UNIVERSIDAD NACIONAL AUTONOMA DE NICARAGUA – LEÓN

FACULTAD DE CIENCIAS

DEPARTAMENTO DE COMPUTACION

SISTEMA DE PEDIDOS PARA DISPOSITIVOS MÓVILES: J2ME Y HERRAMIENTAS DE DESARROLLO.

Monografía para optar al título de Lic. en Computación

Autor:

- Br. Walter Leonel Baldelomar Estrada

Tutor:

M.Sc. Álvaro Altamirano

León, Nicaragua. **Junio 2011**

Agradecimientos

Agradezco primeramente a Dios por haberme permitido llegar hasta este punto.

A mi Tutor, M.Sc. Álvaro Altamirano por darme la oportunidad de trabajar con él.

Dedicatoria

A aquel ser que dio descendencia en Mí y que siempre será mi Padre por ser respecto de Mí y al cual siempre lo tendré en mi corazón.

A mi Madre que siempre me apoyó y motivó en cuanto pudo.

A mi hija Rose Arianna Baldelomar por servir de motivación, todo por el bien de ella.

A nuestro Padre que en algún momento me dio consejos y ayudó en mi educación.

Índice

Introducción

En la actualidad al ver a nuestro alrededor, podemos observar que el mundo móvil adquiere cada vez mayor importancia y presencia en nuestras vidas diarias. Pero debemos darnos cuenta que más que una moda, la tecnología móvil es una evolución que ha transformado la forma en que nos divertimos, en que trabajamos, como nos informamos y como nos comunicamos.

Dentro de este mundo móvil uno de los integrantes más importantes son los sistemas de cómputo móvil, implantados en dispositivos tales como **Teléfonos**, **IPHONE**, **IPODS**, **PDA** y toda la gama de dispositivos móviles de última generación. Podemos considerarlos como uno de los grupos más importantes debido a que su uso ha tenido un crecimiento bastante considerable, incorporando grandes avances tecnológicos y revolucionando la forma en que las personas trabajan, permitiéndoles hacerlo donde y cuando deseen, incrementando su productividad y eficiencia.

Pero toda nueva tecnología trae consigo problemas y necesidades inéditas y en el caso de estos dispositivos una de las necesidades es poderlos utilizar más que para una simple llamada. Hoy en día se nos hace necesidad lo que tiempos pasados era un lujo, ya sea enviar y recibir correos a sus familiares en países lejanos, unirse a redes sociales y por qué no manejar parte de su negocio a través de un celular.

Este documento aborda el desarrollo de software con **J2ME**, conceptos relacionados a esta tecnología y una aplicación **Cliente-Servidor** destinada a la Toma de Pedidos utilizando herramientas de software libre.

La tesis está organizada en 4 Capítulos: **Capítulo 1** Introducción a Java y la Plataforma J2ME. En el **Capítulo 2** Conectividad a la Red y Comunicación y Configuración Cliente Servidor (Aplicación de Ejemplo), **Capitulo 3** Autenticación y el **Capitulo 4** Describe una aplicación y herramientas de desarrollo.

Objetivos

General:

- Desarrollar un sistema simulado de pedidos de Productos utilizando J2ME.

Específicos:

- Indagar Ventajas y Desventajas de desarrollar una aplicación utilizando este tipo de comunicación.
- Obtener información apropiada en el punto mismo de la toma de orden.
- Suministrar la habilidad de ingreso de datos de los pedidos en el dispositivo móvil.

Capítulo 1
1.1 Introducción a Java

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros a memoria.

Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun.

En 1998, lanzan la plataforma Java2, y para 1999 deciden dividir el lenguaje en tres partes, orientándolo cada una de ellas a un segmento de Mercado, Servidores, Usuarios y Dispositivos de Usuario.

Así nacen las tres plataformas de Java:

- J2SE orientada al diseño de aplicaciones en el cliente
- J2EE orientada a la creación de aplicaciones empresariales
- J2ME orientada a aplicaciones que se ejecutan en dispositivos móviles.

Entre diciembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java aún no lo es.

1.2 Plataforma J2ME

La plataforma Java 2 Micro Edition es la plataforma para dispositivos de consumo y dispositivos integrados como teléfonos móviles, PDA's, adaptadores de televisión, sistemas de control de vehículos y un gran número de sistemas empotrados. Tal y como pasa con las demás plataformas Java (J2EE, J2SE y Java Card), la plataforma J2ME es un conjunto de API's estándar definido a través del programa Java Community Process, por los grupos de expertos que incluyen las compañías líderes de dispositivos, creadores de software y proveedores de servicios.

La plataforma J2ME proporciona a los dispositivos una interface flexible, un modelo de seguridad, un amplio rango de protocolos de red y soporta tanto las aplicaciones en red como las desconectadas. Con J2ME, las aplicaciones se escriben una vez para un tipo de dispositivos y se pueden descargar dinámicamente y utilizar las capacidades nativas de cada dispositivo.

1.2.1 Arquitectura J2ME.

La arquitectura J2ME define Configuraciones, Perfiles y Paquetes opcionales para construir el entorno de ejecución de Java con el que cubren gran variedad de dispositivos en distintos tipos de mercados. En cada combinación ésta optimiza la memoria, la capacidad de proceso, y las capacidades de entrada / salida de cada dispositivo dando como resultado una plataforma común que potencia las capacidades de cada dispositivo movil.

Figura 1: Arquitectura de la plataforma Java 2

1.2.2 Configuraciones:

Las configuraciones están compuestas de una máquina virtual, y un mínimo conjunto de librerías. Estas proporcionan las funcionalidades básicas para un tipo particular de dispositivos, que comparten características similares, como pueden ser conexiones de red.

Actualmente, hay dos tipos de configuraciones J2ME:

1. La Connected Limited Device Configuration (CLDC)
2. Connected Device Configuration (CDC).

- CLDC es la más pequeña de las dos configuraciones, diseñada para dispositivos conectados a la red intermitentemente, procesadores lentos, y memoria limitada, y está orientada para dispositivos como teléfonos móviles, PDA's etc. Estos dispositivos tienen normalmente CPU's de 16 o 32 bits, y un mínimo de 128 a 512 Kb de memoria disponible para la plataforma Java y aplicaciones asociadas.

- CDC está diseñada para dispositivos que tienen más memoria, procesadores más rápidos y un gran ancho de banda, tv set-top boxes, gateways residenciales, sistemas integrados en vehículos y PDA's de alto nivel. CDC incluye una máquina virtual completa, y un subconjunto de librerías de J2SE mucho más amplio que CLDC. Como resultado, la mayor parte de los dispositivos CDC tiene procesadores de 32 bits, y un mínimo de 2 Mb de memoria disponible para la plataforma Java y los programa asociados.

1.2.3 Perfiles:

Un perfil define el tipo de dispositivo soportado. Por ejemplo: El perfil para Dispositivos de Información Móvil **(Mobile Information Device Profile (MIDP))** define las clases para las aplicaciones que se ejecutan en teléfonos celulares y buscapersonas.

Los perfiles tienen sus ventajas y desventajas. Por un lado proporcionan mecanismos muy importantes y necesarios para la funcionalidad que se requiere de las aplicaciones ejecutándose sobre dispositivos de tan limitada capacidad, pero también esto hace que los fabricantes proporcionen perfiles propietarios para sus propios dispositivos, de forma que las aplicaciones escritas para los aparatos de un fabricante pueden no funcionar sobre aparatos semejantes de otros fabricantes aunque todos soporten el mismo perfil. Para poder utilizar un perfil el dispositivo debe cumplir los requisitos mínimos de la configuración en que se basa y también los requisitos adicionales definidos en la especificación formal del perfil.

Actualmente hay muchos perfiles definidos o en proceso de definición, lo que da idea la importancia que está adquiriendo la tecnología J2ME dentro del mundo de los dispositivos móviles. El principal perfil y el más utilizado por haber sido el primero del cual se ha proporcionado una implementación es el perfil ***MIDP,*** que es un perfil basado en la configuración ***CLDC*** para ejecutar aplicaciones en teléfonos celulares, buscapersonas, dispositivos con pantallas reducidas, conexión HTTP inalámbrica y memoria limitada. (*Perfil implementado en la aplicación de ejemplo que se explicara más adelante*).

Hay otros perfiles como por ejemplo:

- **Foundation Profile (FP):** Conjunto de API's para aplicaciones que se ejecutan sobre dispositivos pequeños con algún tipo de conexión de Red y que no necesitan interfaz gráfica. Ej.: gateways, routers, etc.
- **Personal Profile (PP):** Orientada a dispositivos con necesidades de alto grado de conexión a internet y fidelidad a la web con soporte para applets y AWT.
- **Personal Basics Profile (PBP):** Proporciona un entorno para dispositivos conectados a red que soporten un nivel básico de capacidades graficas no basadas en AWT: Televisión Digital, Sistemas de Navegación de Coches, etc.

1.2.4 Paquetes opcionales:

La plataforma J2ME puede ser extendida mediante la combinación de paquetes opcionales con CLDC, CDC y sus correspondientes profiles. Creados para resolver necesidades específicas del mercado. Los paquetes opcionales ofrecen un conjunto estándar de API's para utilizar con tecnologías emergentes como Bluetooth, servicios web, redes wireless, conexión con base de datos etc. Ej.:

- Mobile Media API
- Wireless Messaging API
- Java Bluetooth APIs
- J2ME Web Service
- Location API for J2ME
- Mobile 3D Graphics API for J2ME, etc.

1.3 Connected Limited Device Configuration

Esta configuración tiene por objetivo mantener en lo posible las características de Java sobre dispositivos limitados a procesadores de 16 y 32 bits de entre 8 y 32 MHz y con una cantidad de memoria total disponible para la plataforma Java de al menos 160 – 192 Kb. Para este dispositivo es imprescindible disponer de una Máquina Virtual de Java adaptadas a las restricciones impuestas por las características especiales de los mismos. Esta Máquina Virtual de Java específica es la KVM (*Kilo Virtual Machine*). CLDC es la implementación más minimalista de J2ME y es fundamental de su conocimiento para desarrollar bajo el perfil MIDP.

Existen tres versiones de CLDC:

- CLDC 1.1 (JSR 139): Es una revisión de la especificación CLDC 1.0 e incluye nuevas características: punto flotante, junto con otras mejoras.
- CLDC 1.0 (JSR 30).
- CLDC HotSpot Implementation TM: Es una máquina virtual muy optimizada que presenta una diferencia de rendimiento muy alta frente a la KVM. Incluye características como: ejecución más rápida de aplicaciones, gestión de recursos más eficientes, manteniendo los requisitos en cuanto a plataforma de ejecución.

También se definen requisitos para la Máquina Virtual Java que ejecutará las aplicaciones en este tipo de dispositivos. Aquí mencionamos algunas.

- No hay soporte para coma flotante
- No finalización de objetos ni referencias inconsistentes.
- No hay soporte para la serialización de objetos.
- El control de errores está definido por la implementación. Esto es para todos aquellos errores en tiempo de ejecución no definidos.

1.4 Mobile Information Device Profile JSR 118 (MIDP 2.0)

Es una versión de (Java 2 Micro Edition) integrada en el hardware de celulares que permite el uso en estos de aplicaciones java denominadas MIDlets, mejorando el modelo de seguridad, incorporando soporte para juegos 2D y sonido, mejorando también los controles de posicionamiento de elementos en la pantalla y la interfaz de usuario de estos elementos. Hasta el momento se han definido dos perfiles, el MIDP 1.0 y el MIDP 2.0.

1.4.1 MIDP 1.0

MIDP 1.0 es la versión inicial de MIDP que contiene las características de interfaz de usuario para trabajar en Java ME. Trabaja sobre CLDC 1.0 o 1.1. Para superar las limitaciones de MIDP 1.0 muchos de los fabricantes han incluido algunas librerías propietarias, como la Nokia UI API.

Hardware

Este perfil está orientado para dispositivos con las siguientes características:

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

Capacidades de las API’s

El MIDP 1.0 incluyó solo las Apis que eran consideradas como requerimiento absoluto para alcanzar una amplia portabilidad, estas Apis están relacionadas con:

- Aplicación (Definición de la semántica y control de la aplicación MIDP)
- Interfaz de usuario (manejo de entradas y Despliegues)
- Almacenamiento persistente
- Comunicaciones en red.
- Temporizadores(Timers)

1.4.2 MIDP 2.0

MIDP 2.0 (JSR 118) es una extensión y mejora a MIDP 1.0. Todos los equipos nuevos que se fabrican en la actualidad poseen MIDP 2.0. Estos dispositivos son compatibles con aplicaciones y juegos MIDP 1.0.

Usualmente este perfil funciona sobre CLDC 1.1.

Hardware

Para poder ejecutar aplicaciones MIDP 2.0, los dispositivos deberían tener las siguientes características mínimas:

- **Despliegue**
  - Tamaño de Pantalla: 96x54 píxeles
  - Profundidad de despliegue: 1-bit
- **Entradas:**
  - Teclado
  - Pantalla de tacto
- **Memoria:**
  - 256kilobytes de memoria no volátil para la aplicación MIDP, más de la que requiere CLDC.
  - 8kilobytes de memoria no volátil para datos persistentes creados en la aplicación.
  - 128kilobytes de memoria volátil para el ambiente de ejecución Java.
- **Conexión a redes:**
  - Dos vías, acceso inalámbrico posiblemente intermitente con ancho de banda limitado.
  - Sonido:
  - Capacidad para reproducir tonos, vía Hardware o software.

Capacidades de las API's

Esta versión incluyó las API's consideradas como requerimiento indispensable para asegurar la portabilidad de las aplicaciones.

- Seguridad
- Firmado de las aplicaciones
- Soporta reproducción de audio
- Soporta modo pantalla completa
- Incluye GAME API para la programación de juegos
- Soporta HTTPS
- Puede trabajar con imágenes RGB a nivel de píxeles
- Nuevos controles y características para interfaces de alto nivel
- Arquitectura Push

1.4.3 Modelo de Seguridad de MIDP

El modelo de seguridad de MIDP ha evolucionado a través de sus distintas versiones. En las primeras dos versiones el principal objetivo fue la protección de funciones sensibles del dispositivo, mientras que en la tercera versión la cual no se tratara en este documento, busca proteger los recursos de una aplicación que pueden compartirse con otras aplicaciones. Estos dos niveles de seguridad se describen a continuación.

Seguridad a Nivel de Plataforma

En MIDP 1.0 toda aplicación que no haya sido preinstalada en el dispositivo (por ejemplo, por el fabricante), se ejecuta en un "*ambiente controlado*" denominado ***sandbox*** que prohíbe el acceso a aquellas funciones del dispositivo que se consideran potencialmente peligrosas.

En MIDP 2.0 se refina el modelo sandbox (caja de arena) en uno menos restrictivo, basado en el concepto de "*dominio de protección*". En los dispositivos compatibles con MIDP 2.0cada aplicación instalada está ligada a un único dominio de protección. La política de seguridad es una ***matriz de control de acceso***, donde los sujetos son las aplicaciones asociadas a cada dominio de protección y los objetos son las funciones del dispositivo que deben protegerse.

El dominio de protección otorga permisos de dos formas distintas a las aplicaciones que resultan ligadas al mismo. La primera forma es otorgando el permiso de manera incondicional. La segunda forma implica la participación del usuario mediante una modalidad de intervención: blanket, el permiso otorgado es válido mientras la aplicación está instalada; session, el permiso otorgado es válido mientras la aplicación está ejecutándose; y oneshot, el permiso es concedido sólo para ese uso. Por lo tanto, el uso de ciertos recursos sensibles involucra una consulta al usuario y en función de su respuesta se determina la disponibilidad del recurso.

El conjunto de permisos efectivamente otorgados a una aplicación se determina en función de su dominio de protección, de los permisos solicitados por la aplicación y de los permisos otorgados por el usuario. Una aplicación posee un determinado permiso cuando lo ha solicitado, y además su dominio de protección otorga el permiso incondicionalmente, o su dominio especifica un modo de interacción con el usuario para el permiso, y el usuario ha dado una autorización que continúa siendo válida.

El modelo establece dos clases de aplicaciones.

1. **Una aplicación no confiable:** es aquella para la que no se ha podido determinar el origen y la integridad. Toda aplicación no confiable, así como una desarrollada sobre MIDP 1.0, se asocia a un dominio de protección que provee un entorno de ejecución restringido.
2. **Una aplicación confiable:** es aquella para la que se ha podido determinar su origen e integridad. Cuando se verifica el origen de la aplicación, se determina también el dominio de protección al que se asocia.

Paquete de Seguridad
MIDP 2.0 mejora de manera considerable el modelo de seguridad del MIDP 1.0 a través del paquete ***javax.microedition.pki***. Este paquete provee el manejo de certificados que son usados para la autenticación de la información en conexiones seguras.

En MIDP 2.0 es posible realizar *firmado de aplicaciones* y *manejar dominios privilegiados*. Con el firmado de aplicaciones podemos comprobar la identidad del proveedor de la aplicación y por lo tanto confiar en la misma. A través de los privilegios de dominio los proveedores de aplicaciones y los vendedores de dispositivos, pueden definir qué API's son consideradas como restringidas. Estas nuevas características protegen al dispositivo frente accesos no autorizados por parte de las aplicaciones. Los protocolos seguros estándar como HTTPS, TLS/SSL y WTLS permiten la transmisión segura de datos a través de la encriptación de los mismos.

1.4.4 Los MIDlet
Las aplicaciones J2ME desarrolladas bajo la especificación MIDP se denominan MIDLets, los cuales derivan de la clase MIDlet, la cual está contenida dentro del paquete *javax.microedition.midlet.*

Los MIDLets, son empaquetados en ficheros .jar. Se requiere alguna información extra para la puesta en marcha de las aplicaciones. Esta información se almacena en el fichero de manifiesto, que va incluido en el fichero .jar y en un fichero descriptor, con extensión .jad. Un fichero .jar típico se compone de:

- Clases del MIDLet
- Clases de soporte
- Recursos (imágenes, sonidos...)
- Manifiesto (fichero ".mf")
- Descriptor (fichero ".jad")

Un fichero .jar puede contener varios MIDLets. Esta colección de MIDLets, se suele llamar MIDLet Suite. Esta unión de varios MIDLets en una distribución, permite compartir recursos tales como imágenes o sonidos y, por tanto, optimizar los recursos del dispositivo.

Un MIDlet durante su ejecución pasa por 3 estados diferentes:

- Activo: el MIDlet está actualmente en ejecución.
- Pausa: El MIDlet no está en ejecución. En este estado el MIDlet no usa ningún recurso compartido. Para volver a pasar a ejecutado 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.

El Gestor de Aplicaciones

El gestor de aplicaciones o AMS (Application Management System) es el software encargado de gestionar los MIDlets. Este software reside en el dispositivo y es el que permite ejecutar, pausar o destruir las aplicaciones J2ME.

El AMS realiza dos grandes funciones:

- Por un lado gestiona el ciclo de vida de los MIDlets.
- Por otro, es el encargado de controlar los estados por los que pasa el MIDlet mientras está en la memoria del dispositivo, es decir, en ejecución.

Los MIDlets

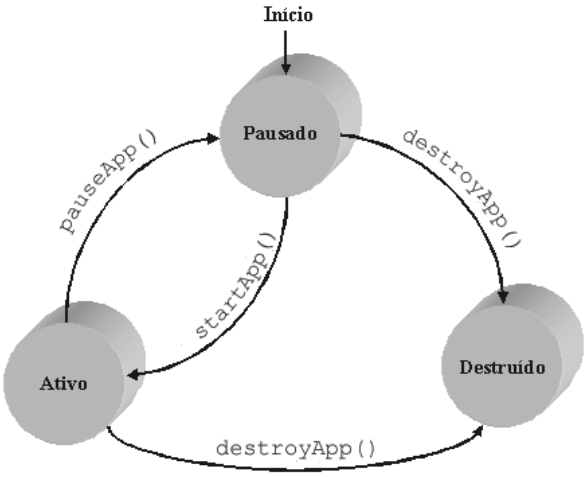


Figura 2: Ciclo de vida de un MIDlet

Estados de un MIDlet

Un MIDlet puede cambiar de estado mediante una llamada a los métodos *MIDlet.startApp(), MIDlet.pauseApp() o MIDlet.destroyApp()* .El gestor de aplicaciones (AMS) 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 realizan 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.

Como se dijo anteriormente los MIDlets derivan de la clase **MIDlet**, la cual está contenida dentro del paquete *javax.microedition.midlet*. Este paquete define las aplicaciones MIDP y su comportamiento con respecto al entorno de ejecución. En la Tabla siguiente podemos ver cuáles son las clases que están incluidas en este paquete:

Clases	Descripción
MIDlet	Aplicación MIDP.
*MIDlet*stateChangeException	Indica que el cambio de estado ha fallado.

A continuación veamos algunas de estas clases con algunos de sus métodos.

Clase MIDlet

Public abstract class MIDlet

Un MIDlet es una aplicación realizada usando el perfil MIDP. La aplicación debe extender a esta clase para que el AMS pueda gestionar sus estados y tener acceso a sus propiedades. El MIDlet puede por sí mismo realizar cambios de estado invocando a los métodos apropiados.

Algunos métodos de los que dispone esta clase son los siguientes:

- *protected MIDlet()*
  Constructor de clase sin argumentos. Si la llamada a este constructor falla, se lanzaría la excepción *SecurityException*.

- *public final intcheckPermission(String permiso)*
  Consigue el estado del permiso especificado. Este permiso está descrito en el atributo MIDlet-Permission del archivo JAD.

  Los valores devueltos por el método se corresponden con la siguiente descripción:

  - 0 si el permiso es denegado
  - 1 si el permiso es permitido
  - -1 si el estado es desconocido

- *protected abstract void destroyApp(boolean incondicional) throws MIDletstateChangeException*

  Indica la terminación del MIDlet y su paso al estado de "Destruido". En este estado el MIDlet debe liberar todos los recursos y salvar cualquier dato en el almacenamiento persistente que deba ser guardado.

- *public final String getAppProperty(String key)*

  Este método proporciona al MIDlet un mecanismo que le permite recuperar el valor de las propiedades desde el AMS. Las propiedades se consiguen por medio del archivo manifiesto y JAD.

- *public final void notify Destroyed()*

  Este método es utilizado por un MIDlet para indicar al AMS que ha entrado en el estado de "Destruido". Todos los recursos ocupados por el MIDlet deben ser liberados por éste de la misma forma que si se hubiera llamado al método MIDlet.destroyApp().

  Otros métodos:
  - public final void notifyPaused()
  - protected abstract void pauseApp()
  - public final Boolean platformRequest(String url)
  - public final void resumeRequest()
  - protected abstract void startApp() throws MIDletstateChangeException

Clase MIDletChangeStateException

public class MIDletstateChangeException extends Exception

Esta excepción es lanzada cuando ocurre un fallo en el cambio de estado de un MIDlet.

1.4.5 Paquetes para el manejo de Interfaces de usuario

Tanto MIDP1.0 como MIDP2.0 definen el mismo paquete para la creación de interfaces de usuario en el dispositivo móvil, no obstante, MIDP 2.0 incluye soporte para interfaces de usuario mejoradas que permiten la creación de aplicaciones más atractivas.

Conforme avancemos, se irán describiendo algunas de las clases que componen el paquete *javax.microedition.lcdui* y sus correspondientes métodos. El objetivo no es ser una guía de referencia, por lo que no vamos a explicar exhaustivamente cada clase y método.

Por un lado estudiaremos solo los componentes de la interfaz de usuario de alto nivel, ya que se consideran las más importantes en la creación y desarrollo de la aplicación de ejemplo.

Al usar estos elementos, perdemos control del aspecto de nuestra aplicación ya que la estética de estos componentes depende exclusivamente del dispositivo donde se ejecute. Usando API's de alto nivel se obtiene portabilidad de la misma aplicación entre distintos dispositivos.

Fundamentalmente, se usan estas API's cuando queremos construir aplicaciones de negocios.

Elementos de la Interfaz de Usuario de Alto Nivel "Especificación MIDP 2.0"

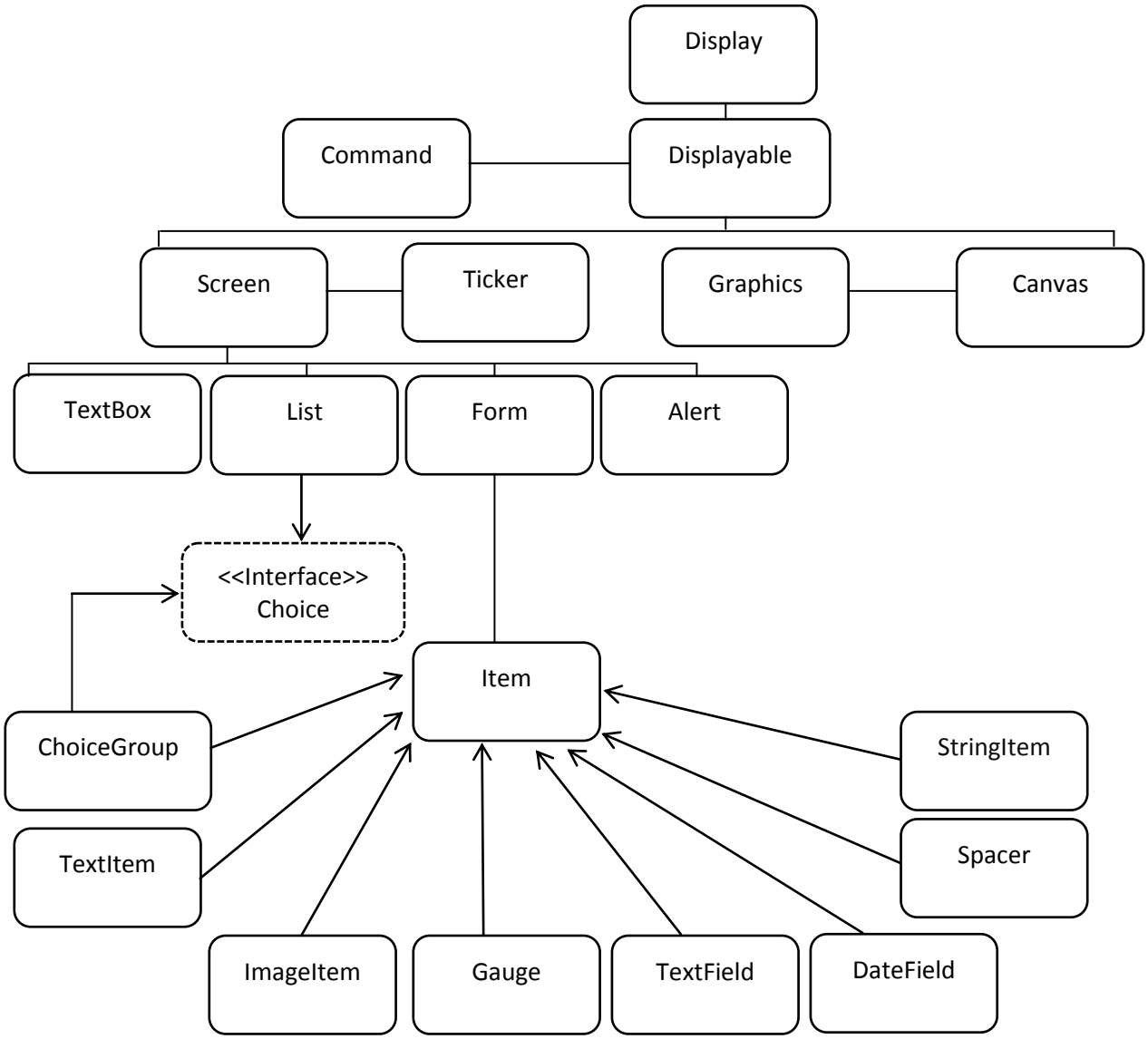


Figura 3: Interfaz de Usuario de Alto Nivel

- **Nuevas Interfaces**

  - ***ItemCommandListener:*** recibe notificaciones de comandos que han sido invocados a través de un objeto ***Item***, En otras palabras con las mismas dos teclas de comandos que proveen los dispositivos, podemos manejar múltiples comandos en pantalla.

- **Nuevas Clases**
  - ***CustomItem***: sirve para crear items personalizados a través de la herencia e introducir nuevos componentes visuales e interactivos, que no pertenecen al estándar, dentro de un Form.

  - ***Spacer***: representa un elemento no interactivo. Útil para ubicar espacios variables entre Items en una misma fila de un Form.

- **Modificaciones y Adiciones a las clases existentes**

  - ***Clase Alert:*** Clase que deriva de Screen, representa una pantalla de aviso. Normalmente se usa cuando queremos avisar al usuario de una situación especial como, por ejemplo, un error.
    Constructores:
      - *Alert(String título)*
      - *Alert(String título, String textoalerta, Image imagen, AlertType tipo)*

    Esta clase se declara de la siguiente manera: public class Alert extends Screen

    Nuevos Atributos
    static Command DISSMIS_COMMAND: un comando entregado a un Oyente para indicar que la alerta ha sido lanzada.

    Nuevos Métodos
    public Gauge getIndicator(): obtiene el indicador de actividad de un objeto *Alert.*

    public void RemoveCommand(Command cmd): remueve un comando que ha sido adicionado.

  - ***Clase Canvas:*** Es la clase base para crear aplicaciones que necesiten control sobre los eventos de bajo nivel y dibujar primitivas graficas sobre la pantalla.

    Nuevos Métodos
    setFullScreenMode(boolean mode): controla si el Canvas se encuentra en modo de pantalla completa o en modo normal.

sizeChanged(int w, int h): método que se llama cuando el área de dibujo del canvas ha sido cambiada. Ejemplo: al cambiar de modo normal a modo full screen.

- **Clase Command:** Esta clase encapsula información sobre una acción realizada por un usuario. Se declara de la siguiente manera: *public class Command.*
  Podemos pensar en él como un botón de Windows. Generalmente los implementaremos en nuestros MIDlets cuando queramos detectar y ejecutar una acción.

  Existen tres parámetros que hay que definir cuando construimos un objeto Command:

  - Etiqueta: cadena de texto que aparecerá en la pantalla del dispositivo que identificará a nuestro Command.
  - Tipo: Indica el tipo de objeto Command que queremos crear.
  - Prioridad: se asigna una prioridad específica aun objeto Command, a mayor numero, menor prioridad.

  Tipos de comandos definidos:

Tipos:	○ BACK: Petición para volver a la pantalla anterior ○ CANCEL: Petición para cancelar la acción en curso ○ EXIT: Petición para salir de la aplicación ○ HELP: Petición para mostrar información de ayuda ○ ITEM: Petición para introducir el comando en un "item" en la pantalla ○ OK: Aceptación de una acción por parte del usuario ○ SCREEN: Un comando que pertenece a la pantalla actual ○ STOP: Petición para parar una operación

  Nuevos Métodos
  Public command (String shortLabel, String longLabel, int commandType, int priority): es un constructor que crea un nuevo comando con las etiquetas, tipo y prioridad dadas.

  public String getLongLabel(): obtiene el longLabel de un Objeto Command.

- ***Clase Display:*** Representa el manejador de la pantalla y los dispositivos de entrada. Todo *MIDlet* debe poseer por lo menos un objeto Display. La clase Display puede obtener información sobre las características de la pantalla del dispositivo donde se ejecute el *MIDlet*, además de ser capaz de mostrar los objetos que componen nuestras interfaces.

  Para obtener una instancia se utiliza el siguiente método:
  Display pantalla = Display.getDisplay(this)

La llamada a este método la realizaremos dentro del constructor del MIDlet. De esta forma nos aseguramos que el objeto Display esté a nuestra disposición durante toda la ejecución de éste. Además, dentro del método startApp tendremos que hace referencia a la pantalla que queramos que esté activa haciendo uso del método setCurrent(). Hay que tener en cuenta que cada vez que salimos del método pauseApp, entramos en el método startApp, por lo que la construcción de las pantallas y demás elementos que formarán parte de nuestro MIDlet la tendremos que hacer en el método constructor.
Ej:

```
import javax.microedition.midlet.*
import javax.microedition.lcdui.*

public class MiMIDletextends MIDlet
{
Display pantalla;
Public MiMIDlet{
pantalla = Display.getDisplay(this);
// Construir las pantallas que vayamos a utilizar en el MIDlet,
}
Public startApp
{
if (pantalla == null)
pantalla.setCurrent(Displayable d);
// d tiene que ser un objeto que derive de la clase Displayable:
    // Form, Textbox, …
}
Public pauseApp{…}
Public destroyApp{…}
}
```

Nuevos Atributos

static int ALERT: tipo de imagen para la imagen de un elemento Alert.
static int LIST_ELEMENT: tipo de imagen para la imagen de un elemento List.
static int COLOR_BACKGROUND: Especifica el color de fondo de la pantalla.

Otros:

static int COLOR_BORDER:
static int COLOR_HIGHLIGHTED_BORDER:
static int COLOR_HIGHLIGHTED_BACKGROUND:
static int COLOR_HIGHLIGHTED_FOREGROUND:

Nuevos Métodos

public boolean flashBacklight(int duration): produce un efecto de flash en la pantalla durante la duración especificada en milisegundos, retorna true si el efecto puede ser controlado por la aplicación.
public void setCurrentItem(Item item): desplaza la pantalla hasta que sea posible ver y seleccionar el item que se ha pasado como argumento.

Otros:
public int getBestImageHeight(int imageType):
public int getBestImageWidth(int imageType):
public int getBorderStyle(Boolean highlighted):
public int getColor(int colorSpecifier):

- ***Clase Displayable*:** Representa a las pantallas de nuestra aplicación. Cada objeto Display puede tener tantos objetos Displayable como quiera. Mediante los métodos getCurrent y setCurrent controlamos qué pantalla queremos que sea visible y accesible en cada momento.

  Nuevos Métodos
  Public Ticker getTicker(): obtiene el objeto Ticker o null si no se ha fijado.
  Public void setTicker(Ticker ticker): fija un objeto Ticker para ser usado en el Displayable. Si se pasa null como parámetro se remueve el ticker actual.
  Public void setTitle(String s): fija el título del Displayable. Si se pasa null como parámetro se remueve el título actual.

- ***Clase Font*:** nos permite seleccionar el tipo de letra y almacenarlo en un objeto de este tipo para posteriormente usarlo en nuestro Canvas. El siguiente fragmento muestra cómocrear una fuente determinada:

  Font.getFont(int aspecto, int estilo, int tamaño) devuelve el objeto Font deseado.

```
Font fuente = Font.getFont(FACE_SYSTEM,STYLE_PLAIN,SIZE_MEDIUM);
```

  Una vez obtenido el objeto Font, deberemos asociarlo al objeto Graphics que es el que realmente puede escribir texto en pantalla.

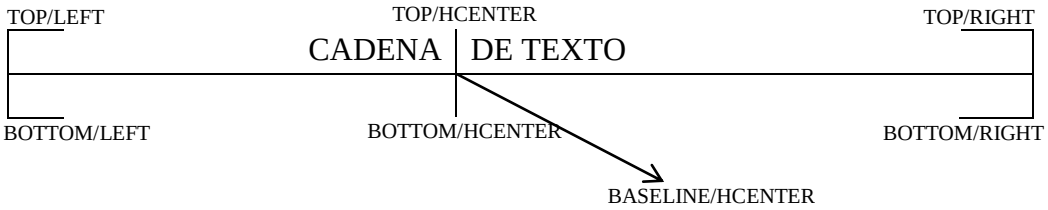
```
g.setFont(fuente); //Seleccionamos la fuente activa.

g.drawString("Cadena de texto", getWidth()/2, getHeight()/2,

BASELINE|HCENTER);
```

  Aquí observamos que seleccionamos una fuente con la que posteriormente escribimos la cadena de texto "Cadena de texto" posicionándola en el punto medio de la pantalla. El parámetro BASELINE|HCENTER indica el posicionamiento del texto con respecto al punto medio.

Nuevos Métodos
public static Font getFont(int fontSpecifier): obtiene la fuente usada por la interfaz de usuario de alto nivel para el especificador de fuente pasado como parámetro(FONT_INPUT_TEXT, o FONT_STATIC_TEXT).

Nuevos Atributos
public static final int FONT_INPUT_TEXT: especificador usado por la implementación para dibujar texto entrado por un usuario.
public static final int FONT_STATIC_TEXT: especificador usado para dibujar contenidos en objetos *Item* y *Screen*.

Posicionamiento del texto



- ***Clase Form:*** es un componente que actúa como contenedor de un número indeterminado de objetos. Esta clase deriva de Screen. Todos los objetos que puede contener un formulario (Form) derivan de la clase Item.

  La posición y el desplazamiento vertical de los ítems es controlada por el dispositivo de forma automática. Los desplazamientos horizontales no son convenientes en dispositivos con pantallas pequeñas. Por lo cual el desarrollador debe procurar, sea cual sea el dispositivo móvil en el cual se ejecute la aplicación, que la presentación no provoque un desplazamiento horizontal de los items.

  Manejo de los eventos:
  Es necesario implementar la interfaz *ItemStateListener* que contiene un solo método abstracto *itemStateChanged*(Item item).

  Cuando realizamos algún tipo de acción en un Item de un Form, ejecutamos el código asociado a ese Item que definamos en el método *itemStateChanged(Item item)* de igual forma que hacíamos con el método *commandAction(Command c, Displayable d)* cuando manejamos Commands.

  Nuevos Métodos
  public void deleteAll(): remueve todos los items del objeto *Form* que lo invoque.
  public int getHeight(): retorna la altura en pixels del área de despliegue disponible para items, este valor es el alto del *Form*.

public int getWidth(): retorna el ancho en pixels del área de despliegue disponible para items. El valor depende de cómo el dispositivo utiliza la pantalla.

- ***Clase Items:*** La clase Items es la superclase de todos los componentes que pueden ser incorporados a un objeto Form. Todos los objetos Items tienen una etiqueta que representa el título del objeto Items.

  Un objeto Items solo puede estar asociado a un Form, lo cual simplifica la implementación, al no tener que compartir el estado interno de los Objetos Items.

  La especificación MIDP 2.0 permite especificar el tamaño mínimo y preferido de los objetos Items, que en caso de no ser fijados explícitamente, son calculados automáticamente por el sistema. También permite indicar la dirección de cualquier objeto Items, no solamente de objetos ImageItem. Además permite indicar un comando por defecto que puede ser invocado desde la interfaz de usuario mediante las acciones que permita el dispositivo, ya sea como un botón especial o cualquier otro método. La especificación también nos permite indicar el comando por defecto mediante *setDefaultCommand().*

Componentes MIDP:

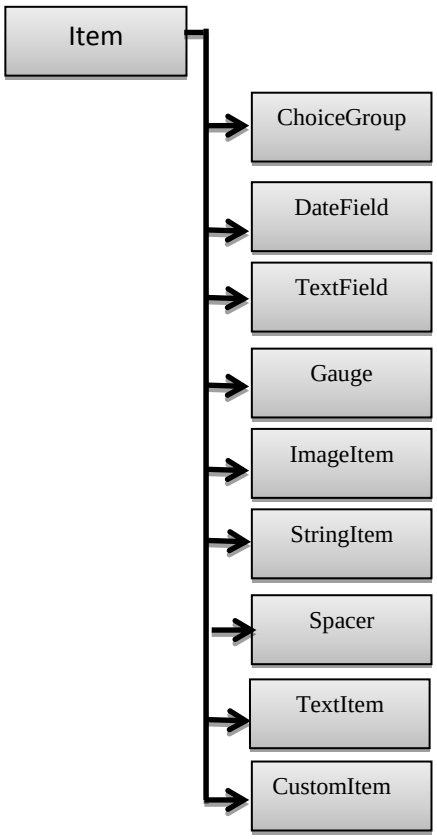


Figura 4: Componentes MIDP

Nuevos atributos
public static final int BUTTON: el Item aparecerá como un botón.
public static final int LAYOUT_BOTTOM: este Item puede tener un alineamiento inferior.
public static final int LAYOUT_CENTER: este Item puede tener un alineamiento centrado horizontalmente.
public static final int LAYOUT_DEFAULT: este Item debe seguir por defecto, las políticas de disposición de su contenedor.

Otros:
public static final int LAYOUT_EXPAND
public static final int LAYOUT_LEFT:
public static final int HYPERLINK:

Nuevos Métodos
public void addCommand(Command cmd): adiciona al item un comando sensitivo al contexto. Es ilegal llamar a este método si el item está contenido dentro de un Alert.
public int getMinimumHeight(): obtiene la mínima altura de este ítem.
*public int getMinimumWidth():*obtiene el ancho mínimo de este item.
*public int getPreferredHeight():*obtiene la altura ideal para este item.
public int getPreferredWidth(): obtiene el ancho ideal para este item.

Otros:
public int getLayout()
public void notifyStateChanged()
public void removeCommand(Command cmd)

- ***Clase Gauge*:** Implementa un indicador de progreso a través de un gráfico de barras. Representa un valor entero que va desde 0 hasta un valor máximo que definimos al crearlo:

  Gauge(String etiqueta, boolean interactivo, int valormax, int valorinicial)

  El constructor de Gauge permite 4 argumentos y crea una instancia Gauge con etiqueta, modo, valores máximo e inicial especificados.

  En el constructor del indicador se puede definir su funcionamiento como: Interactivo o No Interactivo.

  - En el modo Interactivo se permite al Usuario realizar acciones de aumento o disminución del valor representado por el indicador.
  - En el modo No Interactivo el Usuario no puede hacer modificaciones y se utiliza para mostrar el progreso de alguna operación.

Nuevos Atributos
public static final int CONTINUOUS_IDLE: representa el estado de marcha continua lenta de un objeto Gauge no interactivo con rango indefinido.

public static final int INCREMENTAL_IDLE: representa el estado incremental lento de un Gauge no interactivo con rango indefinido. En este estado el gauge muestra un gráfico indicando que no hay tareas en progreso.

public static final int INDEFINITE: indica que el Gauge tiene rango indefinido. Este valor puede ser usado como el parámetro maxValue en el constructor.

Otros:
public static final int CONTINUOUS_RUNNING
public static final int INCREMENTAL_UPDATING

- ***Clase Graphics:*** Este objeto nos proporciona la capacidad de dibujar en una pantalla Canvas. Un objeto Graphics lo podemos obtener sólo de dos maneras:

  1. Dentro del método paint() de la clase Canvas. Aquí podemos usar el objeto Graphics para pintar en la pantalla del dispositivo.

  2. A través de una imagen usando el siguiente código:
  Image imgtemp = Image.createImage(ancho,alto);
  Graphics g = imgtemp.getGraphics();

La clase Graphics posee multitud de métodos que nos permitirán seleccionar colores, dibujar texto, figuras geométricas, etc. Sin embargo vamos a ver algunos métodos nuevos.

Nuevos Métodos
*public void drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor):*copia una región de una imagen específica a una ubicación de destino, permite aplicar efectos de transformación como rotación y reflejado.

public void fillTriangle(int x1,int y1,int x2,int y2,int x3,int y3): rellena el triángulo especificado con el color actual, las líneas que conectan cada par de puntos son incluidas en el triángulo relleno.

public int getDisplayColor(int color): obtiene el color que será desplegado si un color específico es requerido. Este método permite al desarrollador revisar la manera en la cual los valores RGB son mapeados al conjunto de colores distintos que el dispositivo puede desplegar realmente.

- ***Clase Image***: Representa una imagen, su existencia es independiente del dispositivo y no es visualizada hasta que se invoca de forma explícita a paint() de la clase Canvas o al ser incorporado a un objeto Form o Alert.

  public class image extendsObject{…}. Un objeto de este tipo puede incorporarse a un objeto Alert, Choice o ImageItem. El formato soportado por MIDP es PNG.

  Las imágenes pueden ser:

  - Mutables: Creadas conteniendo solo pixeles en blanco y la aplicación puede dibujar en ella por medio de un objeto Graphics obtenido con el método getGraphics();
  - Inmutables: Creadas al cargarse de algún recurso, de archivos o desde conexiones externas.

  Nuevos Métodos
  public static Image createImage(Image image, int x,int y, int with, int height, int transform): Crea una imagen inmutable usando como datos los pixels de una determinada imagen fuente y transformándola de la manera especificada.

  public static Image createImage (java.io.InputStreamstream) throws java.io.IOException: Crea una imagen inmutable a través de la decodificación de datos de imagen obtenidos desde un InputStream.

- ***Clase ChoiceGroup:*** Define un grupo de elementos seleccionables que pueden incluirse en un objeto Form. Soporta los modos de selección Exclusive y Múltiple.

  Para construir un objeto ChoiceGroup realizaremos una llamada a su constructor con los siguientes parámetros:

  - *ChoiceGroup(String etiqueta, int tipo)*
  - *ChoiceGroup(String etiq, int tipo, String[] elementos, Image[] imagenes)*

  ChoiceGroup a diferencia de Lista, puede compartir pantalla con cualquier número de componentes dentro de los límites de memoria del dispositivo. El dispositivo es el responsable de adecuar la apariencia visual del objeto ChoiceGroup en pantalla al modo de selección que le corresponda.

  Tipos de ChoiceGroup: Para los modos de selección Múltiple y Exclusivo respectivamente.

  1. Cajas de Selección
  2. Botones de Radio

En la especificación MIDP 2.0 incluye un nuevo tipo para la apariencia del objeto ChoiceGroup llamado “POPUP”. La apariencia de los objetos de este tipo es de cajas combos. Es decir, muestran uno de los elementos de la lista de selección y un botón para indicar que existen más opciones. Seleccionando este botón aparecerá la lista completa de los elementos seleccionables, permitiendo una nueva selección por el usuario.
Ej.:

//Lista de opciones:
opciones[0]=”Sonido”
opciones[1]=”Video”
opciones[2]=”Seguridad”

//Creamos la selección en base a los arrays
Preferencias = new ChoiceGroup(“Selecciona una opción:”, Choice.POPUP, opciones, imagenes);
Preferencia.setSelectedIndex(2,true);

Manejo de Eventos:

- *itemStateChanged().* Cuándo un usuario selecciona una opción, el formulario registra unItemStateListener y se realiza una llamada al método anterior. En este método podemos ver qué elemento ha sido seleccionado y cuál no y realizar las acciones oportunas. Hay que tener en cuenta que cada vez que cambiemos nuestra selección, realizaremos una llamada a este método.

- *commandAction().* En este caso podemos añadir un componente Command al formulario que nos valide nuestra selección de opciones.

Nuevos Métodos
Public void deleteAll(): borra todos los elementos de un objeto ChoiseGroup.

public Font getFont(int elementNum): devuelve la fuente del elemento elementNum (índice del elemento empezando en cero).

public void setFont(int elementNum, Font font): fija la fuente preferida de la aplicación para ser aplicada a un elemento específico de un Objeto Choice. El parámetro font puede ser un objeto Font válido o puede ser null, en el segundo caso, es usado el valor por defecto para ser aplicado a un elemento.

- ***Clase TextField:*** Es un componente que permite editar texto y que puede ser incorporado a un objeto Form. Este objeto tiene un tamaño máximo, una serie de constantes para indicar el contenido que admite, una etiqueta y un valor inicial.

  Para crear un TextField sólo hay que invocar al constructor con los siguientes parámetros: *TextField(String etiqueta, String texto, int capacidad, int restricciones).*

  Tipos de contenido que admite:
  - ANY
  - NUMERIC
  - PASSWORD
  - PHONENUMBER
  - URL
  - EMAILADDR

  Cada tipo define un conjunto de caracteres específicos válidos para dicha entrada.

  Las diferencias entre TextBox con respecto al objeto TextFieldson:

  - Un TextField tiene que ser insertado en un formulario, mientras que un TextBox puede implementarse por sí mismo.

  - TextField deriva de la clase Item, mientras que TextBox deriva directamente de Screen, y sus eventos los controlamos a través de Commands. Los eventos que produce un TextField los controlamos a través del método itemStateChanged(Item item),.

  Sin embargo, ambas clases comparten las restricciones de entrada que están definidas por los tipos de contenido que admiten.

  Nuevos atributos
  public static final int DECIMAL: permite al usuario ingresar valores numéricos con fracciones decimales opcionales, por ejemplo "-123", "0.123", o ".5".

  public static final int SENSITIVE: indica que el texto ingresado es un conjunto de datos que la implementación nunca debería almacenar en un diccionario o tabla para un uso predictivo o cualquier otro esquema acelerado de entrada.

  Nuevos Métodos
  *public void setInitialInputMode(String characterSubset):*fija el modo de entrada a utilizar por la aplicación cuando el usuario inicie a editar este TextField.

- ***Clase DateField***

Este componente nos permite manejar fechas y horas en nuestro Objeto Form. Para ello, hace uso de la clase java.util.Date ya que es con este objeto con el que trabaja.

Un objeto DateField puede configurarse para aceptar una fecha, hora o ambas cosas a la vez. Si se indica el modo DATE solamente admitirá fechas, si el modo indicado es TIME solamente permitirá la utilización de horas y minutos y si se indica el modo DATE_TIME, permitirá el uso de fecha y hora.

Como crear un objeto DateField:

```
fPedido = new DateField("Fecha toma pedido:      ",DateField.DATE);
fPedido.setDate(new Date());// Creo y Establezco un objeto Date con fecha actual
```

Para dar formato a una fecha J2ME no posee una clase a parte de DateField. Para presentar correctamente fecha y hora DateField necesita conocer la zona horaria local. El constructor de este objeto que recibe dos parámetros utiliza la zona horaria del dispositivo definida por defecto. Mientras que el constructor que permite indicar un tercer parámetro, permite indicar en él una zona horaria distinta, en caso de que la definida por defecto no sea la correcta. No se puede cambiar la zona horaria sin crear una nueva instancia de DateField.

```
String formatoFecha(DateFiel ddf)
  {
      int dia, mes; Calendar hoy;
      hoy = Calendar.getInstance();
      Date ahora = df.getDate();
      hoy.setTime(ahora);

      dia = hoy.get(Calendar.DAY_OF_MONTH);
      mes = hoy.get(Calendar.MONTH)+1;

      String sDia = ""; String sMes = "";
      if( dia< 10)
        sDia = "0";
        sDia+=dia;
      if(mes < 10)
        sMes="0";
        sMes+=mes;
        String fechaFormato = sDia+"/"+sMes+"/"+hoy.get(Calendar.YEAR);
      return fechaFormato;
   }
```

RMS (Record Management System)

MIDP define un conjunto de clases destinadas al control del almacenamiento de datos de forma que sea posible guardar y recuperar información en el propio dispositivo móvil.

RMS es un sistema semejante a una base de datos propia del dispositivo móvil común a todas las aplicaciones.

Los RMS permiten que los datos puedan mantenerse a través de distintas ejecuciones de un midlets o que distintos midlets de una misma aplicación puedan compartir la misma zona de datos dentro de RMS.

Es un sistema orientado a registro, representa una base de datos que cada fila del tiene un identificador único y una serie de bytes que representan el contenido o información.

Registros	Datos
1	Datos
2	Datos
…	…

En la parte de datos, es donde se coloca el contenido de los registros. Cada zona RMS de almacenamiento debe estar identificada mediante un nombre único el cual no debe sobrepasar los 32 caracteres y el sistema es sensible a minúsculas y mayúsculas.

Ej: static final String ID_RMS = “pedidos”;

Las zonas de almacenamientos son en realidad archivos binarios dependientes de la plataforma, porque están creadas en zonas restringidas al uso del dispositivo y que no solamente está disponible para los midlets sino también para almacenar mensajes cortos que recibe o ha enviado un teléfono celular.

La implementación RMS asegura que todos las operaciones que se realicen sobre una zona de almacenamiento sean sincronizadas y serializadas sin que se produzca corrupción alguna por los muchos accesos simultáneos. Sin embargo si los midlets utilizados por aplicaciones externas al propio RMS usan multitareas y se les proporciona acceso a la zona de almacenamiento, es responsabilidad del programador la sincronización de esos accesos.

En los RMS cada bloque de datos es un registro dentro de la zona de almacenamiento. Estos bloques están relacionados entre sí mediante el modelo de listas enlazadas, de forma que cada registro mantiene un enlace al siguiente registro.

La cabecera también tiene un enlace al primer bloque de datos y otro enlace al primer bloque de la zona de espacio libre. Además puede contener lo siguiente:

- El número de registros de la zona de almacenamientos. El valor inicial es cero incrementándose o disminuyendo en 1 cuando se añade o elimina un registro.
- El número de versión que inicialmente es cero y que también incrementa según la implementación, aunque normalmente es 1, puede ser cualquier número positivo mayor que cero.
- Fecha ultima de modificación.
- El identificador que se asignara al siguiente registro. Este valor se incrementa en 1.

La cantidad en byte sobre el espacio disponible en la zona de almacenamiento no está contenida dentro de la cabecera que se acaba de describir. Esta información se puede obtener mediante la llamada al método: getSizeAvailable() aunque la información no es muy fiable ya que la zona de almacenamiento también es ocupada para almacenar información de aplicaciones nativas del dispositivo.

Como descubrimos un RecorStore almacenado en un dispositivo.

De nuestra MIDlet podemos obtener una lista de todos los record stores que almacena la aplicación, utilizando esta función RecordStore.listRecordStores(), esta función retorna una lista de string que contiene el nombre de los record stores que existen en laMIDlet.

```
Public void analizarTodosRMS()
{
String[] nombres = RecordStores.listRecordStores();
for(int i=0; nombre != null&nombres.length; i++)
{
System.out.println(name[i]);
}
}
```

Muestra por consola esta función imprime los nombres de los record store almacenados en la MIDlet.

Como Abrimos y Cerramos un record store

El método RecordStore.openRecordStore(), se utiliza para abrir y opcionalmente crear un record store, pues de lógica este método retorna un objeto RecordStore. El primer parámetro de entrada es el nombre del record store, y el segundo es un dato tipo boolean (true y false), para indicarle si creamos un record store, si este no existe.

Cuando termine con un record store, se debe cerrar con el llamado del método RecordStore.closeRecordStore().

Miremos el ejemplo:

```
Public void analizar( String rsNombre )
{
RecordStorers = null;
try {
        rs = RecordStore.openRecordStore( rsNombre, false );
} catch( RecordStoreException e )
{
                //nombre de record store muy largo
                logger.exception(rsNombre, e );
} finally{
        try {
                rs.closeRecordStore();
        } catch( RecordStoreException e ){}
}
}
```

Adicionando un registro

Para adicionar un registro utilizamos el método RecordStore.addRecord(), este recibe 3parámetros, el primero es el dato aguardar, el segundo el inicio de donde vamos a empezar aguardar (comienzo de offset), el tercero es el total de bytes a almacenar (final del offset), el método retorna un dato entero que es el identificador del registro en donde se almaceno los datos.

```
// creamos un record store
RecordStorers = null;
try {
        rs = RecordStore.openRecordStore( "rsDatos", true );
}
catch( RecordStoreException e ){// no se puede crear o abrir}
byte[] datos = new byte[]{ 0, 1, 2, 3 };
int registroID;
registroID = rs.addRecord( registro, 0, datos.length );
```

Actualizar o modificar un registro

Para realizar esta operación utilizamos el método RecordStore.setRecord(), este método recibe 4 parámetros, el primero es el identificador del registro (el que retorna rs.addRecord()),el segundo parámetro es el objeto que se va aguardar, el tercero y cuarto, es el inicio y final de lo que vamos a guardar, continuando con el ejemplo anterior tenemos.

```
byte[] nuevoDato = new byte[]{ 0, 10, 20, 30 };
// remplaza el registro con valores 10 y 20
rs.setRecord(registroID, nuevoDato, 1, 2 );
//o podemos reemplazarlo por completo
rs.setRecord(registroID, nuevoDato, 0, nuevoDato.length);
```

Lectura de registros de un record store

Para leer de un record store utilizamos la función RecordStore.getRecord(), recibe como parámetro el identificador de registro que se va a leer y nos retornara un array de bytes.byte[]

```
dato = rs.getRecord( registroID );
```

Otra forma para cargar un registro seria esta:

```
byte[] dato = new byte[ rs.getRecordSize( registroID ) ];
rs.getRecord(registroID, datos, 0 );
```

La función getRecordSize(int identificador), nos retorna el tamaño del array debytes del registro especificado por el identificador, como la función getRecord está sobre cargada, esta recibe tres parámetros, el primero es el identificador de registro, el segundo es el array de bytes en donde se cargara la información del registro, y el tercero es el comienzo de donde se empezara a cargar el array.

Recorriendo varios registros en un record store

Para recorrer todos los registros de un record store, primero debemos obtener el número total de registros, esto lo hacemos por medio del método getNextRecordID(), el cual retorna un dato entero con un valor del total de registros almacenados en el record store. Luego debemos obtener el tamaño de un registro en específico, esto lo hacemos con el método getRecordSize(int id), el cual recibe como parámetro el identificador de un registro y retorna el tamaño del registro, y por ultimo obtenemos el registro cargándolo en un array de bytes, pues fácil, veámoslo en un ejemplo.

```
Int numRegistrosID = rs.getNextRecordID(); //obtenemos el número de registros
byte[] dato = null;

for( int id = 0; id <numRegistrosID; ++id )
{//recorremos cada uno de los registros
try {
int size = rs.getRecordSize( id ); //obtenemos el tamaño de un registro
if( dato == null || dato.length<size ){
dato = new byte[ size ];
}
rs.getRecord( id, dato, 0 ); //obtenemos los datos del registro
}
catch( InvalidRecordIDException e ){//no hay record}
catch( RecordStoreException e ){
handleError( rs, id, e ); // call an error routine}
}
```

Borrando un Registro o un record store

Para eliminar un registro de un record store utilizamos el método RecordStore.deleteRecord(int id), recibe como parámetro el identificador del registro a eliminar, el record debe existir si no se genera una excepción InvalidRecordIDException.

Para borrar un record store utilizamos RecordStore.deleteRecordStore(), el cual recibe como parámetro el nombre del record store a eliminar.

```
try
{
        RecordStore.deleteRecordStore( "rsDatos" );
} catch( RecordStoreNotFoundException e ){// no se encuentra el record store}
  catch( RecordStoreException e ){// algo paso}
...
```

Interfaces, Clases y Excepciones que pertenecen al paquete RMS.

RecordComparator**:** Interfaz que define un comparador para poder comparar dos registros.

RecordEnumarator: Interfaz que representa un Enumerador mediante el cual podemos recorrer la zona RMS.

RecordFilter: Interfaz que define un filtro para examinar un registro y comprar si satisface un criterio definido por la aplicación.

RecordListener: Interfaz que se define los eventos que se reciben cuando en una zona de almacenamiento se añade, cambian o eliminan registros.

RecordStore: Representa una zona de almacenamiento individual.

InvalidRecordIDException: Excepción que se lanza cuando el identificador del registro que se indique no es valido.

RecordStoreFullException: Excepción que se lanza cuando esta full la zona de almacenamiento.

RecordStoreNotFoundException: Excepción que se lanza para indicar que en la zona de almacenamiento no se encuentra el registro.

RecordStoreNotOpenException: Excepción que se lanza cuando se producen un error al tratar de cerrar la zona de almacenamiento.

Capítulo 2

2.1 Conectividad a la Red

La conectividad a la red es uno de los pilares de J2ME, aunque un dispositivo puede ser usado sin conexión a la red, la capacidad de conexión es la característica que da verdadero poder al uso de estos dispositivos, ya que les proporciona entre otras cosas la capacidad de acceso a Servidores de Aplicaciones, Bases de Datos, Internet e Intranets Corporativas.

Aunque el desarrollar una aplicación para dispositivos móviles no es nada nuevo, la posibilidad de utilizar Java que ofrece J2ME, sí que es una característica que da un giro enorme al uso de los dispositivos móviles. Además J2ME hace que la misma aplicación se ejecute en diferentes dispositivos, lo que favorece la expansión y desarrollo de nuevas aplicaciones.

Esta capacidad de conexión a red que nos proporciona J2ME, complementa otras tecnologías como Bluetooth o WiFi, que permiten conexiones inalámbricas a los dispositivos.

Un ejemplo práctico funcional de una aplicación J2ME, en un entorno de negocio, es aquel en el que un trabajador desde un dispositivo móvil accede a la red corporativa a través de HTTP. La aplicación que se mostrará como ejemplo más adelante, es una emulación la cual describe el entorno de negocio a continuación planteado:

En la empresa un servidor de aplicaciones o una pasarela inalámbrica proporciona internet al dispositivo, de forma que el trabajador pueda acceder a los servidores y servicios corporativos. La figura siguiente esquematiza la situación descrita.

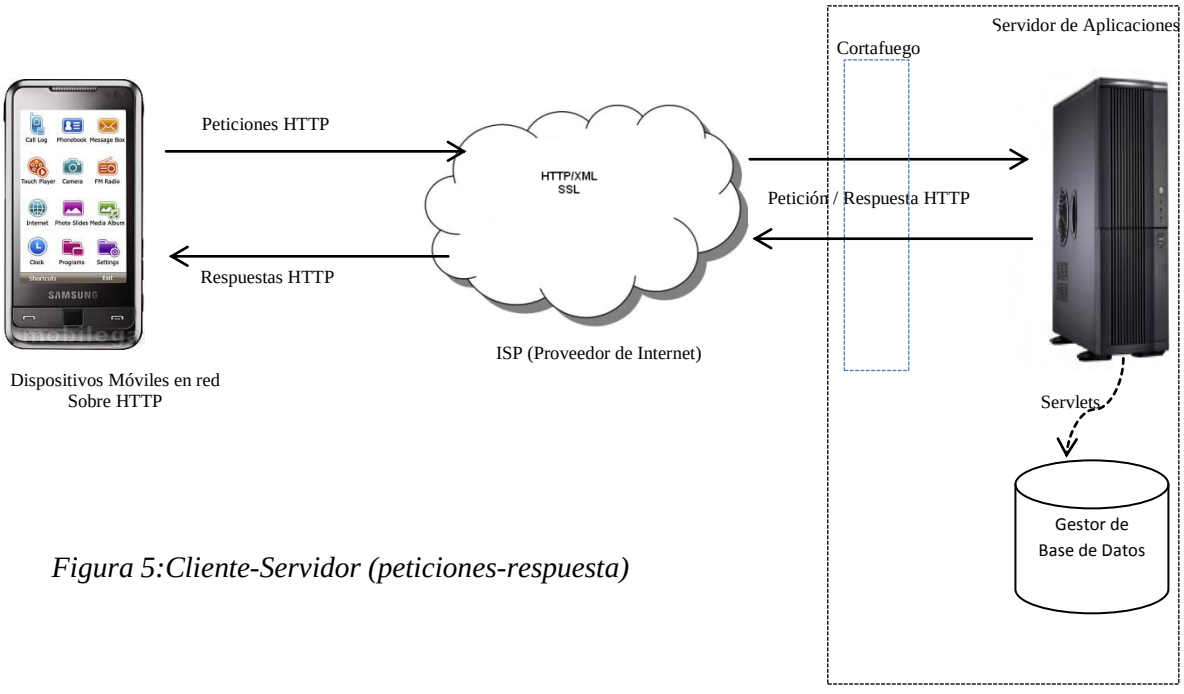


Figura 5:Cliente-Servidor (peticiones-respuesta)

El lector puede deducir de la descripción anterior que la creación de una aplicación inalámbrica con un componente de red no es algo simple. Sin embargo, la parte cliente implementada sobre dispositivos móviles requiere un diseño y una estructura, tanto de la aplicación, como de la lógica de control de la conexión a la red.

La parte del servidor tal y como muestra la figura, consiste en un servlet que procesa las peticiones y generan las respuestas, empleado base de datos como respaldo.

Al igual que en el caso de los Midlets hay partes de la aplicación del servidor que son muy semejantes entre distintas aplicaciones; por ejemplo la forma de leer las peticiones o de escribir las respuestas.

La comunicación entre cliente y servidor en las aplicaciones móviles suele utilizarse HTTP como protocolo por su flexibilidad, por estar universalmente disponible y por estar integrado en casi todos los dispositivos.

A continuación en este capítulo se aborda parte del API que proporciona capacidad de comunicación con las aplicaciones.

Jerarquía de interfaces MIDP2.0

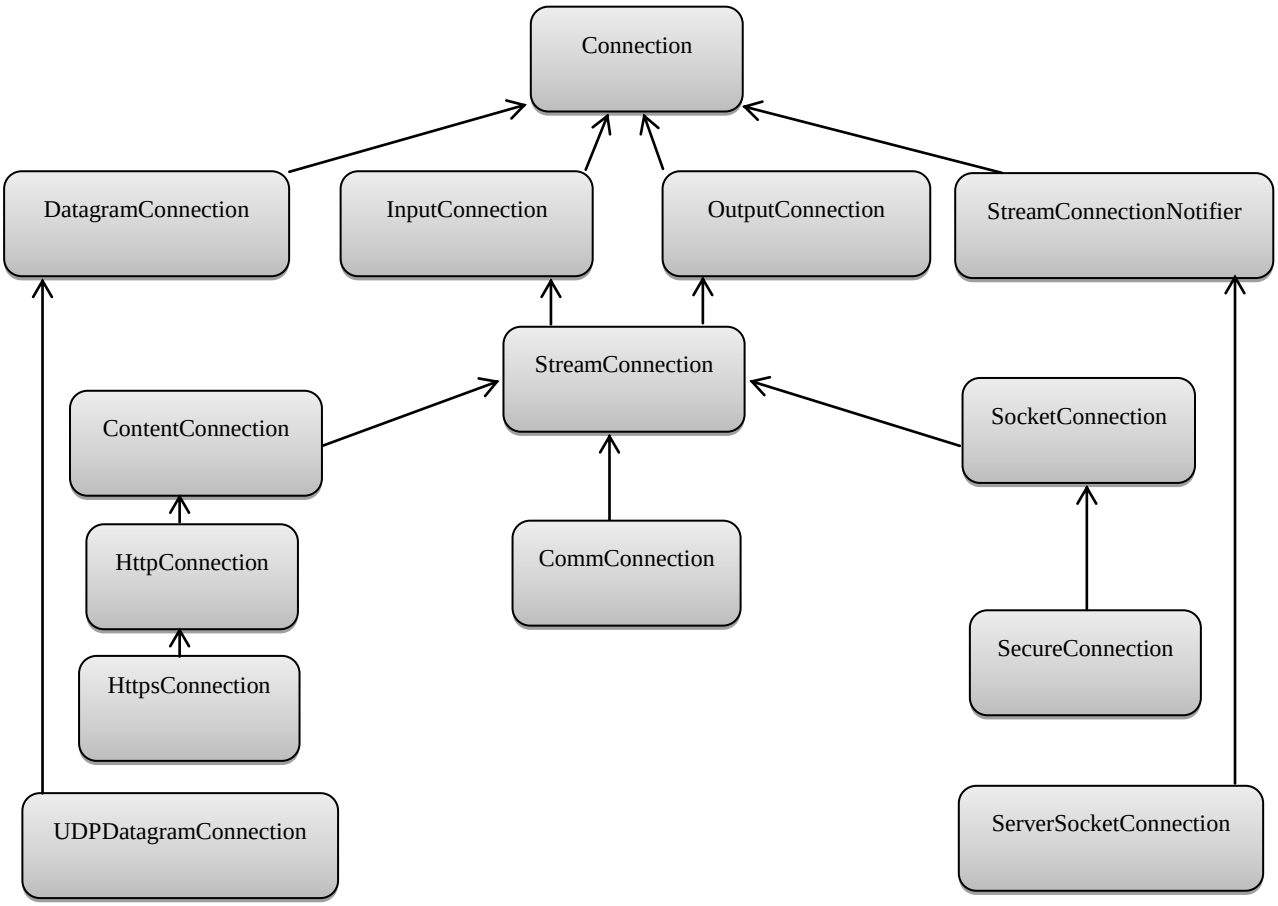


Figura 6: Interfaces MIDP 2.0

Estas clases orientadas a la conexión en red y comunicaciones reciben el nombre de GenericConnectionFramwork. Como vemos, la raíz del árbol es la interfaz Connection que representa la conexión más genérica y abstracta que podemos crear. El resto de interfaces que derivan de Connection representan los distintos tipos de conexiones que podemos crear.

Clases y conexiones del GenericConnectionFramework

Clase Connector

La clase Connector se utiliza para crear una instancia de un determinado protocolo de conexión a través de uno de sus métodos estáticos. La instancia devuelta es una implementación que soporta la interfaz Connection o uno de sus descendientes. En otras palabras la clase Connector no está pensada para ser instanciada sino para ser utilizada en la creación de un protocolo de conexión.

La conexión se realiza mediante el siguiente mensaje genérico:
Connector.open("protocolo:dirección;parámetros");

Donde:

protocolo	Indica la forma en que se establece la conexión (socket, file, http, datagrama, etc).
dirección	Indica el sitio con el cual quiere establecer la conexión, ya sea una dirección, un fichero o un puerto de comunicación.
parámetros	Cuando son necesarios, se indican como parejas de valores.

url = "http://yahoo.com/index.html
Ejemplo de invocación: (HttpConnection)Connector.open(url).

La clase Connector se encarga de buscar la clase específica que implemente el protocolo requerido. Si esta clase se encuentra, el método open() devuelve un objeto que implementa la interfaz Connection. La forma más sencilla de realizar una petición es utilizar el método open() de la clase Connector, se pasa como argumento una URL y se moldea el resultado a un objeto HttpConnection.

Interfaz Connection

Una conexión de tipo Connection se crea después de que un objeto Connector invoque al método open(). Como sabemos, esta interfaz representa a la conexión más abstracta y genérica posible. Por esta razón, el único método que posee esta interfaz es el de cierre de conexión.

```
Public interface Connection
{
        public void close()throws IOEception;
}
```

Interfaz InputConnection y OutputConnection

Ambas representan conexiónes basada en flujos de entrada/salida respectivamente. InputConnection sólo posee dos métodos al igual que OutputConnection, los cuales devuelven objetos de tipo InputStreams/OutputStreams correspondientemente.

Los métodos que proporciona InputConnection son:

1. DataInputStream openDataInputStream(): devuelve un flujo del cual se leen datos como tipos primitivos.
2. InputStream openInputStream(): devuelve un objeto el cual representa un flujo de entrada.

Los métodos que proporciona OutputConnection son:

1. DataOutputStream openDataOutputStream(): devuelve un flujo que escribe datos en tipos primitivos en un flujo.
2. OutputStream openOutputStream(): devuelve un objeto el cual representa un flujo de salida.

En nuestro caso haremos uso del objeto InputStream como flujo de entrada para la lectura de datos.

Ejemplo de conexiónInputStream:

```
HttpConnection con = (HttpConnection)Connector.open(url);
//...
con.setRequestMethod(HttpConnection.POST);
InputStream in = con.openInputStream();
int i;
while((i=in.read())!=-1)
{
baos.write(i);
if(i=='\n')
{
v.addElement(respuesta = new String(baos.toByteArray()));
baos.reset();
        }
      }
//...
```

Interface HttpConnection:

Es una subinterface de ContentConnection y define todos los métodos que permiten realizar peticiones HTTP y procesar las respuestas. Métodos tales como: setRequestMethod() y openInputStream() entre otros, proporcionando la funcionalidad necesaria para establecer cabeceras de petición/respuestas, así como funciones específicas HTTP.
HTTP es un protocolo de funcionamiento petición/respuesta, en el cual los parámetros de la petición deben estar fijados antes de realizar una petición.

Comunicación HTTP

Como dijimos anteriormente el funcionamiento del protocolo HTTP es el siguiente: Un Cliente realiza una petición al Servidor y espera a que éste le envié una respuesta. En nuestra aplicación de ejemplo, esta comunicación la vamos a implementar entre un Midlet y un Servlet, que recibirá las peticiones hechas por el cliente y dependiendo del caso nos devolverá un resultado.

La conexión tiene 3 estados:

1. Setup (Preparada): no se realiza ninguna conexión con el servidor, es donde se establecen los parámetros de comunicación. El cliente prepara la petición que va a realizar al servidor.

   Métodos que solo pueden invocarse en este estado:

   1) setRequestMethod(): indica si la petición va ser GET o POST
   2) setRequestProperty(): indica propiedades y valores de la petición.

   Ej: conn.setRequestMethod(HttpConnection.POST);
   setRequestProperty(“User-Agent”, “Profile/MIDP-1.0 Configuration/CLDC-1.0”)

2. Connected (Conectada): pasa a este estado cuando se invoca cualquier método que requiera enviar/recibir datos del servidor.

   Ej: openInputStream(), getLenght(), etc.

   Estos métodos y muchos otros heredan de la interface ContentConnection y sus super interfaces.

3. Closed (Cerrada). Cuando se llama al método closed(), para finalizar la conexión.

2.2 Comunicación y Configuración Cliente –Servidor (Aplicación Ejemplo)

La mayoría de las aplicaciones MIDP utilizan el paradigma cliente – servidor para realizar operaciones más complejas que las que permiten las limitadas capacidades de los dispositivos móviles. Como la especificación MIDP solamente impone la implementación del protocolo HTTP, dejando la implementación de otros protocolos a decisión de los fabricantes, todas las comunicaciones cliente – servidor deben realizarse a través de este protocolo.

En la implementación del Servlet de la aplicación, se ha utilizado como servidor para soporte de servlets el motor apache tomcat 6.0.20.En la clase del Midlet se utiliza el método POST a la hora de transmitir datos para la autentificación con el servidor. POST nos permite que se incluya cualquier tipo de dato, incluso binario sin restringirse a ningún formato determinado o a una longitud máxima. Para realizar una petición POST utilizamos la clase HttpConnection de la cual hay que crear una instancia e invocar a su método setRequestMethod().

```
String url =
"http://localhost:8084/ServletConnectionMysql/ServletConnectionMysql?u="+user.getString()+"&p="
+passwd.getString()+"&op="+0;
con = (HttpConnection)Connector.open(url);
con.setRequestMethod(HttpConnection.POST);
```

Es importante recordar que se debe realizar la llamada al método setRequestMethod(), antes de obtener los canales de entrada o salida, porque si no la llamada no tendrá efecto alguno. Una vez establecido el método de petición es necesario configurar los elementos de la cabecera HTTP. La especificación MIDP indica que es necesario establecer los atributos User-Agent y Current-Language.

```
setRequestProperty(“User-Agent”, “Profile/MIDP-1.0 Configuration/CLDC-1.0”)
setRequestProperty(“Current-Language”, “es-ES”)
```

En el lado del servidor las cosas son un poco más simples. El servlet recibe los parámetros, establece el tipo de contenido al midlet y abre un canal de salida para poder responder según el caso o petición.

```
response.setContentType("text/vnd.wap.wml");
printWriter escribir = response.getWriter();
```

Por su puesto en caso de error el servlet lo captura y no envía ningún dato como respuesta. En cualquier otro caso, el servlet fija la longitud del contenido a enviar si la sabe y envía los datos.

```
response.setContentLength(codCli.length());
escribir = response.getWriter();
escribir.println(codCli);
```

Procedimiento para ejecutar correctamente un Servlet que accede a una base de datos "MySQL" utilizando como herramienta de desarrollo Netbeans IDE 6.8.

Configuraciones:

1. Instalar el Gestor de Base de Datos de MySQL. El archivo tiene el nombre mysql-essential-6.0.10-alpha-win32. Tener presente la cuenta de usuario administrador "root" y la contraseña correspondiente (en este caso "raíz").

2. Instalar mysql-essential-6.0.10-alpha-win32 para la administración visual de MySQL..

3. Instalar en la maquina servidor jdk-6u17-javafx-1_2_1-windows-i586.

4. Instalar el SDK sun_java_me_sdk-3_0-win.

5. Instalar el Software Netbeans IDE 6.8, en el cual viene Apache Tomcat 6.0.20 como servicio, en la máquina que funcionara como "Servidor". En este caso este software es el que nos permitirá ejecutar el servlet cuando sea llamado por el cliente. Para caso de demostración se prueba localmente.

6. Instalar el Driver JDBC para MySQL. El nombre del driver es mysql-connector-java-5.1.10-bin.jar. Este driver debe quedar ubicado en C:\Archivos de programa\Java\jdk1.6.0_17\jre\lib\ext\ .

7. Crear la base de datos en MySQL que será accedida desde el servlet. En nuestro caso la BD se llamara "syspedidos". A continuación crear todas las tablas pertinentes para el funcionamiento de la BD.

8. En Netbeans Agregar una nueva conexión en el servicio de Databases y configurar Driver name, Host, Port, Database, Username, Password. Una vez configurado solo queda autenticarse para mostrar la BD y realizar operaciones.

9. En Netbeans lanzar Apache Tomcat 6.0.20 en el servicio Servers que escucha peticiones de los clientes que solicitan la ejecución del servlet. Cada vez que se modifique el código del servlet lo mejor es detener el servicio y volverlo a lanzar para que los cambios surtan efectos.

Capítulo 3

3.1 Autenticación

En cualquier comunicación a través de la red, siempre hay dos partes. Un cliente que inicia la comunicación y un servidor que recibe la petición de establecer esa comunicación con el cliente. La autenticación para darse a conocer uno al otro, puede por tanto realizarse en cualquiera de las dos partes. La autenticación del servidor se refiere a que este reciba parámetros confidenciales del cliente y el servlet los valide. Esta será la forma de autentificación que la aplicación ejemplo utilizará.

En la aplicación de ejemplo el cliente proporciona un identificador y una contraseña, asumiendo que solo el cliente conoce la contraseña, el servidor captura la información del cliente(midlet) y comprueba en una base de datos MySQL y si es correcta el cliente es autenticado.

En este caso hay dos problemas:

Por un lado generalmente los usuarios finales no controlan demasiado las contraseñas, tienden a olvidarlas, o anotarlas en lugares no seguros o a elegir contraseñas muy sencillas, fáciles de atacar. Lo cual podría resolverse realizando la autentificación en el lado del cliente, almacenando los datos en el propio dispositivo, para evitar que el usuario tenga que teclearlos en cada una de las ocasiones en que se requiera.

Por otro lado está el envió de la contraseña al servidor como texto claro, aunque esto se puede evitar enviando la contraseña mediante conexiones seguras HTTPS o enviándolas previamente encriptadas, sin embargo los procesos de encriptación son lentos y una espera de varios segundos en un ordenador es admisible, pero en un dispositivo móvil resulta del todo inadmisible.

No obstante la opción más segura entre cliente – servidor es la implementación de certificados, aunque es necesario aclarar que la especificación MIDP no incluye un API para la firma de código, lo cual obliga a utilizar herramientas de terceros y dado al tamaño reducido en cuanto a recursos de los dispositivos móviles, se puede hacer complicada su utilización.

Sin embargo lo primero que debe tener en mente todo usuario/lector es que la máxima de que la seguridad nunca es absoluta; hay sistemas más seguros y sistemas no muy seguros, pero la seguridad perfecta es casi imposible de alcanzar. Por tanto la meta, en el diseño de sistemas seguro es conseguir que el coste de romper el sistema exceda al valor de la información que protege ese sistema seguro. Midiendo el coste tanto en dinero como en riesgo personal.

Capítulo 4

4.1 Desarrollo de una aplicación J2ME.

El ejemplo desarrollado es una simulación de una aplicación de pedidos orientada a la toma de órdenes.

El objetivo central de un sistema de automatización de pedidos es el de permitir al agente que levanta el pedido tener acceso a la información apropiada en el punto mismo de la toma de órdenes, así como suministrar la habilidad de ingreso de datos de los pedidos en dispositivos móviles en este caso Teléfono Celulares Convencionales. Estos podrán manejar conjuntos únicos de reglas de negocio" de la Empresa para alimentar de forma directa el sistema de procesamiento de órdenes, así como enviar información de retorno personalizada para el cliente.

La aplicación consta de un componente, que se instala en los dispositivos móviles de los agentes preventa (Agentes que levantan los pedidos). Un componente software que actúa como servidor al dispositivo móvil, instalado en un servidor web dentro de la instalación de la empresa. Para efectos de demostración se utilizara Netbeans 6.8 como software para gestionar la información ingresada por los agentes preventa.

Infraestructura necesaria:

- La mayoría de las empresas tienen servidores de aplicaciones, ip fijos, conexión a internet, certificados digitales.
- Se tendría que acondicionar servidores para pedidos.

Requerimientos:

- Servidor Web de Aplicaciones: Apache Tomcat 6.0.20
- Gestor de Base de Datos MySQL: MySQLessential para administración visual.
- Cliente Móvil: Equipo celular con soporte Java (MIDP2.0, CLDC 1.0), GPRS o WiFi.

Compatibilidad:

Una aplicación J2ME programada en su totalidad bajo los estándares propuestos puede ser portada en cualquier dispositivo J2ME, con lo cual no estamos atados a un solo fabricante y es más, tampoco a un solo modelo en particular dentro de la misma marca.

De esta forma estamos en condiciones de tener desarrollos ágiles y escalables de esta plataforma.

4.2 Esquema general de comunicación de pedidos:

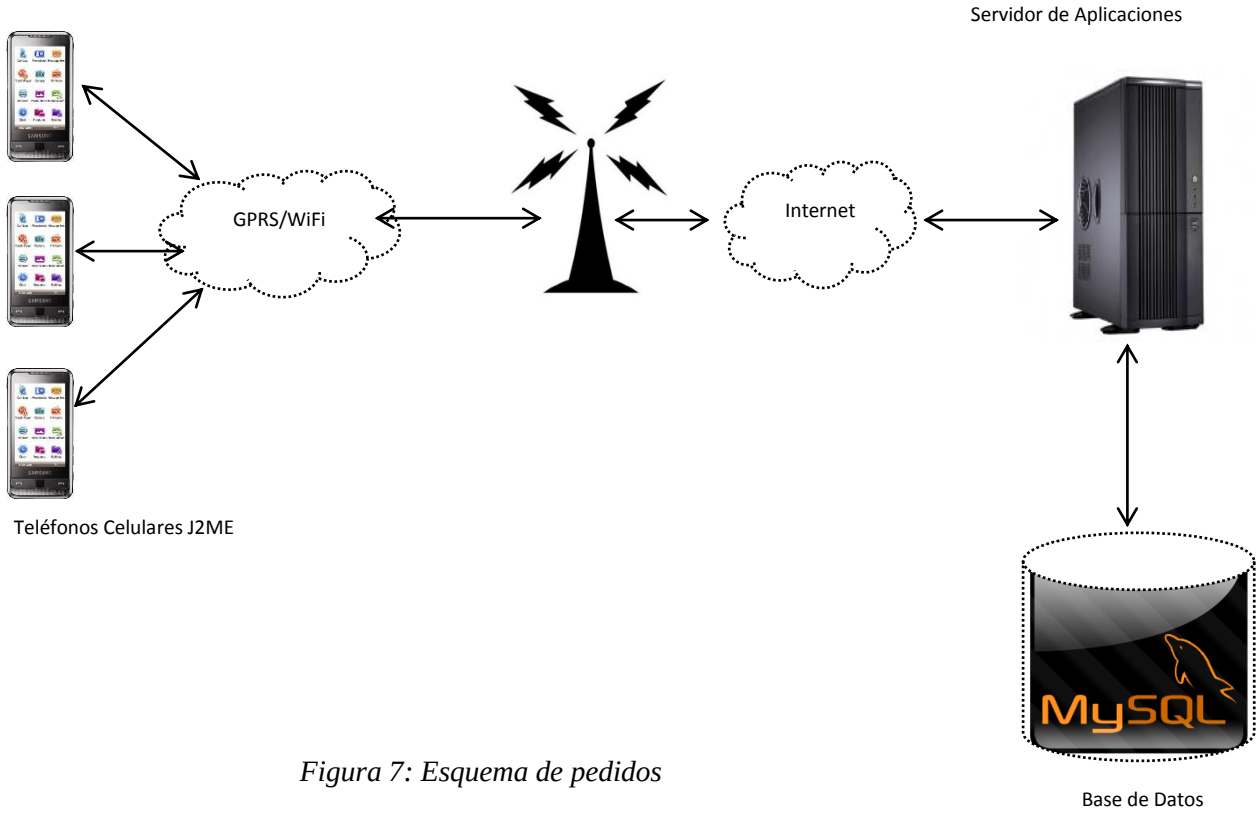


Figura 7: Esquema de pedidos

Aquí vemos como los teléfonos se comunican a sus compañías por GPRS/Wifi de ahí se conectan libremente a Internet, pudiendo acceder a servicios públicos de su empresa.

Ventajas y Desventajas de Desarrollar Aplicaciones de este tipo

Ventaja

- Ahorro de tiempo operativos de la empresa.
- Nuestra aplicación resulta más económica para la empresa debido a usar un móvil y no un dispositivo de altos costos en adquisición.
- Mayor rapidez en consultas
- En muy poco espacio se dispone de una gran cantidad de información.
- Información actualizada.
- Seguridad, etc.

Desventajas

- Dispositivos con pocos recursos.
- Poco espacio en pantalla.
- No se puede acceder a ciertos recursos del teléfono si este no incorpora el API correspondiente, etc.

A continuación presentamos el Documento de Especificación de Requisitos.

En los capítulos anteriores se ha expuesto el propósito y los objetivos del proyecto abordado y se ha tratado de plasmar de una manera clara y concisa, los conceptos y herramientas sobre los que se va a desarrollar este proyecto.

Tratándose de un prototipo software de un sistema de pedido, basado en consultas y respuestas para dispositivos móviles, este proyecto se clasifica en un proyecto de desarrollo software y como tal, debe seguirse las actividades de la Ingeniería del Software para conseguir un buen diseño e implementación. Así pues, detallamos las distintas fases de desarrollo que hemos seguimos para la realización de este prototipo.

4.3 Fases de Desarrollo

El desarrollo del proyecto se puede dividir en varias fases, que en este caso corresponderán al fin y al cabo a los pasos que se detallan mediante las técnicas de ingeniería del software, como serán el ***análisis del sistema***, ***desarrollo del sistema***, ***implementación*** y donde se tendrá en cuenta en cada una de estas fases, las peculiaridades y limitaciones que presentan los dispositivos móviles, aunque la aplicación será emulada en un ordenador, finalmente el prototipo desarrollado se ha diseñado para poder implantarse y ejecutarse en un dispositivo móvil.
Al día de hoy, podrán existir definiciones precisa, única, estandarizada y otras no tan precisas para la Ingeniería del Software. Sin embargo, las dos que se exponen seguidamente, abarcan ampliamente el concepto y resultan perfectamente válidas para su entendimiento:

- Ingeniería del Software es la construcción de software de calidad con un presupuesto limitado y un plazo de entrega en contextos de cambio continuo.

- Ingeniería del Software es el establecimiento y uso de principios y métodos firmes de ingeniería para obtener software económico que sea fiable y funcione de manera eficiente en máquinas reales.

La Ingeniería del Software requiere llevar a cabo numerosas actividades, las cuales se pueden agrupar en etapas, o también llamadas fases, que se detallan a continuación:

- **Especificación de Requerimientos**: define el propósito del sistema, las propiedades y restricciones del mismo, es decir, se describe el comportamiento esperado en el software una vez desarrollado. Gran parte del éxito de un proyecto de software radicará en la identificación de las necesidades del negocio, así como la interacción con los usuarios funcionales para la recolección, clasificación, identificación, priorización y especificación de los requerimientos del software.
- **Análisis del Sistema**: se obtiene un modelo del sistema correcto, completo, consistente, claro y verificable.
- **Diseño del Sistema**: se definen los objetivos del proyecto y las estrategias a seguir para conseguirlos.
- **Implementación**: se traduce el modelo a código fuente, pudiendo ser la parte más obvia del trabajo de la ingeniería del software. La complejidad y la duración de esta etapa está íntimamente relacionada al lenguaje de programación utilizado, al diseño previamente realizado y también, en nuestro caso, a las características y limitaciones propias de los dispositivos móviles.
- **Prueba**: verificar y validar el sistema. Básicamente consiste en comprobar que el software realice correctamente las tareas indicadas en la especificación del problema.

En los puntos siguientes se profundizará en cada una de estas etapas y en cómo se han llevado a cabo en el ámbito de nuestro proyecto.

4.4 Especificación de Requerimientos

El primer paso en la Ingeniería del Software debe ser determinar el propósito último del proyecto, las propiedades que debe satisfacer y las restricciones a las que está sometido. Este es, sin duda, un paso de vital importancia dentro del desarrollo de cualquier proyecto software ya que, sin conocer el propósito del mismo y todas las limitaciones a las que debe hacer frente, resultaría muy difícil poder realizar una aplicación software que cumpliera o se ajustara a dicho propósito.

En este caso, el propósito de nuestro proyecto es conocido desde el mismo momento de su propuesta y como ya se ha expuesto anteriormente, es el siguiente:

Diseño y desarrollo de un prototipo software que implemente un sistema de pedidos para dispositivos móviles utilizando el lenguaje de programación y la tecnología J2ME (Java 2 Micro Edition). Dicho software permitirá al usuario consultar información actualizada en el punto mismo de la toma de órdenes, así también levantar el pedido de un cliente.

Una vez determinado el propósito último del proyecto, el siguiente paso consiste en especificar los requerimientos del mismo. Los requerimientos de un proyecto software son el conjunto de propiedades o restricciones definidas con total precisión, que dicho proyecto software debe satisfacer. Existen dos tipos bien diferenciados de requerimientos:

- **Requerimientos funcionales**: son aquellos que se refieren específicamente al funcionamiento de la aplicación o sistema.
- **Requerimientos no funcionales**: aquellos no referidos al funcionamiento estricto sino a otros factores externos.

En los siguientes sub apartados definiremos cuales son estos requerimientos, tanto funcionales como no funcionales, para este proyecto. Sin embargo, estas definiciones sólo serán previas ya que en la actividad de análisis del sistema se pueden descubrir nuevas necesidades.

Requerimientos funcionales

Las funcionalidades que se emplean en esta aplicación son las siguientes:

- Pantalla Bienvenida
- Pantalla Autenticación
- Pantalla Conexión
- Pantalla Menú
- Pantalla Clientes
- Pantalla Levantar Pedidos
- Pantalla Lista de Clientes
- Pantalla Catalogo Productos
- Pantalla Datos del Cliente
- Pantalla Datos de Productos
- Pantalla Órdenes del Día

Requerimientos no funcionales

Los requerimientos no funcionales, como hemos visto, tienen que ver con características que de una u otra forma puedan limitar el sistema, como puede ser el equipo informático a utilizar, plataforma sobre la que se va ejecutar, el rendimiento, interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), etc. Son tan importantes como los propios requerimientos funcionales y pueden incluso a llegar a ser críticos para la aceptación del sistema.

En este caso, es importante prestar una especial atención a los requerimientos no funcionales, ya que nuestro prototipo de sistema de pedidos va dirigido a dispositivos móviles, los cuales imponen un alto grado de restricciones, debido a sus limitadas capacidades tanto en hardware como en software, y por lo tanto serán críticas a la hora de diseñar nuestro prototipo.

Teniendo esto en cuenta, los requerimientos no funcionales deben obtenerse y analizarse a partir de las restricciones que presenten estos dispositivos. A continuación, definimos dichos requerimientos en detalle:

Requerimientos no funcionales del dispositivo móvil

Nuestra aplicación va dirigida a una amplia gama de dispositivos móviles, muy diferentes entre ellos, por lo que los requerimientos no funcionales propios del dispositivo móvil, los podemos clasificar o dividir en dos tipos, los requerimientos de hardware y los requerimientos software.

Hardware

Los dispositivos móviles deben cumplir los siguientes requisitos de hardware como mínimo para ejecutar la máquina virtual de Java (KVM) necesaria para soportar la aplicación:

- Disponer entre 160 Kb y 512 Kb de memoria total disponible. Como mínimo disponer de 128 Kb de memoria no volátil para la Máquina Virtual Java y 32 Kb de memoria volátil (RAM) para la Máquina Virtual en tiempo de ejecución. También será necesario disponer de 64Kb en memoria permanente para ubicar la aplicación con sus bases de datos.
- Procesador de 16 o 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, ya sea sin cable o con cable, por ejemplo WIFI, bluetooth, USB, para efectuar la descarga de la aplicación al dispositivo.

Pantalla gráfica. Hay que considerar que estos dispositivos móviles presentan unas pantallas con características limitadas y muy distintas entre sí, tanto en tamaño como en resolución y gama cromática, con respecto a un monitor de un PC. Para ello habrá que prestar una mayor atención a los mecanismos necesarios para que la información que se muestre por pantalla aparezca de forma legible y pueda verse en su totalidad, a pesar de sus reducidas dimensiones y gama cromática.

Software

El requisito no funcional software para que el prototipo o aplicación se ejecute en un dispositivo móvil, es que dicho dispositivo soporte y posea la máquina virtual de Java, concretamente la KVM, que es la plataforma que está orientada y pensada para estos dispositivos de reducidas capacidades computacionales, de memoria y gráficas. La KVM actúa de intermediaria entre la aplicación y el sistema operativo del dispositivo móvil.

Hay que tener en cuenta que para el desarrollo de una aplicación para un dispositivo móvil también será necesario saber qué configuración y perfiles soporta, que va en función de las características del dispositivo. En este proyecto, al ir dirigido a un dispositivo móvil del tipo teléfono móvil, PDA, etc, la configuración a usar para la creación del prototipo será la CLDC (configuración de dispositivos limitados con conexión) versión 1.1, junto con el perfil MIDP (Mobile Information Device Profile) versión 2.0, que ofrece más prestaciones y presenta menos limitaciones que la versión MIDP 1.0, acaparando un mayor número de dispositivos. Estas API's y bibliotecas, como se especificó en el capítulos anteriores, describen las características básicas y común esa todos los dispositivos sobre los que se va a ejecutar la aplicación y controlaran el ciclo de vida de la misma, y ya están disponibles en la plataforma de desarrollo de Netbeans, por lo que según el dispositivo móvil sobre el que queremos ejecutar nuestro prototipo o aplicación, habrá que elegir previamente en la fase de desarrollo.

Requerimientos no funcionales de la interfaz

Los requerimientos de la interfaz gráfica entre la aplicación y el usuario están íntimamente ligados a la usabilidad y sus principios. Primeramente introduciremos la definición de

usabilidad, para tener una idea clara de lo que debe contemplarse a la hora de diseñar una interfaz gráfica y seguidamente comentaremos el criterio que se va a seguir para el diseño de la interfaz gráfica de nuestro prototipo.

El concepto de usabilidad se puede definir de varias formas:

- Se define coloquialmente como facilidad de uso, ya sea de una página Web, una aplicación para una PDA o cualquier otro sistema que interactúe con un usuario.

- La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso.

- La usabilidad es la efectividad, eficiencia y satisfacción con la que un producto permite alcanzar objetivos específicos a usuarios específicos en un contexto de uso específico.

A partir de estas tres definiciones se pueden obtener los principios básicos de la usabilidad, los cuales se asociarán a los requerimientos no funcionales que deberá cumplir la interfaz gráfica:

Facilidad de aprendizaje: Este principio se refiere a aquellas características de la interfaz que permiten comprender cómo usarla inicialmente y conseguir una interacción efectiva y productiva con nuevos usuarios. Depende de los siguientes factores:

- Predecibilidad: Una vez conocida la aplicación, se debe saber en cada momento a qué estado se pasará en función de la tarea que se realice.
- Síntesis: Los cambios de estado tras una acción deben ser fácilmente captados.
- Generalización: Las tareas semejantes se resuelven de modo parecido.
- Familiaridad: El aspecto de la interfaz tiene que resultar conocido y familiar para el usuario.
- Consistencia: Siempre se han de seguir una misma serie de pasos para realizar una tarea determinada.

Flexibilidad*:* Relativa a la variedad de posibilidades con las que el usuario y el sistema pueden intercambiar información. Características que hacen a una interfaz flexible son:

- La posibilidad de diálogo llamado también control del diálogo por parte del usuario, donde hay que proporcionar al usuario la capacidad para decidir cuándo empezar o acabar las operaciones, siempre que sea posible.
- Capacidad de adaptación, donde sería ideal que la interfaz pudiera adaptarse automáticamente a las necesidades del usuario actual.

Robustez: Es el nivel de apoyo al usuario que facilita el cumplimiento de sus objetivos y, también, la capacidad del sistema para tolerar fallos. Está relacionada con los siguientes factores o características:

- Navegable: El usuario debe poder observar el estado del sistema sin que esta observación repercuta de forma negativa en él.
- Recuperación de información: La aplicación debe permitir volver a un estado anterior.
- Tiempo de respuesta: Es el tiempo necesario para que el sistema pueda mostrar los cambios realizados por el usuario.
- Persistencia: Un sistema persistente es aquel en el que las notificaciones al usuario permanecen como objetos manipulables después de su presentación.
- Uso de valores por defecto: Ayudan al usuario mediante recuerdo pasivo. Mostrar un valor por defecto ayuda a que el usuario sepa qué tipo de valor debe introducir.

Para cumplir estos requerimientos, vamos a emplear los mecanismos apropiados que nos permitan utilizar la interfaz gráfica que ofrece cada dispositivo móvil. Es decir, la interfaz gráfica del prototipo o aplicación será la que disponga el dispositivo móvil, para que sea desde un principio más amigable, fácil e intuitivo la navegación y uso, de nuestro prototipo de pedido.

Análisis del Sistema

Una vez conocido el propósito del proyecto software, las funciones que debe cumplir y las restricciones a las que debe someterse, llega el momento de analizar el sistema y crear un modelo del mismo que sea correcto, completo, consistente, claro y verificable. Para conseguir esto se estudiarán los perfiles de usuario y se creará el Diagrama de Flujo en base a los requerimientos.

- **Perfil de Usuario**

  En esta fase el primer paso es determinar quiénes son los usuarios potenciales de la aplicación, para a partir de esto, obtener las características generales que nos permitan caracterizar los requisitos de usabilidad que posteriormente habrá que tener en cuenta en el diseño de la aplicación y de su interfaz gráfica. En nuestro caso, la aplicación va destinada a cualquier tipo de usuario que al menos tenga los conocimientos necesarios para manejar el dispositivo móvil donde se ubicará la aplicación.

- **Diagrama de Flujos**

  Representa un flujo de eventos. También se puede definir como la representación de una situación o tarea de interacción de un usuario con la aplicación. En nuestro sistema, vamos a contar sólo con un tipo de usuario, que vamos a definir como agente preventa, y corresponde con la persona que interactúe con el sistema.

Diseño del Sistema

Realizar de manera adecuada cada una de las actividades que conlleva la Ingeniería del Software es, indudablemente, indispensable para el desarrollo de un proyecto software de calidad. Por lo tanto, no se puede decir que ninguna de estas actividades sea más importante que otra. Sin embargo, sí podemos decir que la actividad de diseño es la más delicada y la más laboriosa de llevar a cabo.

Esto es debido a que si no se lleva a cabo correctamente se hace imposible el codificar, de manera correcta, en la fase de implementación el modelo obtenido en el análisis del sistema, lo que puede repercutir en hacer inútil todo el esfuerzo realizado durante las primeras actividades de la Ingeniería del Software.

También se considera la más laboriosa porque las estrategias a seguir para conseguir que la traducción entre modelo y código se lleve a cabo correctamente son muy diversas y complejas. Por tanto, se puede decir que el diseño del sistema es la actividad de la Ingeniería del Software en la que se identifican los objetivos finales del sistema, se plantean las diversas estrategias para alcanzarlos en la actividad de implementación.

Sin embargo, el sistema no se suele diseñar de una sola vez sino que hay que diferenciar entre el diseño y estructura de los datos que se van a manejar y el diseño de la interfaz entre la aplicación y el usuario. Estas dos fases del diseño no se realizan de forma consecutiva una detrás de la otra sino que lo normal es realizarlas de manera concurrente y finalizarlas a la vez.

Diseño de los datos

El objetivo de esta fase del diseño software es determinar la estructura que poseen cada uno de los elementos de información del sistema, es decir, la estructura de los datos sobre los que se va a trabajar.

En nuestro caso y teniendo en cuenta que se trata de un prototipo de pedidos orientado a dispositivos móviles, donde dichos dispositivos poseen unas limitaciones tanto en memoria como en capacidad de computación reducidas, no debemos abordar una excesiva complejidad en los datos que se van a manejar, ya que por un lado no nos lo permitirían estos dispositivos (no soportan todos los tipos de datos que suelen soportar un PC) y además sufriríamos un detrimento de las prestaciones en cuanto a velocidad de reacción y refresco del dispositivo. Por lo tanto, lo que intentaremos es simplificar al máximo y quedarnos con los datos realmente importantes o que más interesen para conseguir nuestro objetivo.

Teniendo en cuenta lo comentado, el elemento que consideramos es el siguiente:

- **Imágenes**: Se ha eliminado envió de dato tipo imagen, puesto que el almacenamiento de este tipo de datos requiere de mucho poder de procesamiento y capacidad.

Una vez determinados cuales son los elementos de información del sistema, se deben obtener sus representaciones gráficas.

Diseño de la interfaz

En esta fase del diseño del sistema software se define cual va a ser la apariencia visual de la aplicación, es decir, se define la interfaz visual entre el usuario y la aplicación. Sin duda, realizar un buen diseño de la interfaz resulta primordial ya que ésta debe presentarse atractiva al usuario de la aplicación pero, a la vez, le debe de resultar fácil de entender y trabajar sobre ella.

Este aspecto, se puede presentar como un reto importante en nuestro caso ya que, por un lado, pretendemos que nuestro prototipo vaya dirigido a un diverso número de dispositivos móviles, donde cada uno presenta una interfaz propia, con sus peculiaridades y limitaciones inherentes, marcada por ejemplo por el tamaño de pantalla y su rango cromático, la distribución y forma de visualizar los elementos en pantalla, la distinta ubicación y funcionalidad del teclado, o por las características de los sonidos que son capaces de emitir,

monofónicos o polifónicos, pero donde se busca que la interacción que se produzca entre el usuario y la aplicación, sea lo más semejante posible en todos los casos e igualmente intuitiva, fácil de entender y usar.

Ahora bien, la tecnología J2ME, a través del perfil MIDP, nos permite definir diversos aspectos gráficos, como diseñar interfaces de usuario, donde queremos resaltar que se hace una división entre interfaces de usuario definidas de alto nivel y de bajo nivel:

- **Las interfaces de usuario de alto nivel** usan componentes tales como botones, cajas de texto, formularios, etc, elementos que son implementados por cada dispositivo y la finalidad de usar las API's de alto nivel es su portabilidad. Al usar estos elementos, se pierde el control del aspecto de nuestra aplicación ya que la estética o guía de estilo de estos componentes depende del dispositivo donde se ejecute. En cambio, usando estas API's de alto nivel ganaremos un alto grado de portabilidad de la aplicación entre distintos dispositivos.

- **Con las interfaces de usuario de bajo nivel**, se tendrá un control completo sobre los recursos del dispositivo y podremos controlar eventos de bajo nivel como, por ejemplo, el rastreo de pulsaciones de teclas. Nos permitirán crear aplicaciones muy variadas en lo que se refiere al aspecto gráfico y por lo general pueden ser más vistosas. Aunque su programación se vuelve más tediosa y aún más importante es que hay que tener en cuenta las peculiaridades del dispositivo al que va dirigido. Por lo que, la aplicación pierde portabilidad entre distintos dispositivos.

Teniendo en cuenta las distintas ventajas e inconvenientes que se presentan al diseñar interfaces de usuario para aplicaciones dirigidas a dispositivos móviles, con cada una de las divisiones expuestas y dado que el objetivo marcado para nuestra aplicación es que sea compatible con el mayor número posible de dispositivos. Nos decantamos por usar para nuestro diseño una interfaz de usuario de alto nivel, ya que es la que nos brinda una mayor compatibilidad, pero además, conseguimos otro objetivo relevante en el diseño de una interfaz, que es que el usuario se va sentir familiarizado desde el principio con la interfaz de la aplicación, ya que es la que su propio dispositivo móvil utiliza, tanto en la asignación de las funciones del teclado, como el aspecto visual de menús y demás elementos gráficos.

Así pues, en todo diseño de una interfaz, hay que definir una serie de aspectos, entre los que se destaca el estilo de la aplicación, las pantallas, los caminos de navegación y secuencias de diálogo, que en nuestro caso van estar condicionados por la interfaz proporcionada por el dispositivo.
Pon tanto, tendremos que el estilo y parte del diseño de las pantallas vendrán ya definidos por la interfaz propia del dispositivo móvil sobre el que se ejecute. El diseño final de las pantallas, en lo referente a qué elementos mostrará y parte de la distribución que presentarán los mismos, junto con los caminos de navegación y secuencias de diálogo, dependerán de nuestro criterio.

Pantallas de la aplicación

En este apartado vamos a definir la estructura de nuestra interfaz con el usuario. Mediante la elaboración de pantallas se pretende esbozar lo que será la interfaz de usuario de nuestra aplicación. Dichas pantallas expresan el diseño final, de lo que es nuestro sistema. El aspecto final puede diferir mucho de un dispositivo a otro, en función de su interfaz.

Usualmente para diseñar las pantallas se utilizan los prototipos de papel, que son una forma de crear una imagen palpable de lo que será una futura aplicación. Su creación y manipulación es rápida y elástica. Además permite a los usuarios imaginarse lo que será la futura aplicación en funcionamiento. Sin embargo, nosotros disponemos de una herramienta muy útil como el emulador predeterminado de aplicaciones de dispositivos móviles, desarrollado por Sun Microsystems y que se puede lanzar desde el entorno de desarrollo de Netbeans. Este emulador permite a los usuarios ejecutar previamente su aplicación antes de descargarla en su dispositivo móvil, para ver de una forma aproximada, lo que será su futura aplicación en funcionamiento, y permitiendo depurar los posibles fallos que pueda tener, aunque el aspecto final realmente dependerá de la interfaz del dispositivo sobre el que se ejecute, pero al menos cumple con el objetivo de mejorar y depurar el diseño de la interfaz y el funcionamiento general de la aplicación. Con él, mostraremos las diversas pantallas que se diseñarán y que componen nuestra aplicación.

Pantalla Bienvenida: Es un mensaje que se muestra al usuario, no tiene ninguna función más que lograr una presentación amena en la aplicación.

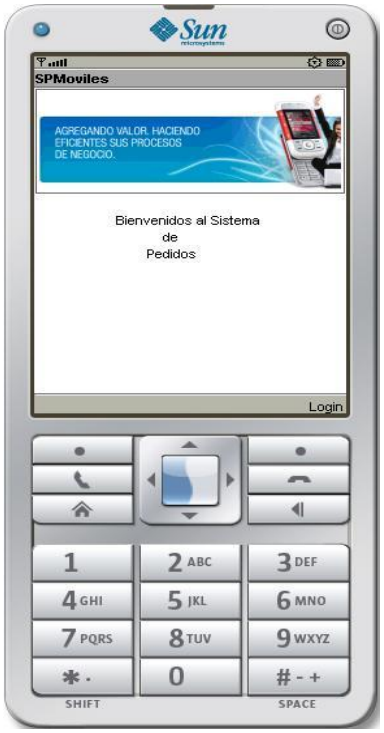


Figura 8: Pantalla Bienvenida

Pantalla Autenticación: Esta pantalla nos da la facilidad de ingresas un usuario y una contraseña, datos que serán confirmados para validar la existencia de dicho usuario en una base de datos. Este proceso nos permite acceder a todas las funciones de la aplicación en caso de que la información dada al sistema sea la correcta.

Figura 9: Pantalla Autenticación

Pantalla Conexión: Es una pantalla que nos sirve como un stop al usuario para no interferir en los procesos de conexión. Es una pantalla que no tiene ninguna funcionalidad adicional para la aplicación, más que para darle mayor realce y apariencia.

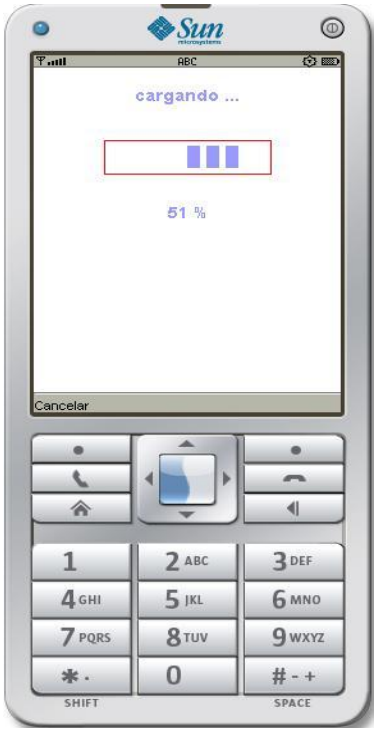


Figura 10: Pantalla Conexión

Pantalla Menú: Esta pantalla se encarga de mostrarnos las distintas acciones que podemos realizar en la aplicación. Podemos realizar el pedido, ver catálogos de productos, catálogos de clientes, información de cada cliente así como información de cada producto. Además de ver los pedidos que se realizaron en el día.

Figura 11: Pantalla Menú

Pantalla Clientes: Una vez que se elige la opción "Toma Pedidos" nos permite elegir a que cliente le vamos a levantar la orden de pedido.

Figura 12: Pantalla Clientes

Pantalla Levantar Pedidos: Realiza todo el proceso de la toma del pedido. Se establece al cliente al cual se le va a realizar el pedido y se eligen datos como la fecha en que se le está tomando el pedido y la fecha en que se le entregara el pedido. Así también se elige el producto que el Cliente quiere y se establece la cantidad y se calcula el total. El precio del producto seleccionado es obtenido a través de una petición al sistema. Una vez realizado el pedido este se almacena en la memoria del dispositivo para llevar un control de todos los pedidos hechos en el día.

Figura 13: Pantalla Levantar Pedidos

Pantalla Lista de Clientes: Esta pantalla nos permite ver todos los clientes del sistema.

Figura 14: Pantalla Lista Clientes

Pantalla Catalogo Productos: Esta pantalla nos permite ver todos los productos en existencias.

Figura 15: Pantalla Catalogo Productos

Pantalla Datos del Cliente: En esta pantalla se realiza una búsqueda de un determinado cliente. Se implementa una lista con todos los clientes debido a que la introducción de datos en un móvil resulta un poco complicada por el tamaño de las teclas por lo cual se optó por presentar los clientes en una lista y que el usuario elija el que desee.

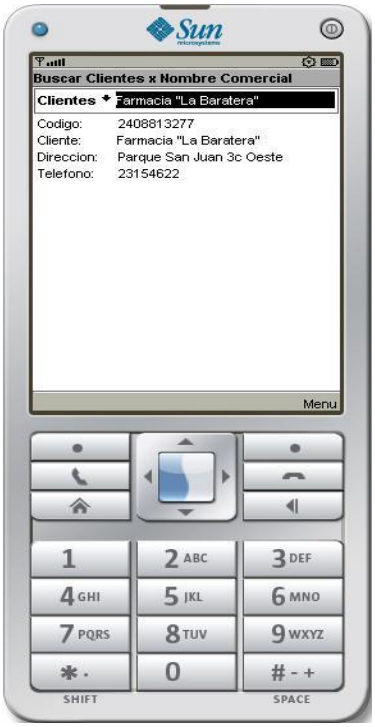


Figura 16: Pantalla Datos Cliente

Pantalla Datos de Productos: Es una pantalla similar a la del cliente. Se elige un producto en específico y se muestra información únicamente referente a ese producto que se eligió.

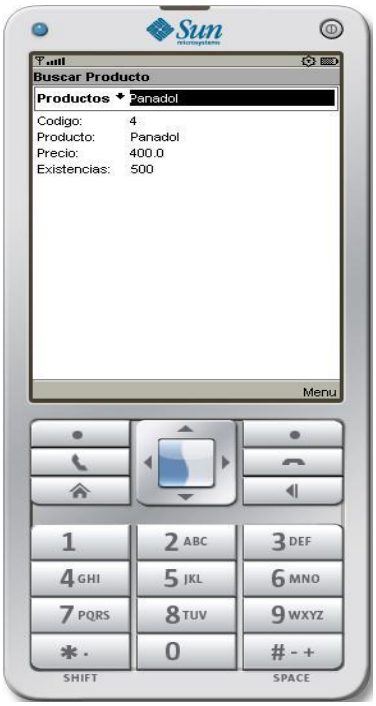


Figura 17: Pantalla Datos Producto

Pantalla Órdenes del Día: Nos muestra todos los pedidos realizados en el transcurso del día. Es como un back up de todos los pedidos enviados al sistema de base de datos de la empresa.

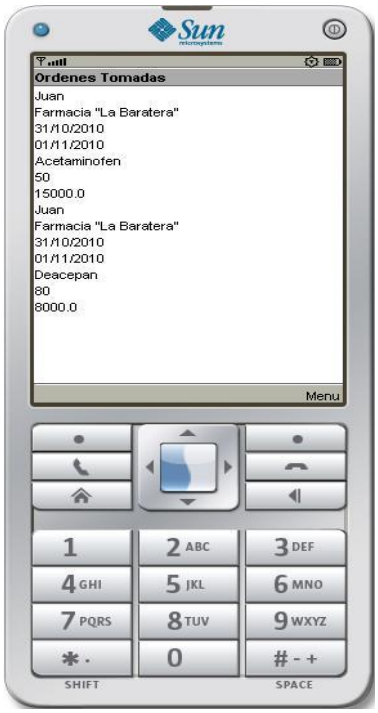


Figura 18: Pantalla Ordenes del Día

Implementación

La implementación es la actividad final de la Ingeniería del Software, aquella en la que el modelo obtenido en las actividades anteriores se debe transformar en código fuente. Para ello se debe ser cuidadoso en la elección del lenguaje de programación empleado para la codificación y de la herramienta utilizada para generarla.

La elección del lenguaje de programación así como el de la herramienta utilizada para su desarrollo viene especificada desde la definición del proyecto y en Capítulos anteriores se ha tratado números aspectos sobre el lenguaje y sobre la implementación de aplicaciones para dispositivos móviles con dicho lenguaje.

Tipo de arquitectura de la aplicación

En nuestro caso, se trata de una sencilla aplicación destinada a una gran variedad de dispositivos móviles, los cuales deben disponer de la Plataforma Java correspondiente. Esta plataforma, como ya se comentó anteriormente, es intermediaria entre la aplicación y el sistema operativo del dispositivo, la cual permite que una aplicación implementada con el lenguaje J2ME se pueda ejecutar en este tipo de dispositivos, con características muy distintas entre ellos. Concretamente, esta aplicación va dirigida a los dispositivos que posean el perfil MIDP 2.0 (perfil para dispositivos de información móvil versión 2.0). Actualmente es el que más dispositivos móviles soporta, aunque no sea la versión más actual.

La aplicación ha de ser descargada en el dispositivo móvil, por alguna de las vías de comunicación que presente, como puede ser por el puerto USB, por Bluetooth, infrarrojos, por WIFI o acceso a Internet. Según el dispositivo móvil, puede ser que en el mismo proceso de descarga se realice la instalación, o bien, haya que localizar los archivos con extensión .jar y .jad, concretamente ConexionMidletsServletBD.jar y ConexionMidletsServletBD.jad y proceder a la instalación.

Lenguajes de programación utilizados

Como ya se ha especificado con anterioridad, tanto en la definición del propósito y objetivos del proyecto, la implementación de esta aplicación se ha realizado usando Java J2ME. Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems. El lenguaje en sí mismo es similar en sintaxis a C y C++, pero tiene un modelo de objetos más sencillo y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la

manipulación directa de punteros o memoria. En este caso J2ME es una versión reducida de J2SE, que se puede traducir como Plataforma Java, Edición Estándar.

Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.
No vamos a entrar en más detalles, ya que como se ha puntualizado, se ha tratado de forma detallada con anterioridad.

Entornos de Desarrollo Utilizados

Netbeans 6.8: es un entorno de desarrollo integrado (IDE) que permite editar programas en java, compilarlos, ejecutarlos, depurarlos, construir rápidamente la interfaz gráfica de una aplicación eligiendo los componentes de una paleta, etc.

Este software tanto para linux como para Windows, puede descargarse gratuitamente de la página web http://www.netbeans.org/.

Configuraciones Netbeans para una conexión a base de datos MySQL 5

Crear una conexión a una Base de Datos MySQL desde NetBeans

Para crear una conexión a una Base de DatosMySQL desde NetBeans tenemos que seleccionar el Driver de MySQL para Java, en este caso MySQL (Connector/J driver) y seguidamente seleccionar ConnectUsing...

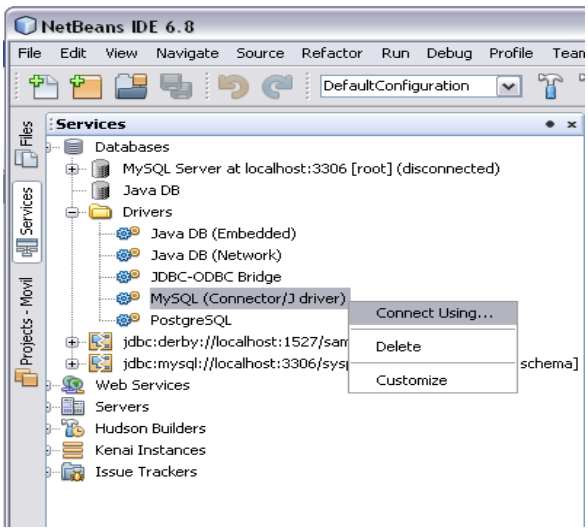


Figura 19: Conexión Base de Datos Con MySQL

En el cuadro de diálogo New Database Connection completamos los siguientes datos

- Host: localhost
- Port: 3306
- Database: syspedidos
- UserName: root
- Password: raiz

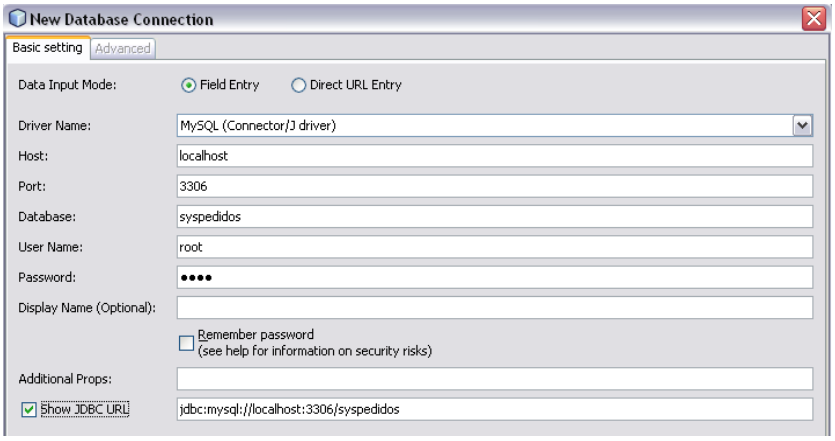


Figura 20: Configuración de Conexión a Base de Datos

Como podemos observar, la conexión a la Base de Datos syspedidos se ha realizado correctamente.

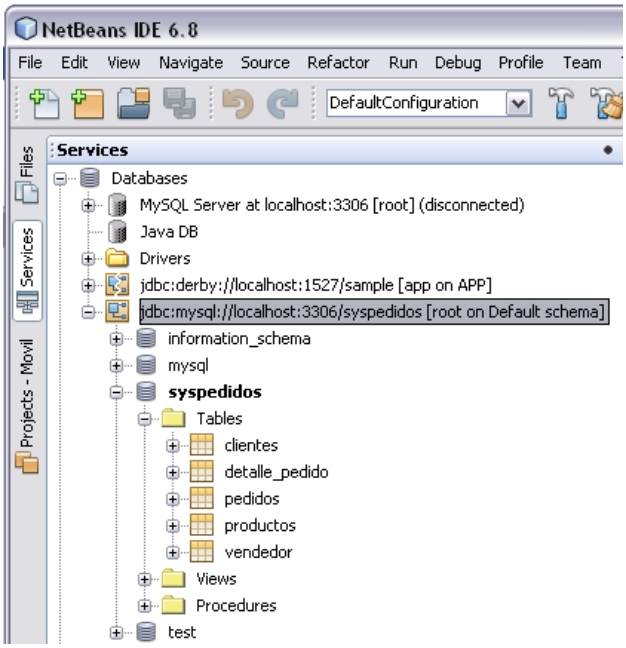


Figura 21: Conexión Realizada a Base de Datos MySQL

Gestor de Base de Datos MySQL

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario.

Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI C.

¿Por qué MySQL?

Esta utilidad se traduce en ventajas, entre las que podemos mencionar las siguientes:

- Acceso a las bases de datos de forma simultánea por varios usuarios y/o aplicaciones.

- Seguridad, en forma de permisos y privilegios, determinados usuarios tendrán permiso para consulta o modificación de determinadas tablas. Esto permite compartir datos sin que peligre la integridad de la base de datos o protegiendo determinados contenidos.

- Potencia: SQL es un lenguaje muy potente para consulta de bases de datos, usar un motor nos ahorra una enorme cantidad de trabajo.

- Portabilidad: SQL es también un lenguaje estandarizado, de modo que las consultas hechas usando SQL son fácilmente portables a otros sistemas y plataformas.

En concreto, usar **MySQL** tiene ventajas adicionales:

- Escalabilidad: es posible manipular bases de datos enormes, del orden de seis mil tablas y alrededor de cincuenta millones de registros, y hasta 32 índices por tabla.

  MySQL está escrito en C y C++ y probado con multitud de compiladores y dispone de API's para muchas plataformas diferentes.

- Conectividad: es decir, permite conexiones entre diferentes máquinas con distintos sistemas operativos. Es corriente que servidores Linux o Unix, usando **MySQL**, sirvan datos para ordenadores con Windows, Linux, Solaris, etc. Para ello se usa TCP/IP, tuberías, o sockets Unix.

- Es multihilo, con lo que puede beneficiarse de sistemas multiprocesador.
- Permite manejar multitud de tipos para columnas.
- Permite manejar registros de longitud fija o variable.

4.5 Tablas de la Base de Datos

Entidades y Relaciones

Las entidades básicas de este ejemplo son:

- Clientes (Compradores de productos).
- Productos (Items ofertados a clientes).
- Pedidos (Ordenes de items).
- Vendedor (Personal que atiende el pedido)
- Detalle de pedidos (Cantidad solicitada por el Cliente y Total)

PEDIDOS					
ID_PEDIDO	**ID_CLIENTE**	**ID_VENDEDOR**	**FECHA_PEDIDO**	**FECHA_ENVIO**	**DESCRIPCION**

PRODUCTOS			
ID_PRODUCTO	**NOMBRE**	**PRECIO**	**EXISTENCIAS**

DETALLE_PEDIDO			
ID_PEDIDO	**ID_PRODUCTO**	**CANTIDAD**	**TOTAL**

CLIENTES			
ID_CLIENTE	**NOMBRE_CLIENTE**	**DIRECCION_CLIENTE**	**TELEFONO_CLIENTE**

VENDEDOR	
ID_VENDEDOR	**NOMBRE**

Información requerida para cada una de las entidades descritas

Entidad	Información Requerida
Clientes	idNumClientes, Nombre, Dirección, Teléfono
Productos	idNumProducto, nombreProd, precio, existencias
Pedidos	idNumPedidos, Clientes_idNumClientes, idVendedor, fechapedido, fechaenvio, descripcion
detalle_pedido	Productos_idNumProducto, Pedidos_idNumPedidos, cantidad, total
Vendedor	idVendedor, nombreVendedor

4.6 Diagrama de la Base de Datos utilizado

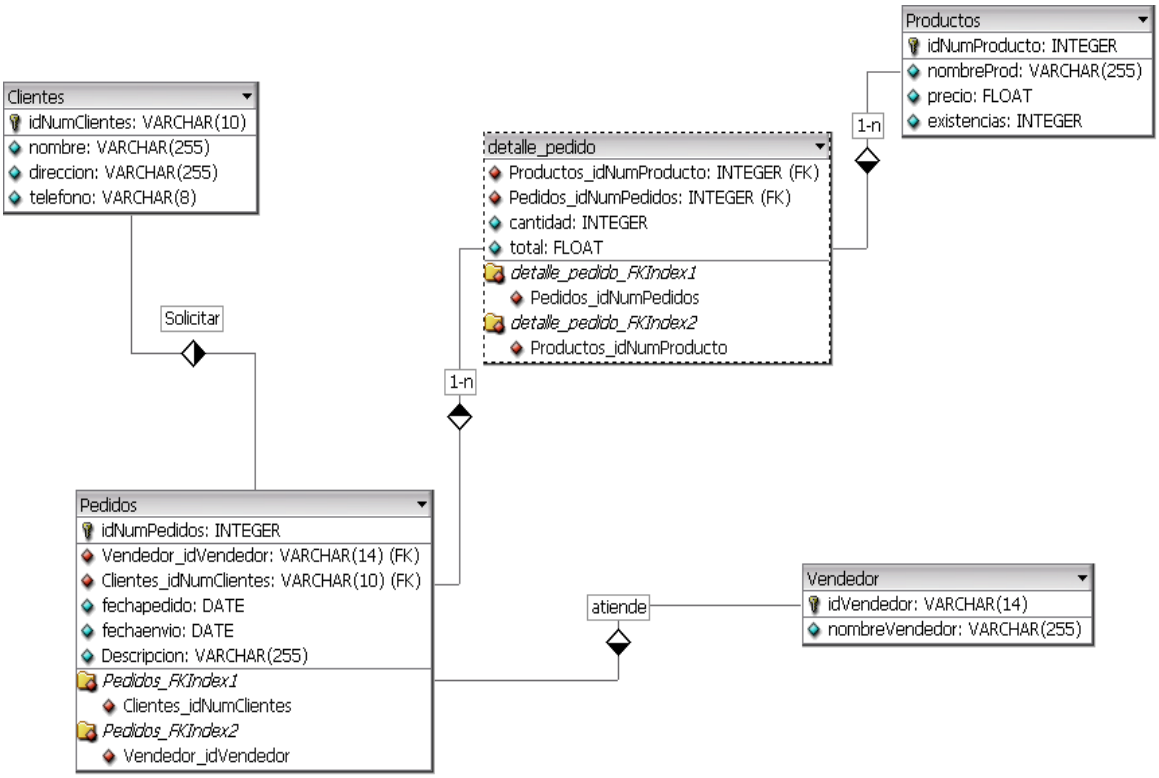


Figura 22: Diagrama de Base de Datos

4.7 Descripción de la Aplicación

La aplicación consiste en la creación de un software para móviles, que permita el acceso a información situada en una base de datos MySQL, la cual será consultada a través de una aplicación Servlets alojado en un Servidor de Aplicaciones que a su vez estará comunicado con la aplicación móvil.

Se trata de un sistema orientado a actividades típicas de la toma de orden a una entidad (Farmacia), donde se pueden realizar operaciones tales como:

- Toma de Orden del Producto
- Catálogo de Productos
- Catálogo de Clientes
- Búsqueda de Productos
- Búsqueda de Clientes
- Historial de Pedidos

Para el desarrollo del trabajo se utilizó el lenguaje de programación Java, debido a que sus características lo hacen adecuado para el propósito planteado: seguridad, robustez y sobre todo, portabilidad.

El interfaz gráfico es simple, permite una fácil interacción con el usuario, es estándar para todos los terminales móviles que soportan J2ME.

Por razones de que se trata de una aplicación de negocios y para lograr la mayor portabilidad posible del aplicativo se eligió para desarrollar la interface de usuario las API's de alto nivel, donde no se tiene un control total del aspecto de los controles, su estética depende exclusivamente del dispositivo donde se ejecute.

Otro aspecto muy interesante a la hora de desarrollar una aplicación móvil utilizando J2ME es poder almacenar localmente cierta información útil en el teléfono celular para no tener que volver a realizar una petición al servidor sobre datos solicitados anteriormente.

Las aplicaciones que se desarrollan bajo la configuración CLDC (Conected Limited Device Configuration) y el perfil MIDP (Mobile Information Device Profile) se denominan MIDlets. Por lo tanto esta aplicación como se desarrolló bajo la configuración CLDC 1.1 y el perfil MIDP 2.0 es un MIDlet.

Conclusión

Java 2 Edition (J2ME), constituye la clave del acercamiento de las aplicaciones inalámbricas a los dispositivos móviles. Nos proporciona una serie de API´s para desarrollar aplicaciones rápida y eficazmente.

Se ha podido comprobar las ventajas y desventajas en la utilización de la tecnología J2ME para de desarrollar aplicaciones conectadas a red, así como también el uso de bases de datos MySQL y NetBeans como herramienta de desarrollo.

Se pudo demostrar que en las aplicaciones orientadas a conexión se puede obtener información online cuando el usuario que manipule el Dispositivo Móviles lo requiera, no importando el punto donde este se encuentre siempre y cuando tenga una conexión a red.

Así mismo se pudo apreciar la facilidad en la creación de interfaces para aplicaciones móviles con J2ME que nos han permitido el ingreso y manipulación de datos en Dispositivos Móviles.
.

Recomendaciones

Aunque en la Aplicación de ejemplo se hace uso del protocolo http, se recomienda implementar el protocolo https el cual tiene como idea principal crear un canal seguro sobre una red insegura. Esto proporciona una protección razonable contra ataques.

Aunque J2ME es una buena alternativa para desarrollar aplicaciones basadas en un estándar para que aplicaciones sencillas se puedan implantar en una gran variedad de dispositivos, el uso de Android para el desarrollo de aplicaciones es una excelente opción para que programadores puedan desarrollar software para todo tipo de aparato con algunos mínimos requerimientos. Y aunque hasta el momento las aplicaciones Java para móviles eran uno de los grandes problemas dentro del sector. Cada aplicación tenía que ser optimizada no sólo para cada marca, sino también, en ocasiones, según plataforma. Gracias al desarrollo de Myriad J2Android se puede convertir de manera sencilla cualquier MIDLet Java para móvil en una aplicación nativa de Android, instalarla y ejecutarla de manera sencilla pudiendo realizar los ajustes de resolución desde la propia aplicación.

Bibliografía

- Software de aplicación para móviles – Rubén Darío Sánchez
- The Complete Reference J2ME – Mc Graw Hill
- Java 2 Micro Edition Manual de Usuario y Tutorial – Agustín Froufes Quintas, Patricia Jorge Cárdenes.
- Eric Giguere, Databases and MIDP, Part 1: Understandingthe Record Management System, February 2004.
- Eric Giguere, Databases and MIDP, Part 2: Data Mapping, May 2004.
- http://studentdevelopers.ning.com/

  Forum de desarrollo para dispositivos móviles
- Acceso a Base de Datos a través de Teléfonos Celulares – Sergio Andrés Soto, Argentina

ANEXO

MySQL Essentials para Windows

MySQL es un sistema gestor de bases de datos relacionales *"open source"* o de código abierto extendido por todo el mundo en especial por su uso en el desarrollo web, pero también utilizado por grandes empresas como Yahoo o Google.

Instalación y Configuración Paso a Paso

Instalación MySQLServer

Paso 1.

Una vez descargada la versión correspondiente, iniciamos la instalación a través del asistente.

Avanzamos a la siguiente pantalla, pudiendo escoger entre las opciones Typical, Complete y Custom. La instalación Típica está recomendada para los usuarios que desconocen los parámetros de instalación e instala MySQL SERVER con los valores por defecto. Si seleccionamos la instalación completa instalará en disco todos los componentes en los directorios predeterminados.

Por último, la opción personalizada o Custom es la recomendada a los usuarios avanzados pues muestra los parámetros de forma que puedan ser modificados.

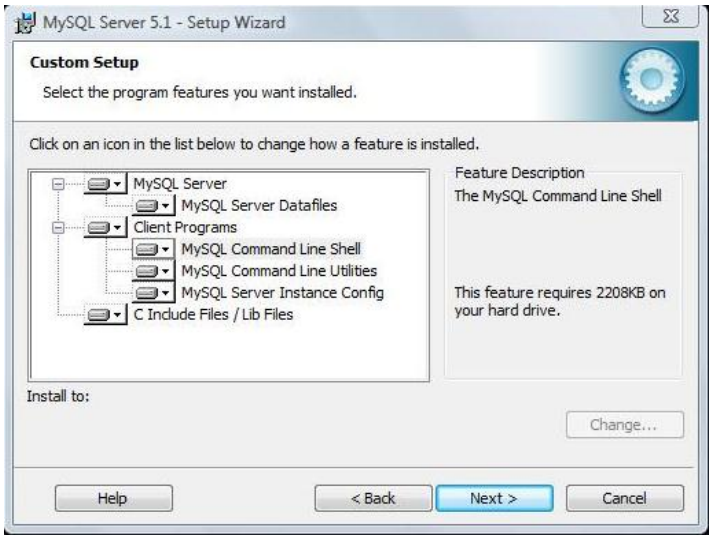
Es esta última opción, la Personalizada, la que seleccionamos para poder observar qué componentes del servidor MySQL y dónde los está instalando el asistente.

Paso 2

La instalación incluye la creación de un directorio en el que se instalará el servidor y una carpeta para los *datafiles* o ficheros de datos. Los componentes del servidor, cliente y librerías los instalará por defecto en la carpeta Program FilesMySQLMySQL Server 5.1 y los *datafiles* en Program DataMySQLMySQL Server 5.1.

Pulsando el comando *Change* podremos cambiar el directorio de almacenamiento de instalación por defecto del servidor, los programas cliente y las librerías.

En este caso pulsamos *Next* con las opciones por defecto.

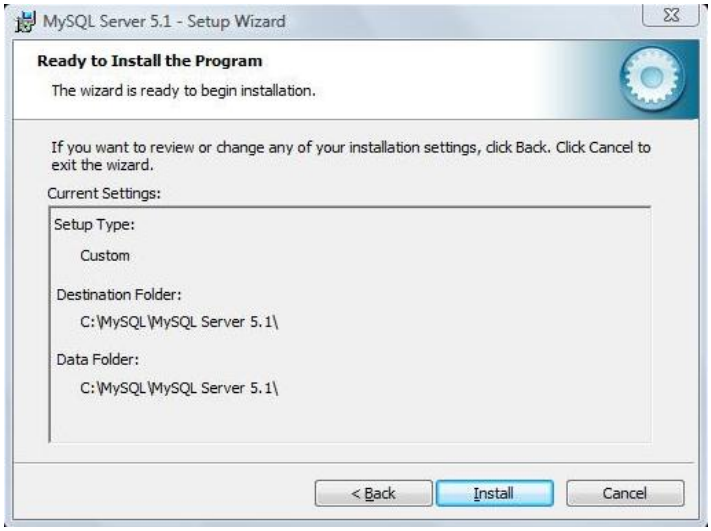


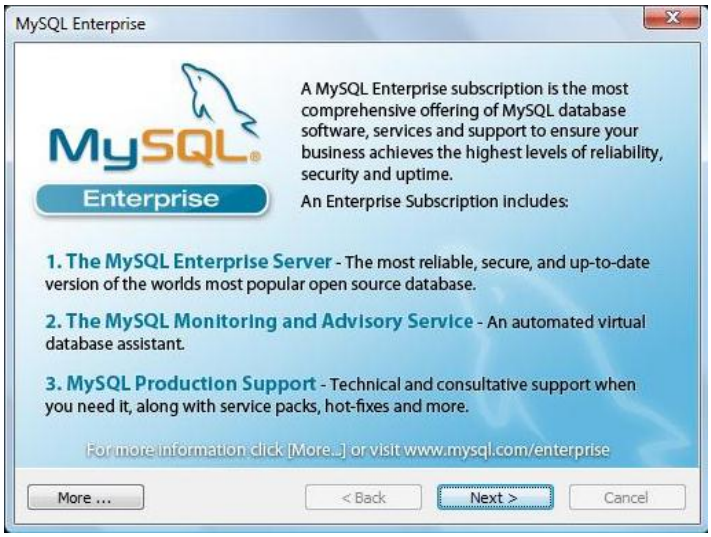
Paso 3.

Como podemos observar, nos muestra los destinos elegidos en la siguiente pantalla, si es necesario cambiarlos porque hemos previsto otras carpetas de instalación o no están dados los permisos de escritura y ejecución correspondientes, se puede volver a la pantalla anterior con *Back.*

En cualquier caso, ya es sólo necesario pulsar *Install*.

Mientras se realiza la instalación se nos muestran las posibilidades de MySQL Enterprise, la versión de MySQL más completa dirigida especialmente para empresas.





Paso 4

Una vez realizada la instalación nos dará la posibilidad de Configurar el Servidor iniciando el asistente de forma automática. Esta opción estará disponible también después de la instalación como *MySQL Server Instance Config Wizard* como veremos en el apartado de mantenimiento y también puede ejecutarse desde la línea de comando del sistema operativo con la orden*MySQLInstanceConfig.exe*. Con cualquiera de las tres posibilidades iniciamos el asistente de configuración MySQL Server Instance.

En este caso aprovechamos que la opción de inicio de la Configuración está activada para iniciar el asistente.

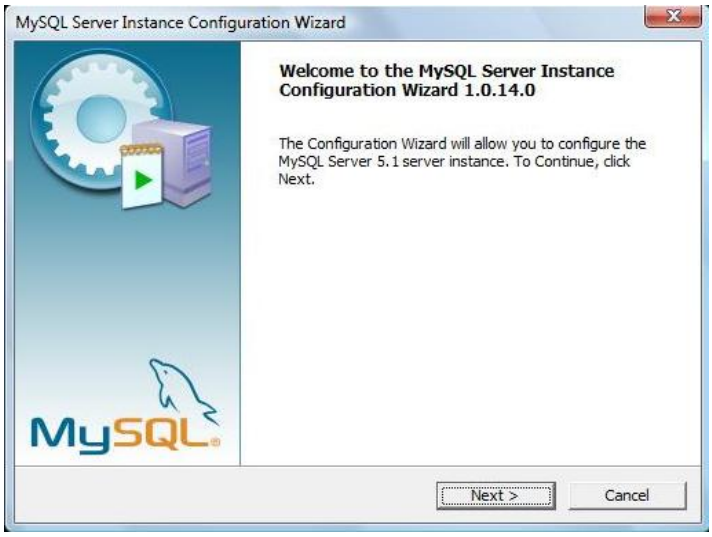
Configuración de MySQL Server

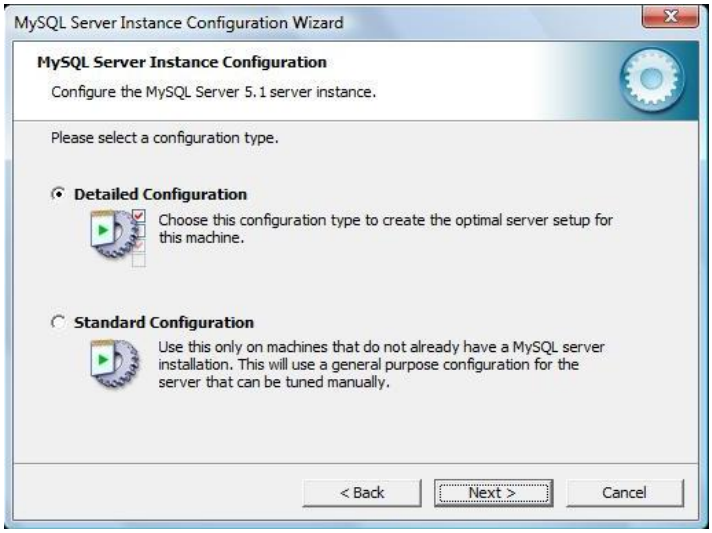
Paso 1

Podemos elegir entre la instalación Estándar y la Detallada. La versión Estándar únicamente pedirá valores para las opciones de configuración del Servicio (Paso 8) y las opciones de seguridad o modo de conexión (Paso 9); está indicada para usuarios no avanzados que desean implantar el servidor con los parámetros por defecto.

La versión Detallada permite al usuario adaptar la configuración a las premisas de su entorno de implantación, para ello es preciso tener claro el objetivo o fin para el que se utilizará la base de datos, pues es fundamental a la hora de configurar ciertas opciones.

En esta guía vamos a utilizar la opción de configuración Detallada.





Paso 2

Dentro de la configuración detallada, lo primero que debemos escoger es el tipo de servidor en el que vamos a alojar el sistema gestor de base de datos dependiendo del uso que vamos a hacer de este y de los recursos de que va a disponer para su ejecución. Existen tres posibilidades:

- Developper Machine: MySQL va a convivir con otras aplicaciones que no conforman un servidor en su conjunto. Normalmente se aplica para uso local o redes de poca extensión. Los recursos destinados al servicio serán los mínimos necesarios.
- Server Machine: MySQL va a convivir con otros servicios como el alojamiento de ficheros, el envío de correo electrónico, el servicio de páginas web, etc.

- Dedicated MySQL Server Machine: MySQL está alojado en un servidor dedicado y por tanto es el único servicio que corre en la máquina y todos los recursos del sistema estarán destinados a su funcionamiento.

En este caso seleccionamos la primera opción, ya que es probable que un entorno docente de trabajo existan otras aplicaciones destinadas a diversos fines ya instaladas en el equipo anfitrión.
Si el usuario estuviese implantando un servidor web, puede escoger la segunda opción para instalar cada uno de los servicios por separado, esta es una práctica excelente en módulos sobre implantación de aplicaciones.

Paso 3

Después debemos escoger el tipo de almacenamiento de los datos. Esto dependerá del destino o fin al que vaya dirigida la instalación.

Existen dos términos especiales que hay que explicar en este apartado antes de que usted elija: Tablas MyISAM e InoDB.

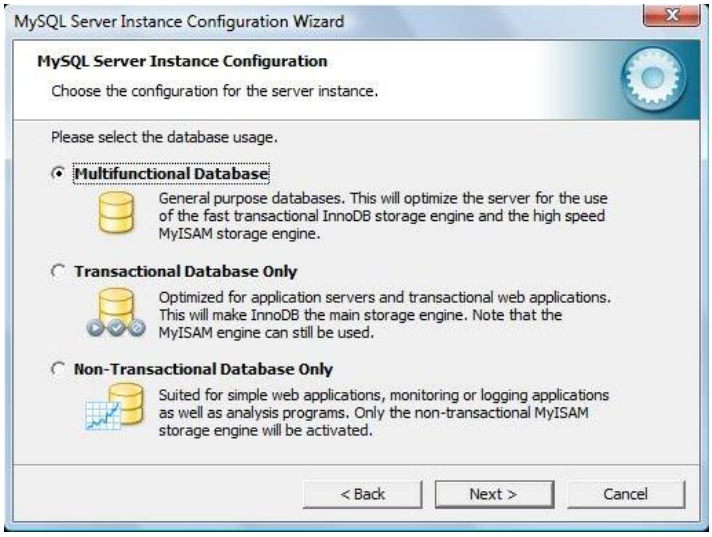
- Las tablas MyISAM son la adaptación de MySQL de una tecnología que manipula los datos a gran velocidad, pero sin tratamiento de las transacciones.
- Las tablas con tecnología de almacenamiento InoDB tienen un alto control de las transacciones que se realizan en ellas, por lo que aumenta la seguridad e integridad de los datos pero disminuye la velocidad de acceso a ellos. Es recomendable para las

tablas en las que se realizan actualizaciones en la que se precisa controlar que estas se realicen correctamente.

En base a estas tecnologías soportadas por MySQL, las tres opciones entre las que podemos elegir son:

- Base de Datos Multifuncional. Por tanto esta base de datos almacenaría tanto tablas con control de transacciones InoDB como tablas de acceso rápido MyISAM, compartiendo los recursos entre las dos alternativas. Es la opción recomendada por defecto.
- Base de Datos Transaccional: Sus tablas deben controlar las transacciones de forma segura no importando tanto la velocidad de acceso. Por ello el almacenamiento por defecto será InoDB, pudiendo el usuario configurar en tablas concretas el almacenamiento MyISAM.
- Base de Datos No Transaccional: Prima la velocidad de acceso a la información contenida en sus tablas, por lo que por defecto el almacenamiento será MyISAM. En este caso el usuario no podrá posteriormente seleccionar la tecnología InoDB sin volver a configurar la instancia, para ello es recomendable hacer previamente una copia de seguridad de la base de datos.

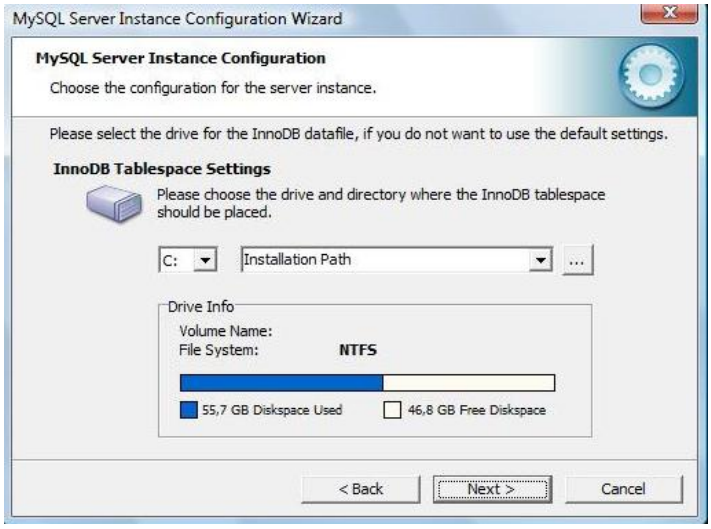
En este caso concreto escogemos la primera opción de forma que podamos optar por el uso uniforme de tablas con las dos tecnologías de almacenamiento.



Paso 4

Para el almacenamiento InoDB se precisa que escojamos el directorio de alojamiento para situar el espacio de almacenamiento de tablas (tablespace para los profesores que estén habituados al término con Oracle).

En este caso dejamos la ubicación por defecto.

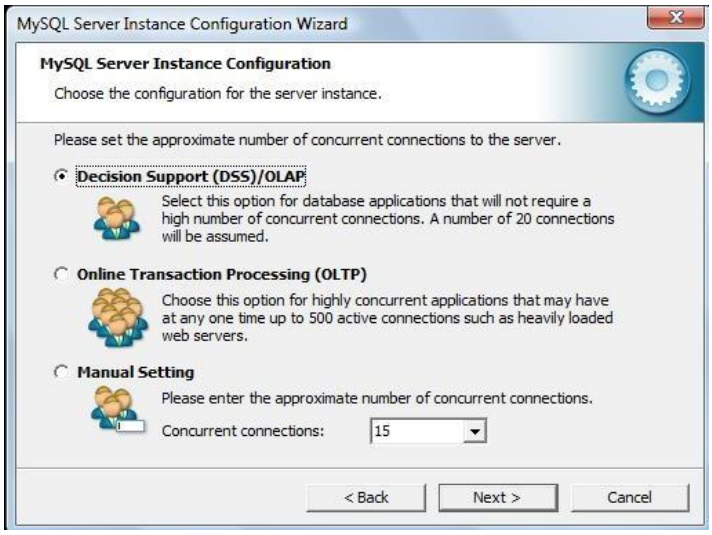


Paso 5

En la siguiente pantalla es necesario escoger el número de conexiones o accesos concurrentes posibles a los datos del servidor de bases de datos. Existen tres opciones disponibles:

- *Decision Support* a través del proceso OLAP, es la opción predeterminada. El control de conexiones se hará a través de un servicio de decisión optimizada que permite hasta 100 conexiones simultáneas con un promedio de 20 para un servicio óptimo.
- *Online Transaction Processing* soportado por el proceso OLTP que permite hasta 500 conexiones simultaneas.
- *Manual Setting* en el que el usuario decide cuántas conexiones se van a realizar de forma manual.

En nuestro caso hemos elegido la primera.

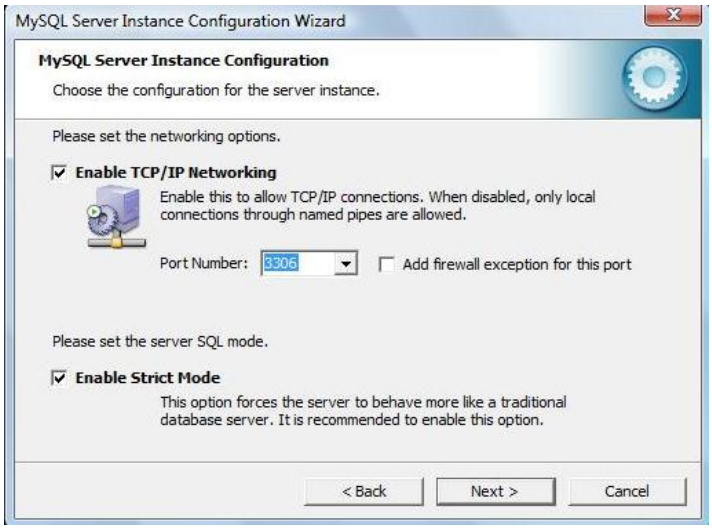


Paso 6

En esta pantalla, podemos elegir la configuración del acceso por red. Puede desactivarse el protocolo TCP/IP cuyo puerto de acceso por defecto es 3306. Se puede cambiar el puerto sobre todo si tenemos varios servicios MySQL ejecutándose en el mismo servidor. Además, en el caso de Windows, se puede añadir una regla al firewall del sistema para poder iniciar el servicio remoto de forma segura.

Por otro lado, en esta pantalla podemos elegir si queremos un comportamiento similar a los sistemas gestores tradicionales respecto al tratamiento del lenguaje SQL Estándar, aun escogiendo esta opción por defecto, MySQL es un sistema que utiliza el lenguaje SQL con cierta flexibilidad.

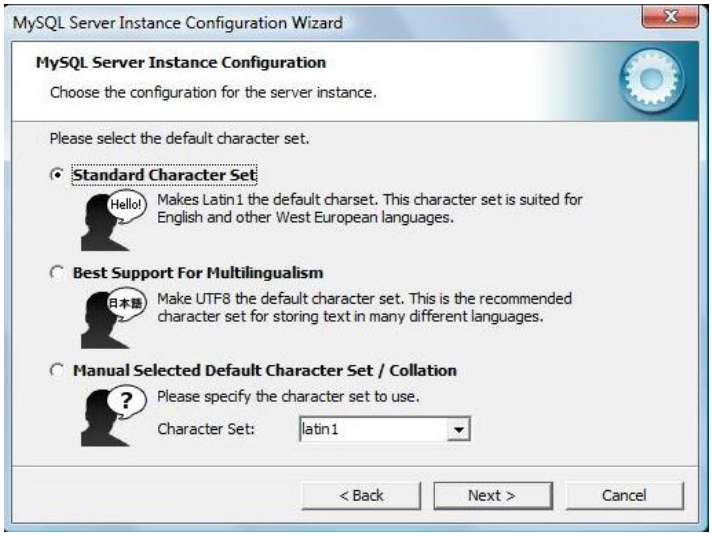
En este caso, imaginando que sólo tenemos instalada esta instancia de MySQL dejamos el puerto por defecto y habilitamos el modo estricto de SQL.



Paso7

Debemos escoger el juego o colección de caracteres de trabajo. Por defecto está activo Latin1 que permite todos los caracteres anglosajones y la mayoría de los lenguajes europeos de la Europa Occidental. En segunda opción podemos escoger el juego de caracteres UTF8 multilingüe y como alternativa escoger de forma manual uno de los juegos que están incluidos en la instalación.

En este caso dejamos la opción de Latin1 por defecto.



Paso 8

Este es el primer paso en común entre la configuración Estándar y la Detallada. En esta pantalla debemos configurar el servicio que como tal se va a instalar en Windows.

Podemos cambiar el nombre del servicio para que nos sea más sencillo identificarlo, y sobre todo si vamos a tener varios servicios MySQL corriendo en el mismo servidor. Una vez instalado podremos acceder a él a través del Sistema Operativo.

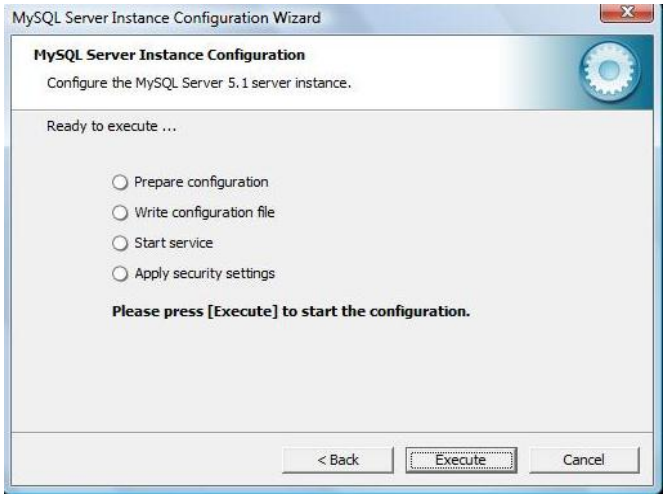
Paso 9

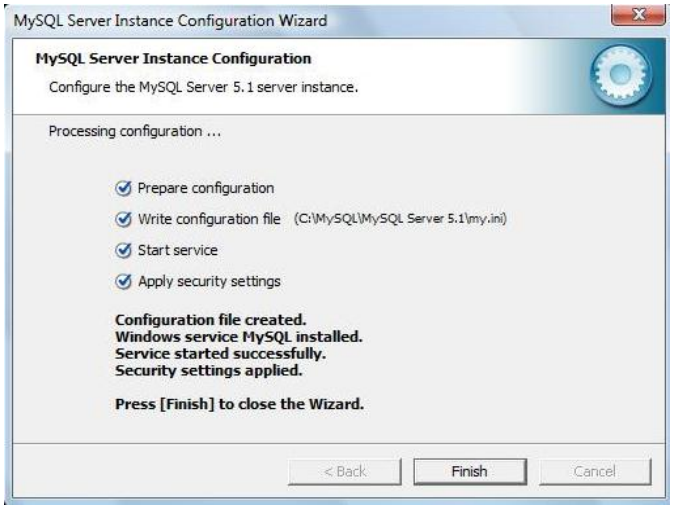
En este último paso se configura el modo de conexión. Es primordial introducir una contraseña para el usuario administrador root. Es posible crear otros usuarios administradores, pero la existencia de una cuenta root es obligatoria y además está creada por defecto. Podemos desactivar la opción de modificar la contraseña, pero sería crear un alto grado de vulnerabilidad en nuestra base de datos, por lo que es recomendable introducir la contraseña en este paso y establecer un protocolo de cambio de la misma según los parámetros de seguridad que se deseen implantar.

Paso 10

Una vez establecida la configuración, el asistente mostrará los pasos a seguir para configurar la instancia: Preparar la configuración, escribir el fichero de configuración (my.ini), iniciar el servicio sobre Windows y aplicar las opciones de seguridad. Para que estas tareas se lleven a cabo es preciso pulsar *Execute*.

Errores comunes que deben subsanarse antes de comenzar la instalación son no tener permisos de escritura sobre la carpeta o disco destino, tener un servicio MySQL ya ejecutándose por una instalación anterior o tener activado un firewall que impida aplicar la configuración de seguridad.

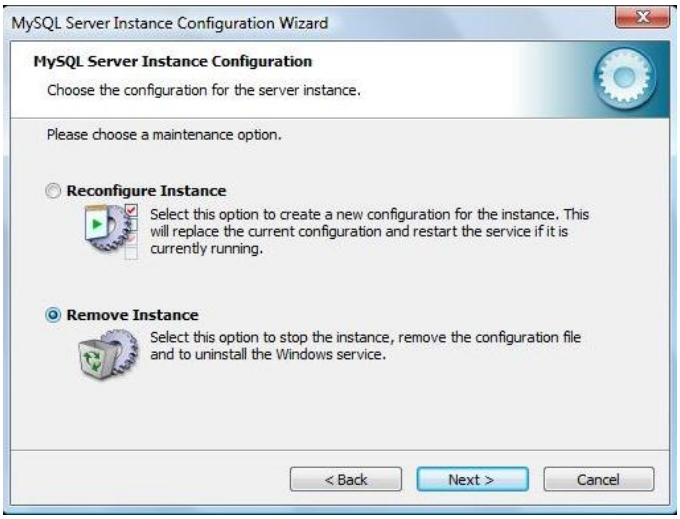




Mantenimiento y desinstalación

Es posible desinstalar el servicio o instancia instalada a través de la opción *MySQL Server Instance Configuration Wizard* que se ha creado en el menú de programas dentro de la carpeta del menú llamada MySQL

Si pulsamos reconfigurar (*Reconfigure Instance*) se iniciará la configuración desde el Paso 1. Si pulsamos *Remove* desinstalará el servicio y podremos instalarlo de nuevo desde la misma opción.



Para mantener el Servidor de MySQL podemos volver a ejecutar el paquete de instalación MySQL ESSENTIAL y nos mostrará la siguiente imagen. Podremos escoger modificar para cambiar los parámetros escogidos en la instalación, que realice una reparación automática o desinstale. Esto conlleva la eliminación completa de todos los archivos y el servicio de MySQL de nuestro sistema.

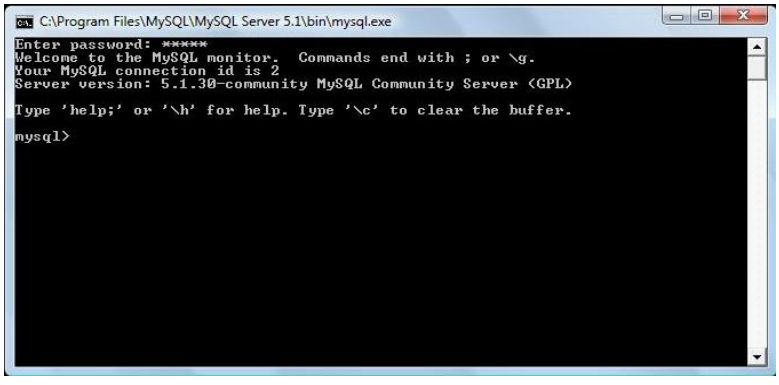
Modo de Trabajo

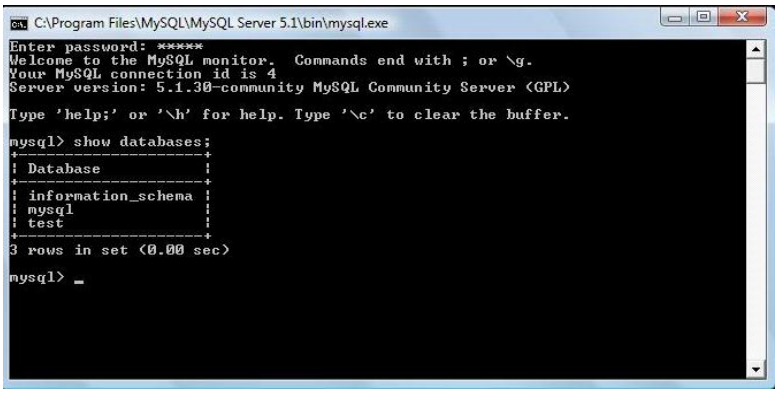
Si establecemos que el servicio MySQL arrancará automáticamente, este estará activo y podremos comprobar su funcionamiento en Panel de Control->Herramientas Administrativas-> Servicios. Podremos escoger su estado y su forma de arranque (automática o manual), así como iniciarlo o detenerlo.

Por otro lado se habrá creado un menú de acceso en el menú de programas que incluirá dos aplicaciones fundamentales:

- MySQL Server Instance Configuration Wizard opción de la cual hemos hablado en el apartado de Mantenimiento y desinstalación.
- MySQL Command Line Client: Abre la terminal cliente de acceso a MySQL Server a través de las ordenes propias de MySQL y de SQL estándar. Al ejecutarlo es preciso introducir la contraseña de root que establecimos en la Configuración, concretamente en el Paso 9.

Podemos ejecutar la sentencia *show databases* para que muestre los esquemas actuales, que en principio deben ser information_squema, MySQL y test. Los dos primeros esquemas pertenecen al sistema gestor de la base de datos, el esquema test está vacío y permite al administrador crear tablas y objetos de prueban en él.





Introducción a las Herramientas GUI Tools

Este paquete de instalación que también se puede adquirir de forma gratuita desde la página de descargas de la página oficial de MySQL, es un conjunto de herramientas que permiten la gestión y manipulación de la base de datos de forma sencilla y con variadas alternativas de funcionamiento.

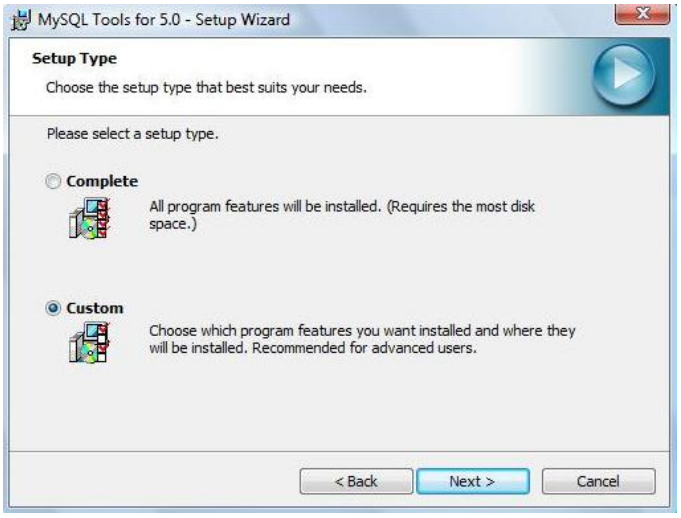
Se puede escoger el paquete de instalación asistida para Windows, al igual que hemos hecho con el Essential de MySQL. La instalación es realmente sencilla:

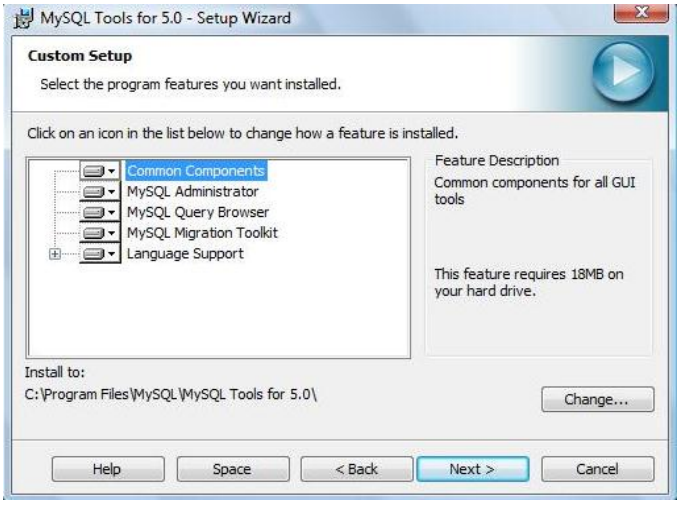
Paso 1

Ejecutamos el paquete de instalación MySQL-gui-tools, aceptamos las condiciones y escogemos el directorio de instalación que por defecto coincide con el predeterminado en la instalación del Server con MySQL Essentials.

Paso2:

Escogemos el tipo de instalación, la opción completa es recomendable. La personalizada nos permite escoger los componentes de forma separada. El botón *Space* permite comprobar el espacio en disco disponible y la opción *Change* cambiar el directorio de instalación de cada componente. Por defecto se instala todo.

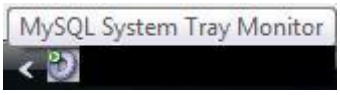




A continuación se describen cada una de estas herramientas brevemente, ya que especialmente, las herramientas Administrator y Query Browser, precisarían un tratamiento especial fuera de este artículo debido a la gran cantidad de alternativas y utilidades que ofrecen a la hora de hacer prácticas con los alumnos:

MySQL System Try Monitor:

Esta herramienta permite arrancar y parar el servicio, así como monitorizar y parametrizar las distintas instancias de MySQL si tenemos varios servidores instalados y además es un camino rápido para ejecutar el Administrator y la herramienta Query Browser. Al ejecutarlo aparece un pequeño icono en la parte inferior derecha de la barra de tareas de windows:



Al pulsar sobre él con el botón de la derecha del ratón aparece el siguiente menú contextual:

Diagrama de Clases