3. J2ME

La plataforma J2ME permite la programación de dispositivos móviles con Java. Este capítulo proporciona una introducción a esta plataforma. En primer lugar se verá cómo es su división en configuraciones y perfiles. A continuación veremos en que consiste una aplicación de tipo MIDlet y se realizará una introducción a su programación.

3.1. Introducción

Sun, dispuesto a proporcionar las herramientas necesarias para cubrir las necesidades de todos los usuarios, creó distintas versiones de Java de acuerdo a las necesidades de cada uno. Según esto nos encontramos con que el paquete Java 2 lo podemos dividir en 3 ediciones distintas:

- Java 2, Standard Edition (J2SE)
- Java 2, Enterprise Edition (J2EE)
- Java 2, Micro Edition (J2ME)

J2SE está destinado a usarse en ordenadores personales, y contiene las herramientas básicas para el desarrollo de Java Applets, así como las APIs orientadas a la programación de aplicaciones de usuario final. J2EE está orientado al entorno empresarial, ya que 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). Java 2 Micro Edition (J2ME) es la plataforma para el desarrollo de aplicaciones Java destinadas a dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs, o electrodomésticos inteligentes. La figura 3.1 muestra un esquema de las diferentes ediciones de Java.

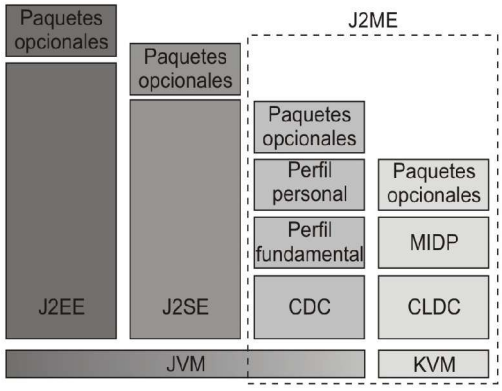

Figura 3.1 Plataformas Java 2

La plataforma J2ME propociona la potencia y beneficios de la tecnología Java (portabilidad del código, programación orientada a objetos, y rápido ciclo de desarrollo) a los pequeños dispositivos. El objetivo fundamental de J2ME es el de descargar aplicaciones dinámicamente que aproveche las posibilidades de cada dispositivo. Los dispositivos a los que se destina van desde dispositivos buscapersonas hasta descodificadores de televisión digital, es decir, dispositivos que en principio varían bastante en cuanto a memoria, potencia de procesamiento, y posibilidades de E/S. Para tratar de englobar toda esta diversidad, la arquitectura J2ME define configuraciones, perfiles y paquetes opcionales, que permiten modularizar y personalizar bastante la plataforma. La figura 3.2 muestra las relaciones que existen entre las distintas capas de la arquitectura J2ME. En las siguientes secciones se da una descripción más detallada de las diferentes capas.

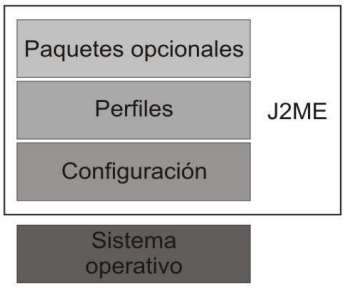

Figura 3.2 Arquitectura J2ME

3.2. Configuraciones

La máquina virtual de Java (JVM) interpreta el código de bytes de Java generado cuando el programa es compilado. De este modo un programa Java puede ser ejecutado en cualquier dispositivo provisto de la máquina virtual de Java y de las librerías de clases apropiadas.

Las configuraciones están compuestas de una máquina virtual de Java y de un conjunto mínimo de librerías de clases. La JVM normalmente se encuentra en la parte más alta del sistema operativo del dispositivo en cuestión. La configuración define la funcionalidad mínima que debe cubrir una categoría o grupo de dispositivos particular. Define con respecto a la máquina virtual de Java las posibilidades mínimas que debe ofrecer y cuáles son sus requisitos para todos los dispositivos de una determinada categoría o grupo. Actualmente, hay dos configuraciones de J2ME: Connected Limited Device Configuration (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria, y Connected Device Configuration (CDC) enfocada a dispositivos con más recursos.

3.2.1. CLDC

La CLDC está orientada a dispositivos con limitaciones en cuanto a capacidad gráfica, cómputo y memoria, y dotados de conexión pero de forma intermitente. Algunos ejemplos de éstos dispositivos son: teléfonos móviles, buscapersonas (pagers), PDAs, organizadores personales, etc. Una implementación de esta configuración incluye generalmente una máquina virtual KVM, cuya particularidad es que tiene un tamaño pequeño, del orden de algunos Kilobytes, por lo que está diseñada especialmente para dispositivos limitados en memoria.

3.2.2. CDC

La CDC está dirigida a dispositivos con mayor capacidad de memoria, mayor potencia computacional y con un mayor ancho de banda para la conexión. Algunos ejemplos típicos son la televisión digital, televisores con internet, algunos electrodomésticos o 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. La CDC incluye en general bastantes más de las características de la plataforma J2SE que CLDC.

3.3. Perfiles

Las configuraciones no son siempre suficientes para proporcionar una solución que se ajuste a cada clase de dispositivos. Los perfiles añaden la funcionalidad y APIs necesarias para completar un entorno de ejecución que se ajuste a cada una. Las configuraciones deben ser combinadas con los perfiles, que definen APIs de alto nivel propocionando posibilidades para un mercado o industria concreto. Algunos ejemplos de perfiles son el Perfil de Dispositivo de Información Móvil (MIDP), el Perfil Fundamental (FP) o el Perfil Personal (PP).

Hay que tener en cuenta que estos perfiles no tienen nada que ver con los perfiles de Bluetooth anteriormente tratados. Recordemos que los perfiles de Bluetooth se refieren a un conjunto de funcionalidades de los protocolos de Bluetooth para un modelo de uso particular. Mientras que los perfiles de J2ME son un conjunto de APIs que extienden la funcionalidad de una configuración J2ME. A continuación se detalla algo más sobre algunos de los perfiles J2ME existentes.

3.3.1. MIDP

Este perfil está construido sobre la configuración CLDC. Al igual que CLDC fue la primera configuración definida para J2ME, MIDP fue el primer perfil definido para esta plataforma. Está disañado para dispositivos como teléfonos móviles, buscapersonas o PDAs de gama baja con conectividad. El perfil MIDP establece las capacidades del dispositivo, por lo tanto,

especifica las APIs relacionadas con aspectos como la interfaz de usuario, el almacenamiento persistente o el trabajo en red. Las aplicaciones que se realizan utilizando MIDP reciben el nombre de MIDlets. Decimos así que un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC. Desde un punto de vista práctico MIDP es el único perfil actualmente disponible.

3.3.2. FP

Es el perfil de más bajo nivel para CDC. Otros perfiles pueden ser añadidos para proporcionar un funcionalidad adicional. FP define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica, como por ejemplo decodificadores de televisión digital. Este perfil incluye gran parte de los paquetes de la J2SE, pero excluye totalmente los paquetes "java.awt" Abstract Windows Toolkit (AWT) y "java.swing" que conforman la interfaz gráfica de usuario (GUI) de J2SE. Si una aplicación requiriera una GUI, entonces sería necesario un perfil adicional.

3.3.3. PP

El Perfil Personal es un subconjunto de la plataforma J2SE v1.3, que proporciona un entorno con un completo soporte gráfico AWT. El objetivo es el de dotar a la configuración CDC de una interfaz gráfica completa, con capacidades web y soporte de applets Java. Este perfil requiere una implementación del Perfil Fundamental. Está orientado a dispositivos como PDAs de alta gama o consolas de juego.

Además existe un Perfil Personal Básico (PBP), que es un subconjunto del PP orientado a dispositivos que sólo requieren un nivel básico de representación gráfica, como un decodificador de televisión.

3.4. Paquetes opcionales

Muchos dispositivos J2ME incluyen tecnologías adicionales como Bluetooth, multimedia, mensajes inalámbricos, o conectividad a bases de datos. Para aprovechar estas tecnologías mediante típicas APIs de Java, existen paquetes adicionales a los ya mencionados. Los fabricantes de los dispositivos pueden añadir estos paquetes según se necesite para utilizar las distintas posibilidades del dispositivo.

Además de las configuraciones, perfiles y paquetes opcionales, los fabricantes pueden definir nuevas clases Java para aprovechar características propias de cada dispositivo. Estas clases se denominan Clases de Licencia Abierta (LOCs). Una LOC define clases disponibles para todos los desarrolladores. Las Clases de Licencia Cerrada (LCCs) definen clases que sólo están

disponibles para el fabricante del dispositivo. Los programas que usen estas clases puede que pierdan la característica de portabilidad, incluso entre dispositivos que tengan la misma configuración y perfil.

3.5. MIDlets

Como ya se ha citado una aplicación creada con el perfil MIDP se denomina MIDlet. Esta sección pretende dar una nociones básicas sobre los MIDlets, viendo cuales son sus propiedades, y conociendo cómo es su ciclo de vida y estados por los que puede pasar. Además se dará una rápida introducción al paquete Java dedicado a la creación de MIDlets, exponiendo además un ejemplo básico.

En los dispositivos susceptibles de ejecutar MIDlets, existe un software encargado de gestionar estas aplicaciones denominado Gestor de Aplicaciones (AMS). Este software será el que nos permita ejecutar, pausar y destruir las aplicaciones. El AMS realiza dos funciones principales, gestionar el ciclo de vida de los MIDlets y controlar los estados por los que pasa mientras está en ejecución.

3.5.1. Ciclo de vida

El ciclo de vida de un MIDlet consta de 5 fases: descubrimiento, instalación, ejecución, actualización y borrado.

- **Descubrimiento:** Es la fase en la que se selecciona, con ayuda del AMS, la aplicación a descargar para ser instalada. El AMS proporciona varias formas de descarga de la aplicación, que se podrán llevar o no a cabo según las posibilidades del dispositivo. Por ejemplo, se puede realizar conectando el dispositivo a un PC mediante un cable o de forma inalámbrica vía Bluetooth.
- **Instalación:** Una vez descargada la aplicación, se pasa al proceso de instalación del mismo, controlado por el AMS que informará al usuario de como se va desarrollando todo el proceso. Cuando el MIDlet está instalado, todas las clases, archivos y almacenamiento persistente del dispositivo están a su disposición.
- **Ejecución:** La ejecución de la aplicación se podrá realizar con ayuda del AMS, que además gestionará los estados en los que se encuentra el MIDlet, en función de los eventos que se produzcan.
- **Actualización:** El AMS debe de ser capaz de detectar si un MIDlet que se descarga es una actualización de uno ya existente en el dispositivo e informar al usuario de lo mismo.

- **Borrado:** También el AMS es encargado de borrar el MIDlet del dispositivo, pidiendo antes confirmación al usuario e informando de todo el proceso.

3.5.2. Estados de un MIDlet en ejecución

Un MIDlet durante su ejecución se puede encontrar en tres estados diferentes: activo, suspendido y destruído. Como ya hemos citado los estados son gestionados por el AMS. El cambio de un estado a otro lo puede realizar el mismo MIDlet llamando al método adecuado, aunque también lo puede realizar el mismo AMS porque lo vea conveniente. Por ejemplo, si el MIDlet se ejecuta en un teléfono que recibe un llamada o un SMS, entonces el AMS suspenderá el MIDlet. En general, se cambia de un estado a otro llamando a los métodos `startApp()`, `pauseApp()` o `destroyApp()`. En la figura 3.3 se muestra cuales son las transiciones posibles de un estado a otro y mediante qué métodos.

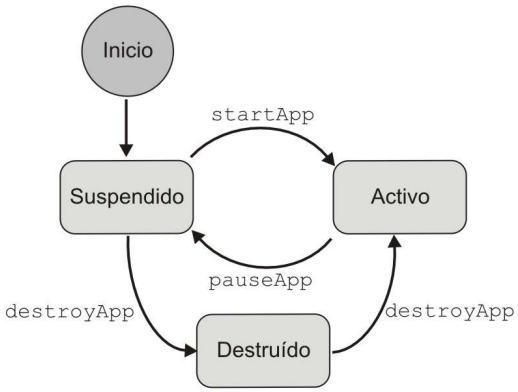

Figura 3.3 Estados de un MIDlet

En una ejecución típica el MIDlet pasará por los tres estados, y normalmente lo hará de la siguiente manera. En primer lugar se realiza la llamada al constructor del MIDlet pasando al estado suspendido 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 `startApp()` para entrar en el estado activo, en el que el MIDlet ocupará todos los recursos necesarios para su ejecución. Durante este estado el MIDlet puede pasar al estado suspendido 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. Esto se realizaría llamando al método `pauseApp()` y, para volver al estado activo, habría que llamar de nuevo a `startApp()`. Tanto en el estado activo como suspendido, el MIDlet puede pasar al estado destruido realizando una llamada al método `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 se liberan todos los recursos ocupados.

3.5.3. Estructura de un MIDlet

Una vez visto el ciclo de vida y los estados de un MIDlet vamos a ver como sería su estructura básica. Así se verá someramente la API empleada para crear un MIDlet. Tampoco se entrará en demasiado detalle pues, a pesar de que en el proyecto se emplea, no es el objetivo primordial de éste.

Antes que nada, veamos que paquetes se usan principalmente para la creación de un MIDlet:

- `javax.microedition.midlet`, que sería el paquete fundamental para la creación de un MIDlet y que define las relaciones de éste con su entorno.
- `javax.microedition.lcdui`, que provee una API para la implementación de interfaces gráficas de usuario.

a) Clase MIDlet

Para la creación de una aplicación MIDP tenemos que partir de la clase `MIDlet`, ya que cualquier MIDlet debe heredar esta clase. Esta clase la podemos encontrar en el paquete `javax.microedition.midlet`. Los métodos de esta clase están bastante ligados a los estados del MIDlet, anteriormente mencionados. Por un lado contiene los métodos `startApp()`, `pauseApp()`, y `destroyApp()`, que se ejecutarán al entrar en los estados correspondientes. Otros métodos, como `notifyDestroyed()` o `notifyPaused()`, indican al AMS que el MIDlet desea entrar en el estado destruido y suspendido respectivamente. Para más detalles sobre el resto de los metodos consultar la especificación de MIDP 2.0 [E6].

Además de la clase `MIDlet` en este paquete se encuentra la excepción `midletStateChangeException`, que es lanzada cuando no es posible cambiar de un estado a otro.

b) Interfaces gráficas de usuario

La regla básica del funcionamiento de un MIDlet es visualizar determinadas informaciones y dirigir consultas al usuario en forma de una serie de formularios sencillos, visualizados en la pantalla del dispositivo. El control de esta pantalla se basa en la clase `Display`, que representa un manejador de la pantalla y los dispositivos de entrada. Todo MIDlet debe poseer como mínimo un objeto de esta clase, que se obtendrá llamando al método `Display.getDisplay()`. La llamada a este método se deberá realizar en el constructor del `MIDlet`, para asegurarnos que exista desde el principio de la ejecución. En la figura 3.4 se puede ver la jerarquía de clases derivada de `Display`.

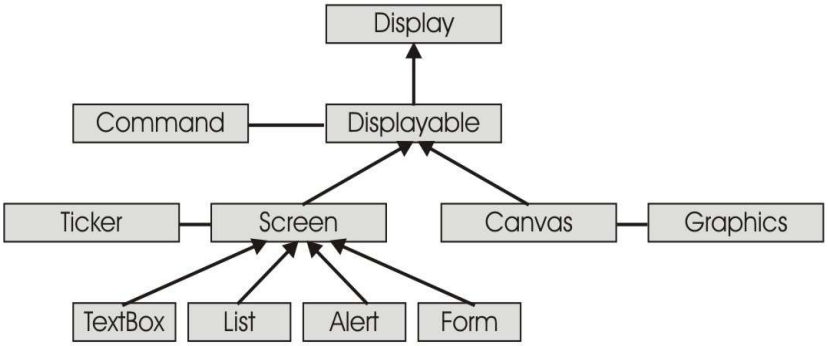

Figura 3.4 Jerarquía de clases de Display

Un MIDlet tendrá en general distintas pantallas que irán alternándose en el dispositivo mostrando distintas informaciones al usuario. La clase `Displayable` representa las pantallas de la aplicación, que podrán ser de diferente tipo. Para cambiar de una a otra el objeto `Display` nos proporciona el método `setCurrent`.

Dentro de la clase `Displayable` se encuentra la clase `Screen` que representa las pantallas de alto nivel, que proporcionan componentes típicos para la interfaz de usuario como formularios, cajas de texto, botones, etc. Éstas no dejan mucha libertad en cuanto al aspecto se refiere, ya que éste depende más que nada de la implementación, pero lo que si aporta es una gran portabilidad entre los distintos dispositivos en los que se puede ejecutar.

Por otro lado existen las pantallas de bajo nivel, englobadas dentro de la clase `Canvas`, que dan un control completo sobre todo lo que aparece en pantalla. Además ofrecen un control total sobre los recursos del dispositivo y se podrán controlar eventos de más bajo nivel. Estas pantallas son necesarias sobre todo en el desarrollo de juegos.

Para la realización de este proyecto nos interesarán principalmente las de alto nivel. Por ello a continuación se detallan las distintas pantallas de este tipo, derivadas de la clase `Screen`:

- `Alert`: La clase alerta muestra un aviso al usuario, a la que se le puede configurar un cierto tiempo de espera antes de mostrar la siguiente pantalla. Una alerta puede contener un título, texto y una imagen opcional. Su propósito fundamental es el de informar de errores u otras situaciones excepcionales.
- `List`: Permite construir pantallas que poseen una lista de opciones. Esto nos será muy útil para crear menús de manera independiente.
- `TextBox`: Esta clase es una pantalla que permite al usuario escribir o editar un texto.
- `Form`: Es una pantalla que puede contener muchos elementos de distinto tipo, como imágenes, textos, campos editables de texto, barras de progreso o grupos de selección. Estos elementos pertenecerán a la clase `Item`.

c) Soporte de eventos

Las clases que van a dedicarse al control de interacciones por parte del usuario deben implementar la interfaz `CommandListener`, para lo que tendrán que definir su método `commandAction()`, que será llamado cuando la pantalla activa en ese momento reciba un evento. En este método se llevará a cabo el código necesario según el comando que se reciba.

Los comandos serán instancias de la clase `Command`, que tiene algunas propiedades interesantes. En general, podemos decir que un comando mantiene información sobre un evento. Serían equivalentes a los botones de Windows por establecer una analogía. La especificación MIDP supone en el dispositivo un máximo de tres botones. Pero se pueden añadir muchos más comandos, y el dispositivo los irá agrupando entre estos tres botones y menús emergentes de la mejor forma posible.

Al crear un comando, además de especificar la etiqueta, que será el nombre que se muestre en pantalla, habrá que indicar su tipo y prioridad. El tipo indicará como será la acción de ese comando. Así `BACK` se utiliza para volver a la pantalla anterior o `EXIT` para salir de la aplicación en curso. Otros tipos posibles son `CANCEL`, `HELP`, `ITEM`, `OK`, `SCREEN`, y `STOP`. Por otro lado, la prioridad también será bastante útil, ya que podrá servir por ejemplo al AMS a la hora de colocar los comandos en la pantalla.

A continuación se expone un ejemplo en el que se hace uso de los conceptos básicos vistos hasta ahora necesarios para crear un MIDlet:

```
/*
 * HolaMundo.java
 */

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 * Ejemplo de MIDlet que muestra un texto y que tiene un comando exit
 * para abandonar la aplicación.
 */
public class HolaMundo extends MIDlet implements CommandListener {

    private Command exitCommand; // Comando exit
    private Display display;    // Display único para este MIDlet

    // Constructor
    public HolaMundo() {
        display = Display.getDisplay(this);
        exitCommand = new Command("Exit", Command.SCREEN, 2);
    }

    /**
     * Inicia el MIDlet, creando un formulario que muestra un texto y
     * asociandole el comando exit y el manejador de eventos (listener).
     */
    public void startApp() {
        Form formulario = new Form("MIDlet básico");

        formulario.append("Hola mundo");

        formulario.addCommand(exitCommand);
```

```
        formulario.setCommandListener(this);

        display.setCurrent(formulario);
    }

    /**
     * Al entrar en el estado Suspendido no se hace nada.
     */
    public void pauseApp() {
    }

    /**
     * Al entrar en estado Destruido se suelen liberar los recursos.
     * En este ejemplo no hay nada que liberar.
     */
    public void destroyApp(boolean unconditional) {
    }

    /*
     * Este es el método que implementa las acciones a llevar a cabo al
     * recibir algún comando. Cuando se reciba exit se destruye el MIDlet.
     */
    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

3.6. Operaciones de E/S con GFC

Por último veamos como hacen las aplicaciones desarrolladas con J2ME para llevar a cabo operaciones de E/S. Para esto tendrán que hacer uso de los paquetes `java.io` y `javax.microedition.io`, que forman lo que se denomina Generic Connection Framework (GFC).

La idea del GFC es la de crear un marco para abstraer el proceso de comunicación a través de una única clase llamada `Connector`. De este modo se puede usar `Connector` para crear conexiones de cualquier tipo como pueden ser flujos de E/S de ficheros, sockets de TCP/IP, conexiones HTTP, etc. Para usar una conexión sólo habrá que utilizar el método `open` de esta clase. Como parámetro tomará la cadena de conexión. Por lo que en general para crear una conexión sólo habrá que usar este método de la siguiente forma:

```
Connector.open("protocolo:direccion:parametros");
```