90 el lenguaje de programación Java que, aunque en un principio fue diseñado para generar aplicaciones que controlaran electrodomésticos como lavadoras, frigoríficos, etc. debido a su gran robustez e independencia de la plataforma donde se ejecutase el código, desde sus comienzos se utilizó para la creación de componentes interactivos integrados en páginas web y programación de aplicaciones independientes.

Estos componentes se denominaron applets y casi todo el trabajo de los programadores se dedicó al desarrollo de estos. Con los años, Java ha progresado enormemente en varios ámbitos como servicios HTTP, servidores de aplicaciones, acceso a bases de datos (JDBC)... Java se ha ido adaptando a las necesidades tanto de los usuarios como de las empresas ofreciendo soluciones y servicios tanto a unos como a otros.

Debido a la explosión tecnológica de estos últimos años Java ha desarrollado soluciones personalizadas para cada ámbito tecnológico. Sun ha agrupado cada uno de esos ámbitos en una edición distinta de su lenguaje Java. Estas ediciones son:

- *Java 2 Stantard Edition* (J2SE): orientada al desarrollo de aplicaciones independientes y de applets. Esta edición de Java es la que en cierta forma recoge la iniciativa original del lenguaje Java. Tiene las siguientes características:
    - Inspirado inicialmente en C++, pero con componentes de alto nivel, como soporte nativo de strings y recolector de basura.
    - Código independiente de la plataforma, precompilado a bytecodes intermedios y ejecutado en el cliente por una JVM (*Java Virtual Machine*).
    - Modelo de seguridad tipo sandbox proporcionado por la JVM.
    - Abstracción del sistema operativo subyacente mediante un juego completo de APIs de programación.

  Esta versión de Java contiene el conjunto básico de herramientas usadas para desarrollar Java Applets, así como las APIs orientadas a la programación de aplicaciones de usuario final: interfaz gráfica de usuario, multimedia, redes de comunicación, etc.

- *Java 2 Platform, Enterprise Edition* (J2EE): Esta versión está orientada al entorno empresarial. El software empresarial tiene unas características propias marcadas: está pensado no para ser ejecutado en un equipo, sino para ejecutarse sobre una red de ordenadores de manera distribuida y remota mediante EJBs (*Enterprise Java Beans*). De hecho, el sistema se monta sobre varias unidades o aplicaciones. En muchos casos, además, el software empresarial requiere que se

sea capaz de integrar datos provenientes de entornos heterogéneos. Esta edición está orientada especialmente al desarrollo de servicios web, servicios de nombres, persistencia de objetos, XML, autenticación, APIs para la gestión de transacciones, etc. El cometido de esta especificación es ampliar la J2SE para dar soporte a los requisitos de las aplicaciones de empresa.

- *Java 2 Platform, Micro Edition* (J2ME): Esta versión de Java está enfocada a la aplicación de la tecnología Java en dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs o electrodomésticos inteligentes. Esta edición tiene unos componentes básicos que la diferencian de las otras versiones, como el uso de una máquina virtual denominada KVM (*Kilo Virtual Machine*, debido a que requiere sólo unos pocos Kilobytes de memoria para funcionar) en vez del uso de la JVM clásica, inclusión de un pequeño y rápido recolector de basura y otras diferencias.

En esta última edición de Java es en donde se va a centrar todo el estudio de ahora en adelante. En la siguiente gráfica se muestran las distintas ediciones de Java junto con algunos de los aparatos en donde se utilizan.

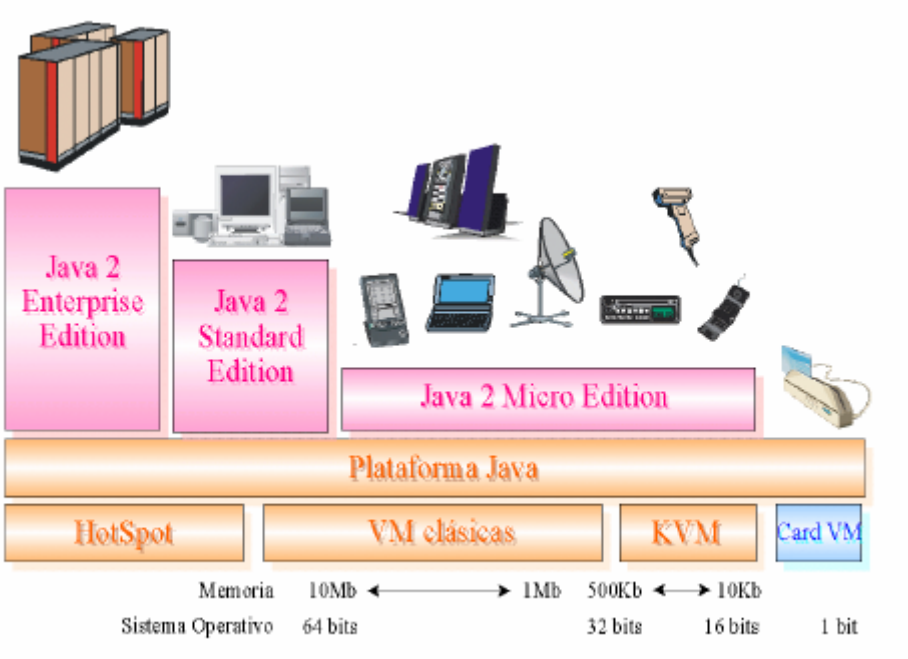

Figura 6.1 Arquitectura de la plataforma Java 2 de Sun.

6.2 Java 2 Micro Edition

La edición *Java 2 Micro Edition* fue presentada en 1999 por Sun Microsystems con el propósito de habilitar aplicaciones Java para pequeños dispositivos. En esta presentación, lo que realmente se enseñó fue una primera versión de una nueva *Java Virtual Machine* (JVM) que podía ejecutarse en dispositivos Palm.

Para empezar podemos decir que *Java Micro Edition* es la versión del lenguaje Java que está orientada al desarrollo de aplicaciones para dispositivos pequeños con capacidades

restringidas tanto en pantalla gráfica, como de procesamiento y memoria (teléfonos móviles, PDAs, Handhelds, Pagers, etc). La tardía aparición de esta tecnología, (hemos visto que la tecnología Java nació a mediados de los 90 y *Java Micro Edition* apareció a finales), puede ser debido a que las necesidades de los usuarios de telefonía móvil ha cambiado mucho en estos últimos años y cada vez demandan más servicios y prestaciones por parte tanto de los terminales como de las compañías. Además el uso de esta tecnología ha dependido del asentamiento en el mercado de otras, como GPRS, íntimamente asociada a J2ME y que no estuvo a nuestro alcance hasta 2001. J2ME es la tecnología del futuro para la industria de los dispositivos móviles. Actualmente las compañías telefónicas y los fabricantes de móviles están implantando los protocolos y dispositivos necesarios para soportarla.

J2ME representa una versión simplificada de J2SE. Sun separó estas dos versiones ya que J2ME estaba pensada para dispositivos con limitaciones de proceso y capacidad gráfica. También separó J2SE de J2EE porque este último exigía unas características muy pesadas o especializadas de E/S, trabajo en red, etc. Por tanto, separó ambos productos por razones de eficiencia. Hoy, J2EE es un superconjunto de J2SE pues contiene toda la funcionalidad de este y más características, así como J2ME es un subconjunto de J2SE (excepto por el paquete javax.microedition) ya que, como se ha mencionado, contiene varias limitaciones con respecto a J2SE.

Sólo de manera muy simplista se puede considerar a J2ME y J2EE como versiones reducidas y ampliadas de J2SE respectivamente, en realidad cada una de las ediciones está enfocada a ámbitos de aplicación muy distintos. Las necesidades computacionales y APIs de programación requeridas para un juego ejecutándose en un móvil difieren bastante con las de un servidor distribuido de aplicaciones basado en EJB.

6.3 Arquitectura de J2ME

Un entorno de ejecución determinado de J2ME se compone de una selección de:

- Máquinas virtuales Java con diferentes requisitos, cada una para diferentes tipos de pequeños dispositivos.
- Configuraciones, que son un conjunto de clases básicas orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas. Existen 2 configuraciones definidas en J2ME:
    - *Connected Limited Device Configuration* (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria.
    - *Connected Device Configuration* (CDC) enfocada a dispositivos con más recursos.
- Perfiles, que son unas bibliotecas Java de clases específicas orientadas a implementar funcionalidades de más alto nivel para familias específicas de dispositivos.

La arquitectura de un entorno de ejecución la podemos ver en la siguiente figura:

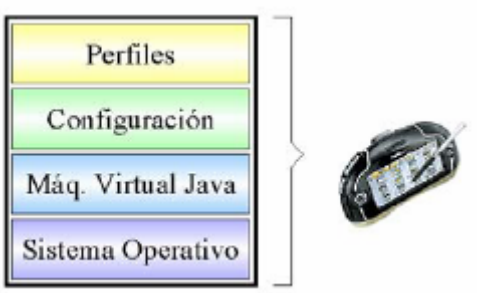

Figura 6.2 Entorno de ejecución.

La relación entre las distintas ediciones de Java y entre las configuraciones definidas en J2ME se muestra en la siguiente figura:

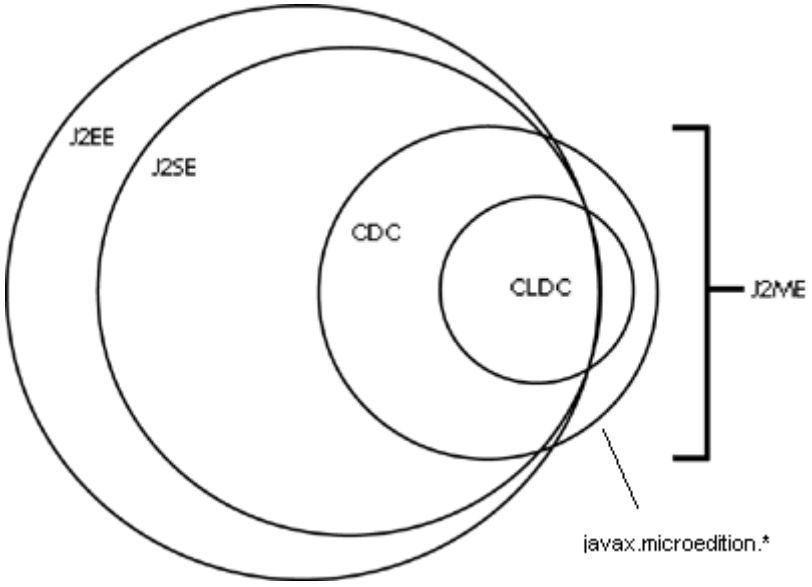

Figura 6.3 Relación entre las APIs y las configuraciones de la plataforma Java.

6.3.1 Máquinas Virtuales J2ME

Una máquina virtual de Java (JVM) es un programa encargado de interpretar código intermedio (bytecode) de los programas Java precompilados a código máquina ejecutable por la plataforma, efectuar las llamadas pertinentes al sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java. De esta forma, la JVM proporciona al programa Java independencia de la plataforma con respecto al hardware y al sistema operativo subyacente. Las implementaciones tradicionales de JVM son, en general, muy pesadas en cuanto a memoria ocupada y requerimientos computacionales. J2ME define varias JVMs de referencia adecuadas al ámbito de los dispositivos electrónicos que, en algunos casos, suprimen algunas características con el fin de obtener una implementación menos exigente.

Cada una de las configuraciones CLDC y CDC requiere su propia máquina virtual. La VM (*Virtual Machine*) de la configuración CLDC se denomina KVM y la de la configuración CDC se denomina CVM

6.3.1.1 KVM

Se corresponde con la Máquina Virtual más pequeña desarrollada por Sun. Su nombre KVM proviene de Kilobyte (haciendo referencia a la baja ocupación de memoria, entre 40Kb y 80Kb). Se trata de una implementación de Máquina Virtual reducida y especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria. La KVM está escrita en lenguaje C, aproximadamente unas 24000 líneas de código, y fue diseñada para ser:

- Pequeña, con una carga de memoria entre los 40Kb y los 80 Kb, dependiendo de la plataforma y las opciones de compilación.
- Alta portabilidad.
- Modulable.
- Lo más completa y rápida posible y sin sacrificar características para las que fue diseñada.

Sin embargo, esta baja ocupación de memoria hace que posea algunas limitaciones con respecto a la clásica *Java Virtual Machine* (JVM):

- No hay soporte para tipos en coma flotante. No existen por tanto los tipos double ni float. Esta limitación está presente porque los dispositivos carecen del hardware necesario para estas operaciones.
- No existe soporte para JNI (*Java Native Interface*) debido a los recursos limitados de memoria.
- No existen cargadores de clases (*class loaders*) definidos por el usuario. Sólo existen los predefinidos.
- No se permiten los grupos de hilos o hilos *daemon*. Cuando queramos utilizar grupos de hilos utilizaremos los objetos Collection para almacenar cada hilo en el ámbito de la aplicación.
- No existe la finalización de instancias de clases. No existe el método Object.finalize().
- No hay referencias débiles. Un objeto que está siendo apuntado mediante una referencia débil es un candidato para la recolección de basura. Estas referencias están permitidas en J2SE, pero no en J2ME.
- Limitada capacidad para el manejo de excepciones debido a que el manejo de éstas depende en gran parte de las APIs de cada dispositivo por lo que son éstos los que controlan la mayoría de las excepciones.
- Reflexión. La reflexión es el mecanismo por el cual los objetos pueden obtener información de otros objetos en tiempo de ejecución como, por ejemplo, los archivos de clases cargados o sus campos y métodos.

Aparte de la no inclusión de estas características, la verificación de clases merece un comentario aparte. El verificador de clases estándar de Java es demasiado grande para la KVM. De hecho es más grande que la propia KVM y el consumo de memoria es excesivo, más de 100Kb para las aplicaciones típicas. Este verificador de clases es el encargado de rechazar las clases no válidas en tiempo de ejecución. Este mecanismo verifica los bytecodes de las clases Java realizando las siguientes comprobaciones:

- Ver que el código no sobrepase los límites de la pila de la VM.
- Comprobar que no se utilizan las variables locales antes de ser inicializadas.
- Comprobar que se respetan los campos, métodos y los modificadores de control de acceso a clases.

Por esta razón los dispositivos que usen la configuración CLDC y KVM introducen un algoritmo de verificación de clases en dos pasos.

El proceso desde que se escribe un programa java hasta que es ejecutado por el usuario queda reflejado en la siguiente figura. Incluye el proceso de compilación y preverificación del código, la descarga del ejecutable, la nueva verificación por parte de la máquina virtual del cliente y finalmente la ejecución del programa con el intérprete de Java.

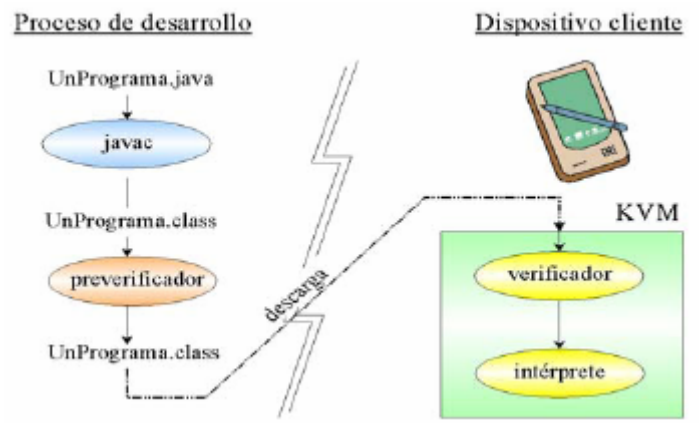

Figura 6.4 Preverificación de clases en CDLC/KVM.

La KVM puede ser compilada y probada en 3 plataformas distintas:

- Solaris Operating Environment.
- Windows.
- PalmOs.

6.3.1.2 CVM

La CVM (*Compact Virtual Machine*) ha sido tomada como Máquina Virtual Java de referencia para la configuración CDC y soporta las mismas características que la Máquina Virtual de J2SE. Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y en torno a 2Mb o más de memoria RAM. Las características que presenta esta Máquina Virtual son:

- Sistema de memoria avanzado.
- Tiempo de espera bajo para el recolector de basura.
- Separación completa de la VM del sistema de memoria.
- Recolector de basura modularizado.
- Portabilidad.
- Rápida sincronización.
- Ejecución de las clases Java fuera de la memoria de sólo lectura (ROM).

- Soporte nativo de hilos.
- Baja ocupación en memoria de las clases.
- Proporciona soporte e interfaces para servicios en Sistemas Operativos de Tiempo Real.
- Conversión de hilos Java a hilos nativos.
- Soporte para todas las características de Java2 v1.3 y librerías de seguridad, referencias débiles, Interfaz Nativa de Java (JNI), invocación remota de métodos (RMI), Interfaz de depuración de la Máquina Virtual (JVMDI).

6.3.2 Configuraciones

Una configuración es el conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos. Estas APIs describen las características básicas, comunes a todos los dispositivos:

- Características soportadas del lenguaje de programación Java.
- Características soportadas por la Máquina Virtual Java.
- Bibliotecas básicas de Java y APIs soportadas.

Existen dos configuraciones en J2ME:

6.3.2.1 CDC

La primera es la configuración de dispositivos con conexión, CDC (*Connected Limited Configuration*). Está orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, decodificadores de televisión digital, televisores con Internet, algunos electrodomésticos y sistemas de navegación en automóviles. CDC usa una Máquina Virtual Java similar en sus características a una de J2SE, pero con limitaciones en el apartado gráfico y de memoria del dispositivo. Esta Máquina Virtual es la que hemos visto como CVM (*Compact Virtual Machine*). La CDC está enfocada a dispositivos con las siguientes capacidades:

- Procesador de 32 bits.
- Disponer de 2 Mb o más de memoria total, incluyendo memoria RAM y ROM.
- Poseer la funcionalidad completa de la Máquina Virtual Java2.
- Conectividad a algún tipo de red.

La CDC está basada en J2SE v1.3 e incluye varios paquetes Java de la edición estándar. Las peculiaridades de la CDC están contenidas principalmente en el paquete javax.microedition.io, que incluye soporte para comunicaciones HTTP y basadas en datagramas.

Los paquetes incluidos en CDC se presentan en la siguiente tabla:

Nombre de Paquete CDC	Descripción
java.io	Clases e interfaces estándar de E/S
java.lang	Clases básicas del lenguaje
java.lang.ref	Clases de referencia
java.lang.reflect	Clases e interfaces de reflection
java.math	Paquete de matemáticas
java.net	Clases e interfaces de red
java.security	Clases e interfaces de seguridad
java.security.cert	Clases de certificados de de seguridad
java.text	Paquete de texto
java.util	Clases de utilidades estándar
java.jar	Clases y utilidades para archivos JAR
java.util.zip	Clases y utilidades para archivos ZIP y comprimidos
java.microdition.io	Clases e interfaces para conexión genérica CDC

Tabla 6.1 Librerías de configuración CDC.

6.3.2.1 CLDC

La segunda es la configuración de dispositivos limitados con conexión, CLDC (*Connected Limited Device Configuration*). Está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Unos ejemplos de estos dispositivos son:

- Teléfonos móviles.
- Buscapersonas (*pagers*).
- PDAs.
- Organizadores personales.
- Etc.

Algunas de estas restricciones vienen dadas por el uso de la KVM, necesaria al trabajar con la CLDC debido a su pequeño tamaño. Los dispositivos que usan CLDC deben cumplir los siguientes requisitos:

- Disponer entre 160 Kb y 512 Kb de memoria total disponible. Como mínimo se debe disponer de 128 Kb de memoria no volátil para la Máquina Virtual Java y las bibliotecas CLDC, y 32 Kb de memoria volátil para la Máquina Virtual en tiempo de ejecución.
- Procesador de 16 ó 32 bits con al menos 25 Mhz de velocidad.
- Ofrecer bajo consumo, debido a que estos dispositivos trabajan con suministro de energía limitado, normalmente baterías.
- Tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

La CLDC aporta las siguientes funcionalidades a los dispositivos:

- Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual (KVM).
- Un subconjunto de las bibliotecas Java del núcleo.
- Soporte para E/S básica.
- Soporte para acceso a redes.
- Seguridad.

Los paquetes incluidos en CLDC se presentan en la siguiente tabla:

Nombre de paquete CLDC	**Descripción**
java.io	Clases y paquetes estándar de E/S. Subconjunto de J2SE
java.lang	Clases e interfaces de la Máquina Virtual. Subconj. J2SE
java.util	Clases, interfaces y utilidade estándar. Subconj. J2SE
javax.microedition.io	Clases e interfaces de conexión genérica CLDC

Tabla 6.2 Librerías de configuración CLDC.

Un aspecto muy a tener en cuenta es la seguridad en CLDC. Esta configuración posee un modelo de seguridad sandbox al igual que ocurre con los applets.

En cualquier caso, una determinada configuración no se encarga del mantenimiento del ciclo de vida de la aplicación, interfaces de usuario o manejo de eventos, sino que estas responsabilidades caen en manos de los perfiles, que se estudian a continuación.

6.3.3 Perfiles

Un perfil es un conjunto de APIs orientado a un ámbito de aplicación determinado. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan (electrodomésticos, teléfonos móviles, etc.) y el tipo de aplicaciones que se ejecutarán en ellos. Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil. Aquí nos podemos encontrar grandes diferencias entre interfaces, desde el menú textual de los teléfonos móviles hasta los táctiles de los PDAs.

El perfil establece unas APIs que definen las características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como de la configuración. Tenemos que tener en cuenta que un perfil siempre se construye sobre una configuración determinada. De este modo, podemos pensar en un perfil como un conjunto de APIs que dotan a una configuración de funcionalidad específica. Un perfil puede ser construido sobre cualquier otro. Sin embargo, una plataforma J2ME sólo puede contener una configuración.

Existen unos perfiles que construiremos sobre la configuración CDC y otros que construiremos sobre la CLDC. Para la configuración CDC, únicamente a título informativo porque este proyecto se basa en aplicaciones sólo para CLDC, tenemos los siguientes perfiles:

- Foundation Profile.
- Personal Profile.
- RMI Profile.

Para la configuración CLDC tenemos los siguientes perfiles:

- *Information Module Profile* (IMP). IMP se usa para dispositivos que no tienen pantalla gráfica por lo que no es de nuestro interés.

- *PDA Profile*. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 píxeles (al menos 200x100 píxeles) con un factor 2:1. No es posible dar mucha más información porque en este momento este perfil se encuentra en fase de definición.

- *Mobile Information Device Profile* (MIDP). Al igual que CLDC fue la primera configuración definida para J2ME, MIDP fue el primer perfil definido para esta plataforma.

  Este perfil está orientado para dispositivos con las siguientes características:

  - Reducida capacidad computacional y de memoria.
  - Conectividad limitada (en torno a 9600 bps).
  - Capacidad gráfica muy reducida (mínimo un display de 96x54 píxels monocromo).
  - Entrada de datos alfanumérica reducida.
  - 128 Kb de memoria no volátil para componentes MIDP.
  - 8 Kb de memoria no volátil para datos persistentes de aplicaciones.
  - 32 Kb de memoria volátil en tiempo de ejecución para la pila Java.

  Los tipos de dispositivos que se adaptan a estas características son:

  - Teléfonos móviles.
  - Buscapersonas (*pagers*).
  - PDAs de gama baja con conectividad.

  El perfil MIDP establece las capacidades del dispositivo, por lo tanto, especifica las APIs relacionadas con:

  - La aplicación (semántica y control de la aplicación MIDP).
  - Interfaz de usuario.
  - Almacenamiento persistente.
  - Trabajo en red.
  - Temporizadores.

  Las aplicaciones realizadas utilizando MIDP reciben el nombre de MIDlets (por analogía a APPlets). Decimos así que un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC.

  Dentro de MIDP existen dos versiones [31]. La versión MIDP 1.0 usa los siguientes paquetes:

Paquetes del MIDP 1.0	Descripción
java.microedition.lcdui	Clases e interfaces para usuarios
java.microedition.rms	*Record Managemetn Storage*. Soporte para el almacenamietnto persistente del dispositivo
java.microedition.midlet	Clases de definición de la aplicación
java.microedition.io	Clases e interfaces de conexión genérica
java.io	Clases e interfaces de E/S básica
java.lang	Clases e interfaces de la Máquina Virtual
java.util	Clases e interfaces de utilidades estándar

Tabla 6.3 Librerías del perfil MIDP 1.0.

MIDP 2.0, especificado en la norma JSR-118, es la versión revisada y mejorada de la especificación MIDP 1.0. Sus aportaciones son:

- Soporte multimedia (subconjunto del *Mobile Media API*).
- Soporte para dibujo a pantalla completa.
- Soporte para transparencias.
- Amplias opciones de conectividad. (HTTPS, *datagram*, *sockets*, *Server sockets*, y comunicación por puerto serie).
- Diversas mejoras específicas para desarrollo de juegos.
- OTA obligatorio.
- Seguridad en las comunicaciones.
- Sólo los móviles más nuevos/caros soportan MIDP2.0.

La versión MIDP 2.0 usa los siguientes paquetes:

Paquetes del MIDP 2.0	Descripción
javax.microedition.lcdui	Clases e interfaces para usuarios
javax.microedition.lcdui.game	Clases e interfaces para juegos
javax.microedition.midlet	Clases de definición de la aplicación
javax.microedition.io	Clases e interfaces de conexión genérica
javax.microedition.pki	Clases de definición de seguridad
javax.microedition.media	Clases e interfaces de sonido
javax.microedition.media.control	Clases e interfaces de control de sonido
javax.microedition.rms	*Record Managemetn Storage*. Soporte para el almacenamietnto persistente del dispositivo
java.io	Clases e interfaces de E/S básica
java.lang	Clases e interfaces de la Máquina Virtual
java.util	Clases e interfaces de utilidades estándar

Tabla 6.4 Librerías del perfil MIDP 2.0.

La siguiente figura muestra la diferencia visual entre las dos versiones de MIDP:

Figura 6.5 Versión MIDP 1 comparada con la versión MIDP 2

El primer modelo que implementó MIDP 2.0 fue el Nokia 6600. Este modelo fue lanzado al mercado en 2004 y resultó ser el más vendido aquel año copando el 16,46 % de las ventas de entre los 30 modelos más vendidos. Sirva este dato para reflejar la interesante proyección comercial de este avance técnico. Actualmente ya se está desarrollando siguiente evolución MIDP 3.0.

La arquitectura completa de J2ME que ya se ha estudiado, máquinas virtuales, configuraciones y perfiles queda reflejada en la siguiente figura:

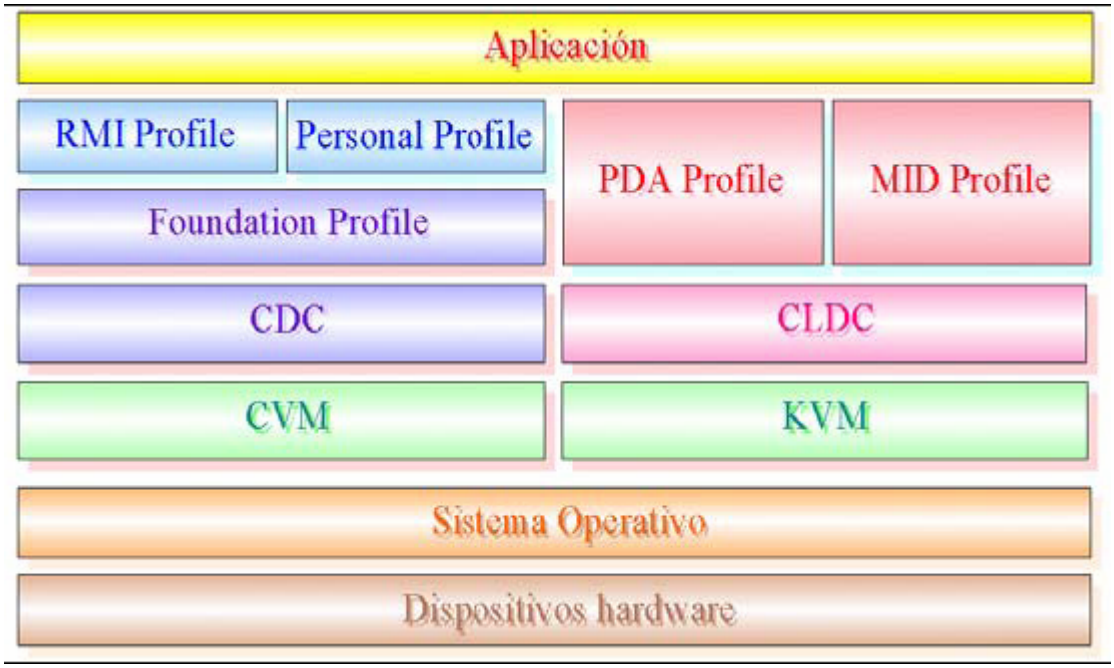

Figura 6.6 Arquitectura del entorno de ejecución de J2ME.

En el siguiente apartado se explica cómo se obtienen los programas para los dispositivos móviles, los MIDlets, y también su entorno de ejecución.

6.4 OTA

Las aplicaciones realizadas con J2ME están pensadas para que puedan ser descargadas a través de una conexión a Internet. El medio empleado para garantizar esta descarga recibe el nombre de OTA (*Over the Air*), y viene totalmente reflejado en un documento denominado "*Over The Air User Initiated Provisioning Recommended Practice*", Sun Microsystems, 2001 [19].

Una aplicación J2ME está formada por un archivo JAR que es el que contiene a la aplicación en sí y un archivo JAD (*Java Archive Descriptor*) que contiene diversa información sobre la aplicación.

Un archivo JAR está formado por los siguientes elementos:

- Un archivo manifiesto que describe el contenido del archivo JAR.
- Las clases Java que forman el MIDlet
- Los archivos de recursos usados por el MIDlet.

El fichero JAD es opcional y describe el contenido del archivo JAR. Contiene atributos de la forma "atributo: valor". La creación de éste puede realizarse desde cualquier editor de texto y tiene los siguientes campos obligatorios:

- MIDlet-n: Contiene una lista con el nombre del MIDlet y también de su icono.
- MIDlet-Name: nombre del MIDlet.
- MIDlet-Vendor: desarrollador.
- MIDlet-Version: versión del MIDlet.
- Microedition-Configuration: configuración necesitada.
- Microedition-Profile: perfil necesitado.

Los siguientes campos son opcionales:

- MIDlet-Description: descripción del MIDlet.
- MIDlet-Icon: nombre del archivo png incluido en el JAR
- MIDlet-Info-URL: URL con información sobre el MIDlet.
- MIDlet-Data-Size: Número de bytes requeridos por el MIDlet.

El propósito de este apartado es describir cómo se pueden descargar los MIDlets "*over the air*" y establecer cuáles son los requerimientos impuestos a los dispositivos que realizan estas descargas.

6.4.1 Requerimientos Funcionales

Los dispositivos deben proporcionar mecanismos mediante los cuales podamos encontrar los MIDlets que deseemos descargar. En algunos casos, encontraremos los MIDlets a través de un navegador WAP o a través de una aplicación residente escrita específicamente para identificar MIDlets. Otros mecanismos como Bluetooth, cable serie, etc, también son válidos.

El programa encargado de manejar la descarga y ciclo de vida de los MIDlets en el dispositivo se llama Gestor de Aplicaciones o AMS (*Application Management Software*).

Un dispositivo que posea la especificación MIDP debe ser capaz de:

- Localizar archivos JAD vinculados a un MIDlet en la red.
- Descargar el MIDlet y el archivo JAD al dispositivo desde un servidor usando el protocolo HTTP 1.1 u otro que posea su funcionalidad.
- Enviar el nombre de usuario y contraseña cuando se produzca una respuesta HTTP por parte del servidor 401 (*Unauthorized*) o 407 (*Proxy Authentication Required*).
- Instalar el MIDlet en el dispositivo.
- Ejecutar MIDlets.
- Permitir al usuario borrar MIDlets instalados.

6.4.2 Localización de la Aplicación

El descubrimiento de una aplicación es el proceso por el cual un usuario a través de su dispositivo localiza un MIDlet. El usuario debe ser capaz de ver la descripción del MIDlet a través de un enlace que, una vez seleccionado, inicializa la instalación del MIDlet. Si éste enlace se refiere a un archivo JAR, el archivo y su URL son enviados al AMS del dispositivo para empezar el proceso de instalación. Si el enlace se refiere a un archivo JAD se realizan los siguientes pasos:

- El descriptor de la aplicación (archivo JAD) y su URL son transferidos al AMS para empezar la instalación. Este descriptor es usado por el AMS para determinar si el MIDlet asociado puede ser instalado y ejecutado satisfactoriamente.

- Este archivo JAD debe ser convertido al formato Unicode antes de ser usado. Los atributos del JAD deben ser comprensibles, acorde con la sintaxis de la especificación MIDP, y todos los atributos requeridos por la especificación MIDP deben estar presentes en el JAD.

- El usuario debería de tener la oportunidad de confirmar que desea instalar el MIDlet. Asimismo debería de ser informado si se intenta instalar una versión anterior del MIDlet o si la versión es la misma que ya está instalada. Si existen problemas de memoria con la ejecución del MIDlet se intentarían solucionar liberando componentes de memoria para dejar espacio suficiente.

6.4.3 Instalación de MIDlets

La instalación de la aplicación es el proceso por el cual el MIDlet es descargado al dispositivo y puede ser utilizado por el usuario.

Cuando existan múltiples MIDlets en la aplicación que deseamos descargar, el usuario debe ser avisado de que existen más de uno.

Durante la instalación, el usuario debe ser informado del progreso de ésta y se le debe de dar la oportunidad de cancelarla. La interrupción de la instalación debe dejar al dispositivo con el mismo estado que cuando se inició ésta.

Veamos cuáles son los pasos que el AMS sigue para la instalación de un MIDlet:

- Si el JAD fue lo primero que descargó el AMS, el MIDlet debe tener exactamente la misma URL especificada en el descriptor.
- Si el servidor responde a la petición del MIDlet con un código 401 (*Unauthorized*) o un 407 (*Proxy Authentication Required*), el dispositivo debe enviar al servidor las correspondientes credenciales.
- El MIDlet y las cabeceras recibidas deben ser chequeadas para verificar que el MIDlet descargado puede ser instalado en el dispositivo. El usuario debe ser avisado de los siguientes problemas durante la instalación:

    - Si no existe suficiente memoria para almacenar el MIDlet, el dispositivo debe retornar el Código de Estado (*Status Code*) 901.
    - Si el JAR no está disponible en la URL del JAD, el dispositivo debe retornar el Código 907.
    - Si el JAR recibido no coincide con el descrito por el JAD, el dispositivo debe retornar el Código 904.
    - Si el archivo manifest o cualquier otro no puede ser extraído del JAR, o existe algún error al extraerlo, el dispositivo debe retornar el Código 907.
    - Si los atributos “MIDlet-Name”, “MIDlet-Version” y “MIDlet Vendor” del archivo JAD, no coinciden con los extraídos del archivo manifest del JAR, el dispositivo debe retornar el Código 905.
    - Si la aplicación falla en la autenticación, el dispositivo debe retornar el Código 909.
    - Si falla por otro motivo distinto del anterior, debe retornar el Código 911.
    - Si los servicios de conexión se pierden durante la instalación, se debe retornar el Código 903 si es posible.

La instalación se da por completa cuando el MIDlet esté a nuestra disposición en el dispositivo, o no haya ocurrido un error irrecuperable.

6.4.4 Actualización de MIDlets

La actualización se realiza cuando instalamos un MIDlet sobre un dispositivo que ya contenía una versión anterior de éste. El dispositivo debe ser capaz de informar al usuario cual es la versión de la aplicación que tiene instalada.

Cuando comienza la actualización, el dispositivo debe informar si la versión que va a instalar es más nueva, más vieja o la misma de la ya instalada y debe obtener verificación por parte del usuario antes de continuar con el proceso.

En cualquier caso, un MIDlet que no posea firma no debe de reemplazar de ninguna manera a otro que sí la tenga.

6.4.5 Ejecución de MIDlets

Cuando un usuario comienza a ejecutar un MIDlet, el dispositivo debe invocar a las clases CLDC y MIDP requeridas por la especificación MIDP. Si existen varios MIDlets presentes, la interfaz de usuario debe permitir al usuario seleccionar el MIDlet que desea ejecutar.

6.4.6 Eliminación de MIDlets

Los dispositivos deben permitir al usuario eliminar MIDlets. Antes de eliminar una aplicación el usuario debe dar su confirmación. El dispositivo debería avisar al usuario si ocurriese alguna circunstancia especial durante la eliminación del MIDlet. Por ejemplo, el MIDlet a borrar podría contener a otros MIDlets, y el usuario debería de ser alertado ya que todos ellos quedarían eliminados.

Todo el proceso de vida de los MIDlet que se ha estudiado queda relacionado mediante el siguiente diagrama:

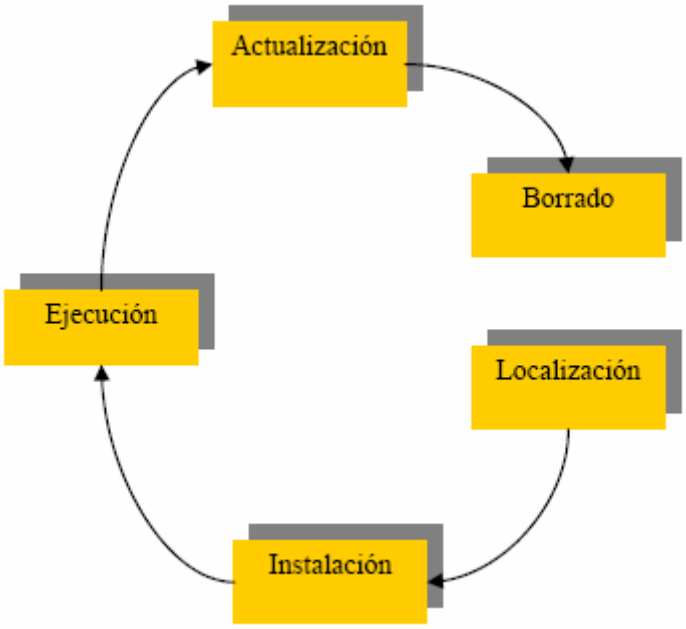

Figura 6.7 Ciclo de vida de un MIDlet.

6.4.7. Estados de un MIDlet

Un MIDlet durante su ejecución pasa por 3 estados diferentes. Estos tres estados son:

- Activo: El MIDlet está actualmente en ejecución.
- Pausa: El MIDlet no está actualmente en ejecución. En este estado el MIDlet no debe usar ningún recurso compartido. Para volver a pasar a ejecución tiene que cambiar su estado a “Activo”.
- Destruido: El MIDlet no está en ejecución ni puede transitar a otro estado. Además se liberan todos los recursos ocupados por el MIDlet.

Un MIDlet puede cambiar de estado mediante una llamada a los métodos MIDlet.startApp(), MIDlet.pauseApp() o MIDlet.destroyApp(). El gestor de aplicaciones cambia el estado de los MIDlets haciendo una llamada a cualquiera de los métodos anteriores. Un MIDlet también puede cambiar de estado por sí mismo.

Ahora vamos a ver por los estados que pasa un MIDlet durante una ejecución típica y cuáles son las acciones que realiza tanto el AMS como el MIDlet. En primer lugar, se realiza la llamada al constructor del MIDlet pasando éste al estado de “Pausa” durante un corto período de tiempo. El AMS por su parte crea una nueva instancia del MIDlet. Cuándo el dispositivo está preparado para ejecutar el MIDlet, el AMS invoca al método MIDlet.startApp() para entrar en el estado de “Activo”. El MIDlet entonces, ocupa todos los recursos que necesita para su ejecución. Durante este estado, el MIDlet puede pasar al estado de “Pausa” por una acción del usuario, o bien, por el AMS que reduciría en todo lo posible el uso de los recursos del dispositivo por parte del MIDlet. Tanto en el estado “Activo” como en el de “Pausa”, el MIDlet puede pasar al estado “Destruido” realizando una llamada al método MIDlet.destroyApp(). Esto puede ocurrir porque el MIDlet haya finalizado su ejecución o porque una aplicación prioritaria necesite ser ejecutada en memoria en lugar del MIDlet. Una vez destruido el MIDlet, éste libera todos los recursos ocupados.

Todos los estados por los que pasa un MIDlet quedan reflejados en el siguiente diagrama:

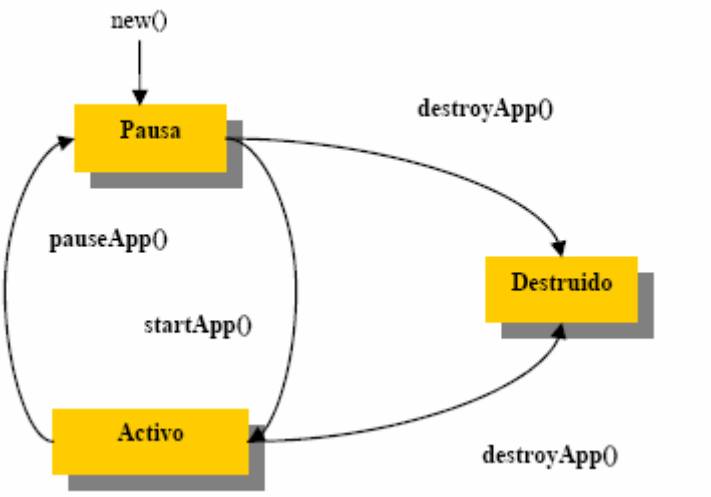

Figura 6.8 Estados de un MIDlet.

6.5 Seguridad en CLDC

Debido a las características de los dispositivos englobados bajo CLDC, en los que se hace necesaria la descarga de aplicaciones y la ejecución de éstas en dispositivos que almacenan información muy personal, se hace imprescindible hacer un gran hincapié en la seguridad. Hay que asegurar la integridad de los datos transmitidos y de las aplicaciones. Este modelo de seguridad no es nuevo, ya que la ejecución de applets (programas Java que se ejecutan en un navegador web) se realiza en una zona de seguridad denominada sandbox y los dispositivos englobados en CLDC se encuentran ante un modelo similar al de los applets.

Este modelo establece que sólo se pueden ejecutar algunas acciones que se consideran seguras. Existen, entonces, algunas funcionalidades críticas que están fuera del alcance de las aplicaciones. De esta forma, las aplicaciones ejecutadas en estos dispositivos deben cumplir unas condiciones previas:

- Los ficheros de clases Java deben ser verificados como aplicaciones Java válidas.
- Sólo se permite el uso de APIs autorizadas por CLDC.
- No está permitido cargar clases definidas por el usuario.
- Sólo se puede acceder a características nativas que entren dentro del CLDC.
- Una aplicación ejecutada bajo KVM no debe ser capaz de dañar el dispositivo dónde se encuentra. De esto se encarga el verificador de clases que se asegura que no haya referencias a posiciones no válidas de memoria. También comprueba que las clases cargadas no se ejecuten de una manera no permitida por las especificaciones de la Máquina Virtual.

6.6 Conclusiones

Java es un lenguaje de programación desarrollado por la empresa Sun Microsystem que se ha extendido mucho debido a su versatilidad y potencia.

Existe una versión simplificada llamada Java 2 Micro Edition (J2ME) enfocada a la aplicación de la tecnología Java en dispositivos con capacidades computacionales y gráficas muy reducidas, tales como los teléfonos móviles. Este lenguaje servirá para implementar los microbrowsers analizados en este proyecto.

El entorno de ejecución de J2ME se compone de una selección de máquinas virtuales Java, configuraciones y perfiles. En el caso de los microbrowsers estamos dentro de la *Kilobyte Virtual Machine* (KVM), *Connected Limited Device Configuration* (CLDC) y *Mobile Information Device Profile* (MIDP), respectivamente. Se trata del entorno de ejecución más limitado en cada apartado, dentro de los entornos gráficos.

Una aplicación J2ME está formada por un archivo JAR que es el que contiene a la aplicación en sí y un archivo JAD que contiene diversa información sobre la aplicación. De modo que esta es también la estructura de los microbrowsers J2ME.