DISEÑO Y PROTOTIPO DE UNA APLICACIÓN CLIENTE-SERVIDOR QUE PERMITA EJECUTAR COMANDOS BÁSICOS EN UN SERVIDOR REMOTO DESDE UN DISPOSITIVO MÓVIL

REGINALDO BRAY MENDOZA
JAIRO IRIARTE VALENCIA

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
PROGRAMA DE INGENIERÍA DE SISTEMAS
CARTAGENA DE INDIAS
2006

DISEÑO Y PROTOTIPO DE UNA APLICACIÓN CLIENTE-SERVIDOR QUE PERMITA EJECUTAR COMANDOS BÁSICOS EN UN SERVIDOR REMOTO DESDE UN DISPOSITIVO MÓVIL

REGINALDO BRAY MENDOZA
JAIRO IRIARTE VALENCIA

Monografía para optar al título de
Ingeniero de Sistemas

Director
GIOVANNY VÁSQUEZ MENDOZA
Ingeniero de Sistemas

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
PROGRAMA DE INGENIERÍA DE SISTEMAS
CARTAGENA DE INDIAS
2006

Nota de aceptación

_____

_____

_____

_____

Firma del presidente del jurado

_____

Firma del jurado

_____

Firma del jurado

Cartagena de Indias, 24 de enero de 2006

Cartagena de Indias, 24 de enero de 2006

Señores
COMITÉ DE EVALUACIÓN DE PROYECTOS DE GRADO
Programa de Ingeniería de Sistemas
Universidad Tecnológica de Bolívar
La Ciudad

Cordial saludo.

A través de la presente me permito entregar la monografía titulada *"DISEÑO Y PROTOTIPO DE UNA APLICACIÓN CLIENTE-SERVIDOR QUE PERMITA EJECUTAR COMANDOS BÁSICOS EN UN SERVIDOR REMOTO DESDE UN DISPOSITIVO MÓVIL"*, para su estudio y evaluación, como requisito fundamental para obtener el titulo de Ingeniero de Sistemas.

En espera de que este cumpla con las normas pertinentes establecidas por la Institución me despido.

Atentamente,

REGINALDO BRAY MENDOZA

Cartagena de Indias, 24 de enero de 2006

Señores
COMITÉ DE EVALUACIÓN DE PROYECTOS DE GRADO
Programa de Ingeniería de Sistemas
Universidad Tecnológica de Bolívar
La Ciudad

Cordial saludo.

A través de la presente me permito entregar la monografía titulada *"DISEÑO Y PROTOTIPO DE UNA APLICACIÓN CLIENTE-SERVIDOR QUE PERMITA EJECUTAR COMANDOS BÁSICOS EN UN SERVIDOR REMOTO DESDE UN DISPOSITIVO MÓVIL"*, para su estudio y evaluación, como requisito fundamental para obtener el titulo de Ingeniero de Sistemas.

En espera de que este cumpla con las normas pertinentes establecidas por la Institución me despido.

Atentamente,

JAIRO IRIARTE VALENCIA

Cartagena de Indias, 24 de enero de 2006

Señores
COMITÉ DE EVALUACIÓN DE PROYECTOS DE GRADO
Programa de Ingeniería de Sistemas
Universidad Tecnológica de Bolívar
La Ciudad

Cordial saludo.

A través de la presente hago constar que he dirigido y asesorado la realización de la monografía titulada *"DISEÑO Y PROTOTIPO DE UNA APLICACIÓN CLIENTE-SERVIDOR QUE PERMITA EJECUTAR COMANDOS BÁSICOS EN UN SERVIDOR REMOTO DESDE UN DISPOSITIVO MÓVIL"*, desarrollada por los estudiantes REGINALDO BRAY MENDOZA y JAIRO IRIARTE VALENCIA.

Atentamente,

GIOVANNY VÁSQUEZ MENDOZA
Ingeniero de Sistemas

AUTORIZACIÓN

Cartagena de Indias, 24 de enero de 2006

Yo, **Reginaldo Bray Mendoza**, identificado con la cédula de ciudadanía número 9'103.390 de Cartagena, autorizo a la Universidad Tecnológica de Bolívar para hacer uso de mi trabajo de grado y publicarlo en el catálogo *online* de su Biblioteca.

Atentamente,

REGINALDO BRAY MENDOZA

AUTORIZACIÓN

Cartagena de Indias, 24 de enero de 2006

Yo, **Jairo Iriarte Valencia**, identificado con la cédula de ciudadanía número 9'145.622 de Cartagena, autorizo a la Universidad Tecnológica de Bolívar para hacer uso de mi trabajo de grado y publicarlo en el catálogo *online* de su Biblioteca.

Atentamente,

JAIRO IRIARTE VALENCIA

AGRADECIMIENTOS

Al ingeniero Giovanny Vásquez, por su valiosa ayuda, su empeño, colaboración, atención y dedicación, factores que influyeron favorablemente en la realización del proyecto.

A Moisés Quintana, por su apoyo e inmensos aportes a nuestra formación profesional a lo largo de la carrera.

A Gonzalo Garzón, Isaac Zúñiga, y todos aquellos docentes que de una u otra forma influyeron positivamente en nuestra formación profesional, académica y personal.

TABLA DE CONTENIDO

TABLA DE FIGURAS

GLOSARIO DE TÉRMINOS

API: *Application Programming interface* (interfaz de desarrollo de programas). Conjunto de convenciones de programación que definen cómo se invoca un servicio desde un programa.

CDC: *Connected Device Configuration*. Configuración J2ME orientada a dispositivos con CPU de 32 bits y un mínimo de 32 Mbytes de memoria para la plataforma y las aplicaciones.

CLDC: *Connected Limited Device Configuration*. Especifica un conjunto mínimo de paquetes y clases, y una máquina virtual Java de funcionalidad reducida. Diseñada para dispositivos con conexiones de red intermitentes, un procesador lento y memoria limitada (teléfonos móviles y PDAs, por ejemplo), CPU de 16 o 32 bits y de 128 a 256 Kbytes de memoria disponible para la implementación de Java y las aplicaciones

CONFIGURACIONES: Están compuestas por una máquina virtual y un conjunto mínimo de bibliotecas de clases que definen un conjunto de dispositivos comunes.

CVM: *C Virtual Machine*. Máquina virtual de JAVA para la configuración CDC.

DISPOSITIVO MÓVIL: Es un aparato pequeño de alta movilidad operado con baterías, el cual es capaz de capturar, procesar, presentar y transmitir información.

GARBAGE COLLECTOR: Es un hilo de ejecución que se encarga de devolver la memoria al sistema operativo (recolector de basura).

GENERIC CONNECTION FRAMEWORK (GCF): Define un conjunto de abstracciones relacionadas para representar diferentes métodos de comunicación en J2ME.

GPRS: *General Packet Radio Service*, es una tecnología digital de telefonía móvil. Es considerada la generación 2.5, entre la segunda generación (GSM) y la tercera (UMTS). Proporciona altas velocidades de transferencia de datos (especialmente útil para conectar a Internet) y se utiliza en las redes GSM.

GSM: *Global System for Mobile communications* (Sistema Global para las Comunicaciones Móviles), formalmente conocida como *"Group Special Mobile"* (GSM, Grupo Especial Móvil) es un estándar mundial para teléfonos móviles digitales.

HTTP: El protocolo de transferencia de hipertexto (*HyperText Transfer Protocol*) es el protocolo usado en cada transacción de la web. El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceder a ellas, y la respuesta de éstas, remitiendo la información que se verá en pantalla.

JAD: Es un archivo que contiene una descripción de una aplicación o MIDlet, normalmente acompaña al JAR.

JAM: *Java Application Manager*. Es un gestor que controla la descarga, instalación, lanzamiento y desinstalación de aplicaciones en el dispositivo móvil.

JAR: Es un archivo que contiene una o varias clases Java comprimidas.

JAVA: Lenguaje de programación orientado a objetos creado por la *Sun Microsystems*.

JAVA 2 STANDARD EDITION (J2SE): Edición de Java diseñada para computadoras de escritorio, puede trabajar en sistemas operativos como: Windows, Linux, MacOS, Solaris, OS x.

JAVA 2 ENTERPRISE EDITION (J2EE): Edición de java para aplicaciones multiusuario o empresariales. Se basa en J2SE y agrega APIs para trabajos en servidor.

JAVA 2 MICRO EDITION (J2ME): Conjunto de tecnologías y especificaciones desarrolladas para dispositivos pequeños como los teléfonos celulares y PDA (palm, agendas electrónicas, etc.). Utiliza derivados de componentes J2SE, como son: una máquina virtual más pequeña y un conjunto de APIs menos potentes.

JNI: Es un mecanismo que nos permite ejecutar código nativo desde Java y viceversa.

JSR185: Es la especificación vigente para el desarrollo de aplicaciones inalámbricas en Java.

JVM: *Java Virtual Machine*. Máquina virtual de Java utilizada en las ediciones J2EE y J2SE.

KVM: *Kilobyte Virtual Machine*. Máquina virtual de Java para la configuración CLDC.

MÁQUINA VIRTUAL: Interprete que ejecuta el *bytecode* generado por el compilador Java.

MIDlet: Es una aplicación J2ME capaz de correr en un dispositivo Móvil.

MIDP: Perfil diseñado para teléfonos móviles y PDAs.

MIME: Siglas de *Multipurpose Internet Mail Extension*. Sistema que permite integrar dentro de un mensaje de correo electrónico ficheros binarios (imágenes, sonido, programas ejecutables, etc.).

PDA: *Asistentes Digitales Personales*. Dispositivos móviles que son una mezcla entre organizador personal y ordenador portátil.

PERFILES: Conjunto de APIs de alto nivel que definen el modelo de ciclo de vida de la aplicación, el interfaz de usuario y el acceso a las propiedades específicas del dispositivo móvil.

PREVERIFICACIÓN: Proceso que asegura que no existe ningún tipo de código malintencionado que pueda dañar o crear un comportamiento extraño en el dispositivo o en la máquina virtual. Se hace externo al dispositivo.

REFERENCIAS DÉBILES: Las referencias débiles se utilizan para poner mapas en ejecución débiles. Un objeto que está no fuertemente o suavemente accesible sino es referido por una referencia débil se llama débil accesible. Un objeto débil accesible será basura recogida durante el ciclo siguiente de la colección.

TCP/IP: Son las siglas de *Transmission Control Protocol/Internet Protocol*, el lenguaje que rige todas las comunicaciones entre todos los ordenadores en Internet. TCP/IP es un conjunto de instrucciones que dictan cómo se han de enviar paquetes de información por distintas redes.

UML: Lenguaje Unificado de Modelado (*Unified Modelling Language*). Es el lenguaje de modelado de sistemas de software más conocido en la actualidad; aún cuando todavía no es un estándar oficial, está apoyado en gran manera por la OMG (*Object Management Group*).

UMTS: La tecnología UMTS (*Universal Mobile Telecommunications System*) es el sistema de telecomunicaciones móviles de tercera generación, que evoluciona desde GSM pasando por GPRS.

UNICODE: Es un tipo de codificación que provee un único número para cada carácter, sin importar la plataforma, el programa o el lenguaje.

WAP: *Wireless Application Protocol* (protocolo de aplicaciones inalámbricas) es un estándar abierto internacional para aplicaciones que utilizan las comunicaciones inalámbricas. Provee una especificación de un conjunto de protocolos de comunicaciones para estandarizar el modo en que los dispositivos inalámbricos se pueden utilizar para acceder a correo electrónico, grupo de noticias y otros.

WML: *Wireless Markup Language*. Lenguaje más ligero y efectivo para mostrar contenidos y servicios en los dispositivos móviles.

RESUMEN

TÍTULO

Diseño y prototipo de una aplicación cliente-servidor que permita ejecutar comandos básicos en un servidor remoto desde un dispositivo móvil.

ÁREA DE INVESTIGACIÓN

Ingeniería de software.

COBERTURA DE INVESTIGACIÓN

Institucional.

CAMPO DE INVESTIGACIÓN

- Desarrolladores Java (J2ME)
- Administradores de aplicaciones
- Ingenieros de sistemas

BREVE DESCRIPCIÓN DEL PROBLEMA

Las necesidades de los usuarios de telefonía móvil han cambiado mucho en estos últimos años y cada vez demandan más servicios y prestaciones por parte tanto de los dispositivos como de las compañías. Es por esta razón que surge J2ME, la tecnología para la industria del desarrollo de software en dispositivos móviles.

La aparición y asentamiento de otras tecnologías como GPRS, SMS, etc., ha permitido que la gama de aplicaciones para telefonía móvil aumente y, por lo tanto, aumente la utilización de la tecnología java J2ME.

El desarrollo de aplicaciones de telefonía móvil que permitan comunicarse con máquinas o dispositivos remotos e incluso manipularlos, ha ido aumentando a medida que se han descubierto este tipo de necesidades que cada vez son mucho más complejas y que han

permitido el crecimiento de este mercado. Es así como se hace necesario dominar el diseño y desarrollo de este tipo de aplicaciones y estudiar más a fondo estas necesidades en particular.

La ejecución de comandos en un servidor conectado a la Internet, muchas veces sólo es imaginable realizarla desde una estación de trabajo igualmente conectada a la red y con acceso a él (mediante distintos protocolos vía TCP/IP o sockets), o desde el mismo servidor. Ahora, gracias a J2ME, es posible cubrir necesidades o tareas específicas desde un dispositivo móvil con todas las ventajas que la tecnología móvil ofrece.

OBJETIVOS

General

Diseñar y crear un prototipo de una aplicación cliente-servidor que permita ejecutar comandos básicos en un servidor remoto desde un dispositivo móvil.

Específicos

- Realizar el diseño en UML de la aplicación
- Desarrollar un prototipo de la aplicación utilizando Java
- Estudiar las tecnologías de redes celulares en Colombia que permiten conexión a la Internet a través del dispositivo móvil.

JUSTIFICACIÓN

La posibilidad de realizar tareas en una máquina o servidor remoto desde un dispositivo móvil, es una de las tareas que es posible realizar hoy en día y que facilita el trabajo diario a administradores de redes, aplicaciones y de servidores en general.

El hecho de no importar la localización física de la persona ni del equipo al que se quiere acceder, la posibilidad de utilizar un dispositivo móvil de fácil manipulación y transporte, y de integrar tecnologías diferentes, hace que nos centremos en este tema para desarrollar nuestra monografía.

TIPO DE INVESTIGACIÓN

Experimental

AUTORES

Jairo Iriarte Valencia

Reginaldo Bray Mendoza

DIRECTOR

Ing. Giovanny Vásquez Mendoza

INTRODUCCIÓN

Las necesidades de los usuarios de telefonía móvil han cambiado mucho en estos últimos años y cada vez demandan más servicios y prestaciones por parte tanto de los dispositivos como de las compañías. Es por esta razón que surge J2ME, la tecnología para la industria del desarrollo de software en dispositivos móviles.

La aparición y asentamiento de otras tecnologías como GPRS, SMS, etc., ha permitido que la gama de aplicaciones para dispositivos móviles sea cada vez mayor, y por lo tanto, ha aumentado la utilización de la tecnología Java J2ME.

Satisfacer las necesidades de las empresas y usuarios particulares, requiere el desarrollo de aplicaciones cada vez más complejas, capaces de comunicar móviles con estaciones remotas, servidores, etc. Es así como se hace necesario dominar el diseño y desarrollo de este tipo de aplicaciones y estudiar más a fondo estas necesidades en particular.

La ejecución de comandos en un servidor conectado a la Internet, muchas veces sólo era imaginable realizarla desde una estación de trabajo conectada a la misma red y con acceso a él (mediante distintos protocolos TCP/IP o sockets), o desde el mismo servidor. Ahora, gracias a J2ME, es posible cubrir tareas específicas desde un dispositivo de mano o de bolsillo con todas las ventajas que la tecnología móvil ofrece.

OBJETIVOS

GENERAL

Diseñar y crear un prototipo de una aplicación cliente-servidor que permita ejecutar comandos básicos en un servidor remoto desde un dispositivo móvil.

ESPECÍFICOS

- Estudiar el trabajo con redes en J2ME.
- Analizar los requerimientos al desarrollar aplicaciones móviles con J2ME.
- Realizar el diseño en UML de la aplicación.
- Desarrollar un prototipo de la aplicación utilizando J2ME Wireless Toolkit.

JUSTIFICACIÓN

En los últimos años ha habido un incremento considerable en el mercado de dispositivos móviles y, en consecuencia, en el desarrollo de software para ellos. Los teléfonos móviles se están convirtiendo en el centro de comunicaciones de cada persona y los servicios de valor agregado crecen a pasos agigantados por lo que la tecnología inalámbrica se está convirtiendo cada vez en algo más asequible y en parte esencial del que hacer diario.

Realizar o ejecutar tareas en una máquina o servidor remoto desde un dispositivo móvil, es una de las tareas posibles de realizar hoy en día, y que facilita el trabajo diario a administradores de redes, aplicaciones y de servidores en general.

El hecho de no importar la localización física de la persona ni del equipo al que se quiere acceder, la posibilidad de utilizar un dispositivo móvil de fácil manipulación y transporte, y de integrar tecnologías diferentes, es lo que motiva a desarrollar el prototipo.

1. MARCO TEÓRICO

1.1. LA PLATAFORMA JAVA

Java fue creado por Sun Microsystems en un intento por desarrollar un lenguaje que facilitara la programación de aparatos electrodomésticos conectados a la red de una casa inteligente. Después de un tiempo de desarrollo, la directiva del proyecto vio que las capacidades del lenguaje iban mucho más allá de una red casera y decidieron enfocarlo hacia las aplicaciones basadas en Internet o con cierto nivel de desempeño requerido en trabajos de redes, además de satisfacer a las empresas que demandaban un lenguaje que pudiera ser implementado en todas sus máquinas, sin importar el sistema operativo sobre el que trabajaran.[1]

Aunque el lenguaje Java es sintácticamente similar a C++, difiere enormemente en su fundamento:

- Java es un lenguaje puramente orientado a objetos.

- Java es un lenguaje interpretado; es decir, que los programas no son ejecutables por si solos, sino que existe otro programa que los interpreta cada vez que son requeridos. Esto puede causar disminución en la velocidad de ejecución de las aplicaciones, pero, aunque aún no se alcanza la rapidez de ejecución de los programas hechos en C++ o Visual BASIC, se ha mejorado enormemente con respecto a las primeras versiones del lenguaje, lo que significa que tal vez en un futuro cercano la rapidez deje de ser un impedimento para usar Java.

[1] Desarrollo de aplicaciones para dispositivos inalámbricos (J2ME).
http://www.mailxmail.com/curso/informatica/dispositivosinalambricos/capitulo1.htm

- C++ utiliza apuntadores a memoria como característica principal, y delega a los programadores, simples seres humanos, la tarea de localizarla, controlarla y liberarla. Mientras tanto, Java utiliza objetos de tipo seguro, no permite la asignación de memoria dinámica y, gracias a su recolector de basura "*garbage collector*", la memoria sin usar se recicla automáticamente.

La base de la plataforma Java es una máquina virtual, la cual puede ser implementada en los más populares sistemas operativos y en gran variedad de hardware. Por lo que se puede tener aplicaciones binarias Java operando consistentemente a través de diferentes implementaciones.

Las APIs (*Application Programming Interface*) Java son el conjunto de clases y objetos que permiten, mediante la utilización del lenguaje Java, la interacción entre el programador y la computadora y entre esta y el usuario final.

Juntos, el lenguaje de programación Java, la máquina virtual y las APIs, forman la plataforma Java, la cual en su versión 2, se puede encontrar en tres ediciones:

- *Java 2 Standard Edition* (J2SE). Diseñada para computadoras de escritorio, puede trabajar en sistemas operativos como: Windows, Linux, MacOS, Solaris, OS x.

- *Java 2 Enterprise Edition* (J2EE). Plataforma para aplicaciones multiusuario o empresariales. Se basa en J2SE y agrega APIs para trabajos en servidor.

- *Java 2 Micro Edition* (J2ME). Conjunto de tecnologías y especificaciones desarrolladas para dispositivos pequeños como los teléfonos celulares y PDA (palm, agendas electrónicas, etc.). Utiliza derivados de componentes J2SE, como son: una máquina virtual más pequeña y un conjunto de APIs menos potentes.

Figura 1. Ediciones de Java y dispositivos aplicables

1.2. JAVA 2 MICRO EDITION (J2ME)

A diferencia de J2SE, la micro edición de Java no es sólo una pieza de software ni una simple especificación; J2ME es una plataforma, una colección de tecnologías y especificaciones diseñadas para diferentes partes del mercado de los dispositivos pequeños. J2ME se divide en configuraciones, perfiles y paquetes opcionales.[2]

1.2.1. Configuraciones

Las configuraciones están compuestas por una máquina virtual y un conjunto mínimo de bibliotecas de clases, las cuales serían un mínimo denominador común con que contarán todos los dispositivos de una configuración dada; o lo que es lo mismo, ofrecen la funcionalidad para un rango particular de dispositivos con características comunes.

[2] Desarrollo de aplicaciones para dispositivos inalámbricos (J2ME). Capítulo 1. http://www.mailxmail.com/curso/informatica/dispositivosinalambricos/capitulo1.htm

Existen dos configuraciones[3] actualmente: *Connected Limited Device Configuration* (CLDC) y *Connected Device Configuration* (CDC):

- **CLDC** es la más pequeña de las dos. Diseñada para dispositivos con conexiones de red intermitentes, un procesador lento y memoria limitada (teléfonos móviles y PDAs, por ejemplo), CPU de 16 o 32 bits y de 128 a 256 Kbytes de memoria disponible para la implementación de Java y las aplicaciones. CLDC está estrechamente asociado a lo que se conoce como Java inalámbrico (*Wireless Java*), es decir, a la posibilidad de que teléfonos móviles puedan descargar aplicaciones Java, conocidas como MIDlets.

- **CDC** está orientado a dispositivos con más memoria, procesadores más rápidos y un ancho de banda mayor (TV set-top boxes); es decir, los aparatos que están entre los que trata CLDC y los ordenadores de escritorio. Esta configuración incluye una máquina virtual completa y un subconjunto de J2SE mayor. Los dispositivos a los cuales se dirige esta configuración tienen CPU de 32 bits y un mínimo de 32 Mbytes de memoria para la plataforma y las aplicaciones.

En la siguiente figura podemos ver un cómo se relacionan las dos configuraciones existentes y también con respecto a J2SE.

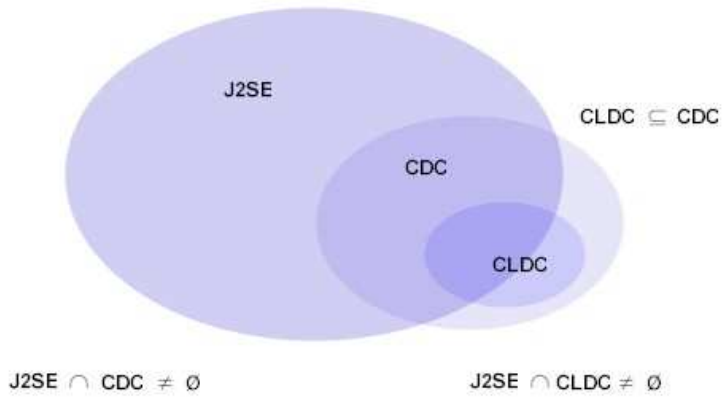

Figura 2. Relación entre CLDC, CDC y J2SE

[3] Programación de dispositivos móviles con Java. http://leo.ugr.es/~fjgc/INTRO/intro_2b.htm

Las especificaciones de las configuraciones no indican ningún tipo de implementación de la máquina virtual, por lo que cada cual puede crear sus propios entornos de ejecución. Los que ha desarrollado Sun son: *Kilobyte Virtual Machine* (KVM) para CLDC, y *C Virtual Machine* (CVM).

1.2.2. Perfiles

Con objeto de ofrecer un completo entorno de ejecución específico para cada categoría de dispositivo, las configuraciones se deben combinar con un conjunto de APIs de alto nivel, conocidas como perfiles[4], que definen el modelo de ciclo de vida de la aplicación, la interfaz de usuario y el acceso a las propiedades específicas del dispositivo. Algunos de los perfiles existentes son:

- **Mobile Information Device Profile (MIDP)**. Diseñado para teléfonos móviles y PDAs. Incluye la interfaz de usuario, conectividad de red, almacenamiento local de datos y gestión de aplicaciones. Combinado con CLDC, MIDP aporta un entorno de ejecución para Java que minimiza los consumos de memoria y de procesador.

- **Foundation Profile (FP)**. Este perfil es el de nivel más bajo asociado a CDC, ya que los perfiles CDC pueden ser vistos como capas que se añaden a CLDC para proveer funcionalidades a los diferentes dispositivos. FP suministra una implementación de CDC con capacidades de acceso a red que se utiliza para aplicaciones embedidas en alto grado y sin interfaz de usuario.

- **Personal Profile (PP)** es un perfil diseñado expresamente para dispositivos con interfaz gráfica completa y con posibilidad de ejecución de Applets. Incluye además las bibliotecas del *Abstract Windows Toolkit* (AWT). Añade, por tanto, una interfaz de usuario básico a FP.

[4] Programación de dispositvos móviles con Java. http://leo.ugr.es/~fjgc/INTRO/intro_2b.htm

- **Personal Basis Profile (PBP)** es un subconjunto de PP que suministra un entorno para dispositivos que se puedan conectar a una red y que dispongan de una interfaz un poco más desarrollada que aquellos que poseen los dispositivos a donde va dirigido PP.

- **PDA Profile (PDAP)** es similar al MIDP pero diseñado para PDAs que tengan mejores pantallas y más memoria que los teléfonos móviles.

- **Game Profile (GP)** ofrece la plataforma para escribir juegos en dispositivos CDC.

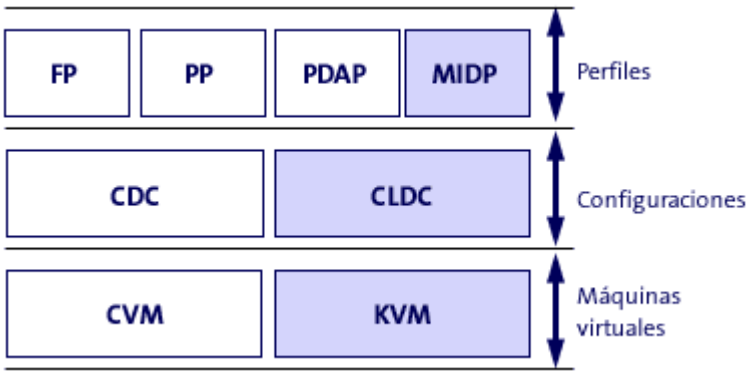

Figura 3. Esquema modular para los perfiles, configuraciones y máquinas virtuales

Las aplicaciones desarrolladas para un determinado perfil serán portables a cualquier dispositivo que soporte ese perfil. Cabe destacar también que un mismo dispositivo puede soportar varios perfiles y que sobre una configuración también pueden residir diversos perfiles.

1.2.3. Mobile information device profile (MIDP)

MIDP 2.0 incorpora APIs de interfaz de usuario, de ciclo de vida del programa, almacenamiento persistente, juegos, trabajo en red y multimedia[5]. Según la especificación

[5] Desarrollo de aplicaciones para dispositivos inalámbricos (J2ME). Capítulo 4. http://www.mailxmail.com/curso/informatica/dispositivosinalambricos/capitulo4.htm

de la tecnología inalámbrica de Java, todo dispositivo que soporte MIDP 2.0 debe incluir mínimo las siguientes características:

- Debe permitir archivos Java (JAR) de más de 64 KB y archivos descriptores de aplicaciones (JAD) mayores a 5 KB.

- Se debe permitir a cada MIDlet (aplicación MIDP) la utilización de 30 KB de almacenamiento persistente y se recomienda que las MIDlets incluyan información acerca del almacenamiento mínimo con el que trabajan correctamente.

- El espacio libre de memoria para una aplicación ejecutándose (*Heap* o del montón) debe ser de por lo menos 256 KB.

- Soporte para pantallas de 125 x 125 píxeles, con una profundidad de color de 12 bits.

- Se deben incluir la capacidad de que el dispositivo reaccione a eventos de tiempo (una alarma a determinada hora, los llamados ticklers o despertadores).

- Mecanismos para tomar un número telefónico del directorio del equipo.

- Soporte para imágenes en formato JPEG y PNG.

- Acceso a contenidos multimedia por el protocolo HTTP 1.1.

Las clases que contiene son:

- *javax.microedition.midlet*: se ocupa del ciclo de vida de la aplicación.
- *javax.microedition.lcdui*: interfaz de usuario.
- *javax.microedition.rms*: sistema de mantenimiento de registros (*Record Management System*) usado para guardar información.
- *javax.microedition.io*: clases para usar redes.
- java.lang: clases de lenguaje.

- java.util: clases de utilidades.

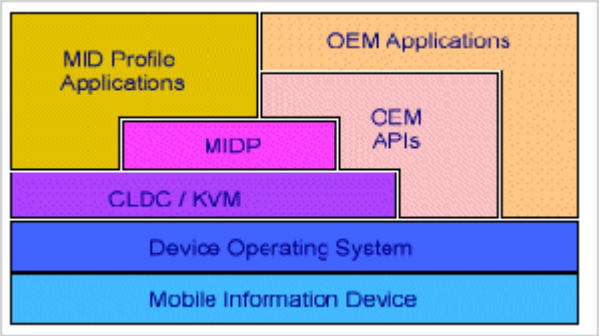

Figura 4. Arquitectura del MIDP

Ciclo de vida del MIDP

El ciclo de vida de un MIDP está muy bien definido, ya que ayuda al MIDlet a coexistir con otros programas en el MIDP[6]. Las fases del ciclo de vida son:

- Retrieval
  El teléfono consigue la aplicación desde la fuente, puede ser vía IrDA, Bluetooth o Internet. En este momento el MIDlet y MID establecen una comunicación para intercambiar información sobre el sistema y la aplicación, y para decidir así si se procede a instalarlo o no.

- Installation
  La aplicación se instala en el MID. La implementación de MIDP verifica que el nuevo MIDlet no viola la seguridad de MID.

- Launching
  El usuario ejecuta la aplicación. En esta fase, el MIDlet se ejecuta en la KVM y los métodos de ciclos de vida del MIDlet son ejecutados.

[6] Iniciación a J2ME. http://www.todosymbian.com/secart36.html

- MIDlet Instance creation – Paused
- MIDlet initialization – Active
- MIDlet termination – Destroyed

- Version Management

  El teléfono mantiene una base de datos sobre que programas han sido instalados y su versión. Así, usando la descripción (descriptor) del MIDlet puede ser actualizado si aparece una nueva versión.

- Removal

  El MIDlet es borrado del teléfono.

1.2.4. Aplicaciones MIDP: MIDlets

Un MIDlet es una aplicación escrita especialmente para el perfil MIDP de J2ME.

Ciclo de vida de un MIDlet

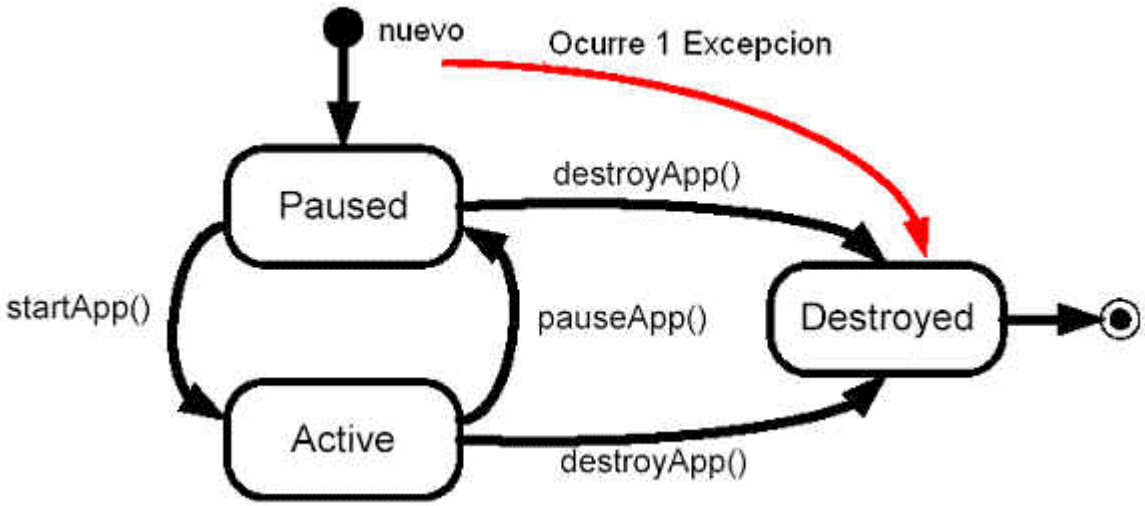

Figura 5. Ciclo de vida de un MIDlet

- **startApp()**
  - El método *setCurrent()* debería ser llamado aquí si no fue llamado antes. *setCurrent()* define qué *display* será visible al usuario, sólo un *display* puede ser visible al mismo tiempo.
  - Puede ser ejecutado más de una vez, así que es mejor no poner ninguna inicialización aquí.

- **pauseApp()**
  Cuando una aplicación es reactivada, puede aparecer con otro *display* usando el método *setCurrent()*.

- **destroyApp()**
  Libera y destruye todos los recursos usados por las aplicaciones, incluidos los componentes de la interfaz de usuario.

Estos tres métodos tienen que aparecer siempre, por lo tanto una aplicación MIDlet mínima puede ser:

```
import javax.microedition.midlet.*;

public class HelloMIDlet extends MIDlet {
 public void startApp() { }

 public void pauseApp() { }

 public void destroyApp(boolean unconditional) { }
}
```

1.2.5. Connected Limited Device Configuration (CLDC)

CLDC es la base para que los perfiles (como MIDP o PDAP) funcionen, proveyendo las APIs básicas y la máquina virtual (KVM). CLDC está diseñada para equipos

microprocesadores RISC o CISC de 16 a 32 bits y con una memoria mínima de 160 KB para la pila de la tecnología Java[7].

La *JSR185* pide como requisitos mínimos para todo equipo que implemente CLDC 1.0 o 1.1:

- Soporte mínimo para diez hilos relacionados con aplicaciones (MIDlets).

- Usar zonas de tiempo personalizables, con referencia en el formato de zonas de tiempo GMT (GMT-5:00, por ejemplo).

- Soporte para propiedades de carácter y conversiones mayúsculas-minúsculas en los bloques suplementales Unicode para Basic Latin y Latin-1 (nuestros caracteres).

Las clases obtenidas de la versión de J2SE son:

- java.lang.*
- java.io.*
- java.util.*

Nuevas clases son:

- java.microedition.io.*

1.2.6. Kilobyte Virtual Machine (KVM)

Como ya se explicó anteriormente, la máquina virtual es la base de la plataforma Java. En el caso de la plataforma J2ME, debido a las capacidades limitadas de almacenamiento, memoria, procesamiento y pantalla de los dispositivos, no se puede integrar una máquina virtual Java (JVM) del las dimensiones de J2SE o J2EE. Por esto se ha creado una nueva máquina virtual: KVM (Kilobyte Virtual Machine).

---

[7] Desarrollo de aplicaciones para dispositivos inalámbricos (J2ME). Capítulo 4. http://www.mailxmail.com/curso/informatica/dispositivosinalambricos/capitulo4.htm

KVM es una máquina virtual Java compacta y portable específicamente diseñada para ser la base de desarrollo en dispositivos pequeños y de recursos limitados. Actualmente CLDC trabaja sobre KVM. Además KVM está diseñada para mantener los aspectos centrales del lenguaje Java ejecutándose en unos cuantos kilobytes de memoria (de ahí su nombre).

Aunque KVM deriva de la máquina virtual J2SE (JVM), algunas características de esta última han sido eliminadas para soportar CLDC, debido a que resultan demasiado costosas de implementar o su presencia supone problemas de seguridad, resultando una KVM con las siguientes limitantes:

- Soporte de punto flotante. KVM no soporta números de punto flotante, esto debido a que la mayoría de los dispositivos en los que se implementa no lo soportan tampoco.

- Finalización. Las APIs CLDC no incluyen el método *object.finalize*, así que no se pueden hacer operaciones de limpieza final antes de que el recolector de basura tome los objetos.

- Manejo de errores. CLDC sólo define tres clases error: *java.lang.Error*, *java.lang.OutOfMemoryError* y *java.lang.VirtualMachineError*. Todo tipo de errores que no sean en tiempo de ejecución se manejan de modo dependiente del dispositivo, esto incluye la finalización de una aplicación o reinicio del dispositivo.

- Interfaz Nativa Java (JNI). No se implementa JNI (posibilidad de incluir código en C dentro de clases Java), primeramente por motivos de seguridad, aunque también es considerado excesivo dadas las limitantes de memoria del dispositivo al que se dirige.

- Cargadores de clases definidas por el usuario. KVM debe tener un cargador de clases que no pueda ser manipulado o remplazado por el usuario, principalmente por razones de seguridad.

- No hay soporte para la API *reflection*. La API *reflection* es más bien usada para aplicaciones de bajo nivel (depuradores, constructores GUI, etc.). La falta de esta API impide también la serialización de objetos.

- Grupos de hilos o “demonios". Aunque CLDC soporta programación multihilo, no soporta grupos de hilos o “demonios". Si se requiere usar operaciones para grupos de hilos hay que usar objetos de colección para almacenar los objetos hilo a nivel de aplicación.

- Referencias débiles. Ninguna aplicación construida con CLDC puede requerir referencias débiles.

En la máquina virtual de J2SE, existe un verificador de clases quien es el responsable de rechazar archivos de clase no válidos. Una máquina virtual que soporte CLDC (ahora KVM) también debe ser capaz de rechazar estos archivos. Sin embargo, el proceso de verificación de clases es tardado y costoso y, por lo tanto, no recomendable para equipos con recursos limitados.

Los diseñadores de KVM decidieron mover la mayor parte del trabajo de verificación de clases fuera del dispositivo, es decir, hacia la computadora de escritorio donde las clases son compiladas o el servidor de donde se descargan. A este proceso de verificación se le llama preverificación. Los dispositivos son únicamente responsables de ejecutar algunas pruebas en la clase preverificada para asegurarse de que aún es válida.

1.3. J2ME Y TRABAJO EN RED

Las capacidades de Java para trabajo en red son bastante robustas. J2SE define alrededor de cien interfaces, clases y excepciones en los paquetes *java.io* y *java.net*. La funcionalidad disponible a través de estas librerías es grandiosa para sistemas computacionales tradicionales con capacidad de CPU sustancial, memoria activa y

almacenamiento persistente de datos, pero no para la mayoría de los dispositivos inalámbricos.

Por lo tanto, J2ME define un subconjunto de esta funcionalidad y provee un paquete consolidado para el trabajo en red y acceso a archivos (el paquete *javax.microedition.io*). Debido a la amplia variedad de dispositivos móviles, este paquete sólo define un conjunto de interfaces, dejando la implementación a cada fabricante. Esto permite un balance óptimo entre la portabilidad y la utilización de características específicas de cada dispositivo.

La abstracción del trabajo en red y de la entrada-salida con archivos definida por las clases del paquete *javax.microedition.io* se conocen como el *Generic Connection Framework* (GCF). El GCF define un conjunto de abstracciones relacionadas para representar diferentes métodos de comunicación. El nivel más alto de abstracción se llama *Connection*, del cual seis interfaces más son declaradas (cuatro directas y dos indirectas). Estas siete interfaces son declaradas como parte del CLDC de J2ME, el cual es la configuración más usada por los dispositivos compatibles con Java. Se diseña para proveer capacidades comunes de trabajo en red y E/S de archivos para todos los dispositivos CLDC (celulares, PDAs, etc.).

Aunque el propósito detrás del GCF es proveer un marco para el trabajo en red y la E/S de archivos, los fabricantes no están obligados a implementar todas las interfaces declaradas en el GCF. Algunos pueden decidir soportar sólo conexiones con *sockets*, mientras otros pueden escoger soportar sólo comunicaciones basadas en datagramas. Con el fin de promover la portabilidad a través de dispositivos similares, la especificación MIDP requiere que todos los dispositivos MIDP implementen la interface *HttpConnection*, la cual no es parte del GCF, pero si se deriva de una de sus interfaces, *ContentConnection*.

Gracias al CLDC *Generic Connection Framework*, MIDP soporta la tecnología cliente/servidor y el uso de datagramas. Desafortunadamente MIDP 1.0 sólo implementa el protocolo HTTP 1.1. Aunque el dispositivo pueda usar sockets, MIDP 1.0 no tiene esta

opción, así que habría que desarrollar una clase propia. La versión MIDP 2.0 contiene una implementación de sockets y datagramas.

El uso de sockets y datagramas, hace que las aplicaciones sean menos portables ya que no todas las redes y teléfonos hacen uso de ellos, así que es especialmente recomendable evitar su uso. XML y PHP pueden ser usados para encapsular información y acceder a ella usando HTTP request.

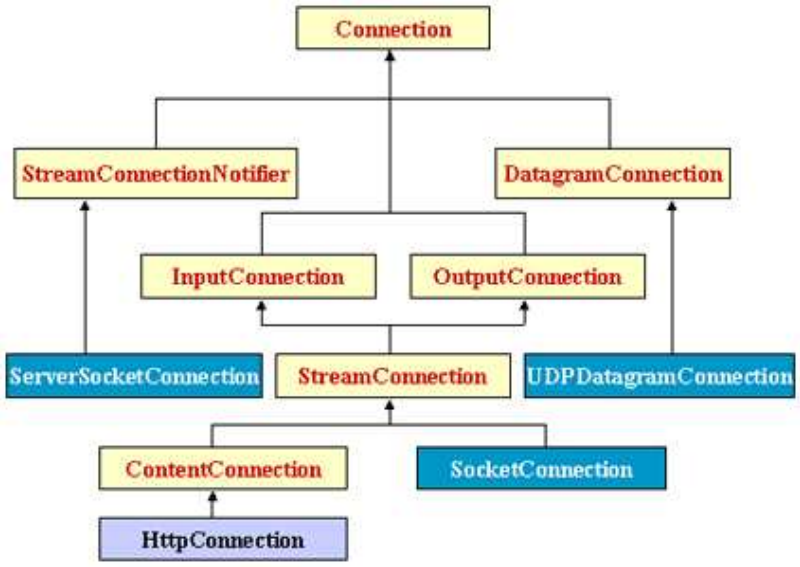

Figura 6. Arquitectura del Generic Connection Framework

Las siete interfaces de conexión definen las abstracciones de los seis tipos básicos de comunicaciones: entrada básica serial, salida básica serial, comunicación con datagramas, comunicación con sockets, mecanismo de notificación en una comunicación cliente-servidor y comunicación HTTP básica con un servidor web.

La clase *Connector* es el núcleo del GCF. La razón de esto es porque todos los objetos tipo *connection* son creados por el método estático *Open* definido en dicha clase. Los diferentes tipos de comunicación se pueden crear por el mismo método cambiando el parámetro tipo *String* que se le pasa. Este diseño hace la implementación J2ME muy extensible y flexible para soportar nuevos dispositivos y productos.

El método se utiliza de la siguiente manera:

Connector.open(String connect);

El parámetro *connect* es una variable de tipo *String*. Debe ser en forma de URL, así: *{protocolo}:[{destino}][{parámetros}]*; consta de tres partes:

- Protocolo. Indica qué tipo de conexión será creada por este método. Existen muchos valores posibles para el protocolo: *file*, *socket*, *comm*, *datagram* y *http*; *file* indica que la conexión se utilizará para entrada y salida en archivo; *comm* indica comunicaciones con el puerto serial y *http* indica comunicación con servidores web.

- Destino. Este puede ser un nombre de *host*, un número de puerto de red, un nombre de archivo o un número de puerto de comunicación.

- Parámetros. Este es opcional y especifica información adicional necesitada para completar la cadena *connect*.

Los siguientes ejemplos ilustran cómo utilizar este método para crear diferentes tipos de comunicación basados en diferentes protocolos:

Comunicación HTTP:

```
Connection hc = Connector.open("http://www.eurolatina.com.co");
```

Sockets TCP:

```
Connection sc = Connector.open("socket://localhost:9000");
```

Comunicación UDP:

```
Connection dc = Connector.open("datagram://localhost:9000");
```

Comunicación por puerto serie:

```
Connection cc = Connector.open("comm.:0;baudrate=9000");
```

Entrada/salida en archivo:

```
Connection fc = Connector.open("file://documento.dat");
```

La interfaz HttpConnection

Teóricamente, se pueden utilizar sockets o datagramas para comunicación remota en una aplicación J2ME, siempre y cuando la configuración MIDP del dispositivo lo soporte. Pero esto crea problemas de portabilidad para estas aplicaciones, ya que no se puede garantizar que otros dispositivos los soporten también. Esto significa que la aplicación puede correr en el dispositivo MIDP pero fallará en otros. Es importante entonces considerar utilizar la clase *HttpConnection*, ya que esta clase se encuentra en todas las implementaciones MIDP.

La interfaz *HttpConnection* se deriva de la interfaz *ContentConnection* en CLDC. Esta hereda de *StreamConnection* todos los métodos para el control de flujos de E/S, de *ContentConnection* todos los métodos para manipular contenidos MIME y añade varios métodos manejar necesidades específicas en la utilización del protocolo HTTP.

Los métodos disponibles en *HttpConnection* son:

- Entrada y salida:
  - *DataInputStream openDataInputStream()*
  - *InputStream openInputStream()*
  - *DataOutputStream openDataOutputStream()*
  - *OutputStream openOutputStream()*

Un *InputStream* puede ser utilizado para leer contenidos de un servidor web mientras que un *OutputStream* puede ser usado para enviar peticiones al mismo.

- Relacionados con MIME:
  - *String getEncoding()*
  - *long getLength()*
  - *String getType()*

Una vez que la conexión *http* se establece, estos tres métodos pueden utilizarse para obtener el valor de los siguientes tres campos en una cabecera http: *Content-Length*, *Content-Encoding* y *Content-Type*.

- Relacionados con el protocolo HTTP:
  - *long getDate()*
  - *long getExpiration()*
  - *String getFile()*
  - *String getHeaderField(int index)*
  - *String getHeaderField(String name)*
  - *long getHeaderFieldDate(String name, long def)*
  - *int getHeaderFieldInt(String name, int def)*
  - *String getHeaderFieldKey(int n)*
  - *String getHost()*
  - *long getLastModified()*
  - *int getPort()*
  - *String getProtocol()*
  - *String getQuery()*
  - *String getRef()*
  - *String getRequestMethod()*
  - *String getRequestProperty(String key)*
  - *int getResponseCode()*
  - *String getResponseMessage()*
  - *String getURL()*
  - *void setRequestMethod(String method)*

- *void setRequestProperty(String key, String value)*

GetResponseCode y *getResponseMessage* pueden ser utilizados para verificar el código de estado de la respuesta y el mensaje del servidor web. Por ejemplo, en este mensaje de respuesta: *"HTTP/1.1 401 Unauthorized"*, se obtendrá el código de respuesta 401 y el mensaje *"Unauthorized"*.

Los siguientes métodos se usan para obtener información adicional de las cabeceras HTTP: *getDate*, *getExpiration*, *getFile*, *getHeaderField*, *getHeaderField*, *getHeaderFieldDate*, *getHeaderFieldInt*, *getHeaderFieldKey* y *getLastModified*.

Los siguientes métodos se usan para obtener los componentes individuales de una cadena URL: g*etHost*, *getPort*, *getProtocol*, *getQuery*, *getRef*, *getRequestMethod*, *getRequestProperty*, *getResponseCode*, *getURL y getResponseMessage*.

Estos tres métodos se usan para el tratamiento de las peticiones HEAD, GET y POST: *setRequestMethod*, *setRequestProperty*, y *getRequestMethod*.

1.4. ARQUITECTURA CLIENTE-SERVIDOR EN J2ME

TCP es un protocolo orientado a conexión. No hay relaciones maestro/esclavo. Las aplicaciones, sin embargo, utilizan un modelo cliente/servidor en las comunicaciones[8].

Un servidor es una aplicación que ofrece un servicio a usuarios de Internet; un cliente es el que pide ese servicio. Una aplicación consta de una parte de servidor y una de cliente, que se pueden ejecutar en el mismo o en diferentes sistemas.

Los usuarios invocan la parte cliente de la aplicación, que construye una solicitud para ese servicio y se la envía al servidor de la aplicación que usa TCP/IP como transporte.

[8] El modelo cliente – servidor. http://neo.lcc.uma.es/evirtual/cdd/tutorial/aplicacion/cliente-servidor.html

El servidor es un programa que recibe una solicitud, realiza el servicio requerido y devuelve los resultados en forma de una respuesta. Generalmente un servidor puede tratar múltiples peticiones (múltiples clientes) al mismo tiempo.

Algunos servidores esperan las solicitudes en puertos bien conocidos de modo que sus clientes saben a que socket IP deben dirigir sus peticiones. El cliente emplea un puerto arbitrario para comunicarse. Los clientes que se quieren comunicar con un servidor que no usa un puerto bien conocido tienen otro mecanismo para saber a qué puerto dirigirse. Este mecanismo podría usar un servicio de registro como *Portmap*, que utiliza un puerto bien conocido.

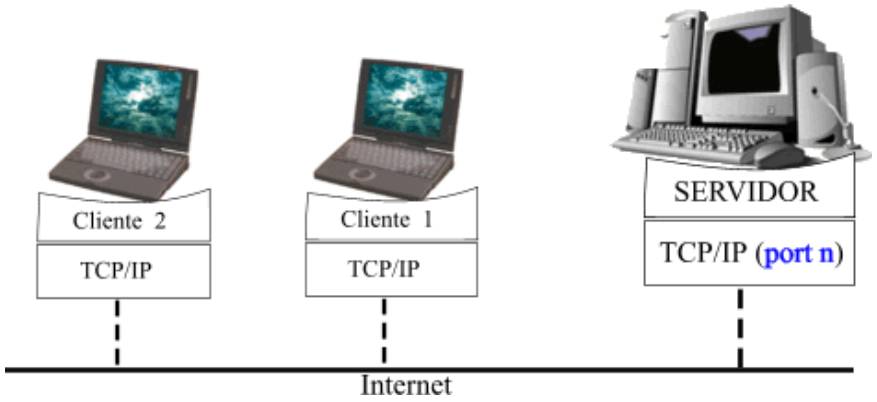

Figura 7. Modelo de aplicación cliente-servidor

Dado que MIDP 1.0 sólo tiene soporte para HTTP, toda la comunicación cliente-servidor debe hacerse a través de un *gateway* HTTP del cual un servidor web es un ejemplo (sin embargo, los fabricantes son libres de agregar soporte para diferentes protocolos de comunicaciones). Esto da una idea de como los MIDlets pueden comunicarse con aplicaciones tipo servidor escritas en cualquier lenguaje; por ejemplo: *perl*, *php*, etc.

1.5. GPRS: EL DESPEGUE DE LA INTERNET MÓVIL

La actual tecnología GSM (*Groupe Special Mobile*) presenta las siguientes características para el envío de datos inalámbricos[9] desde cualquier lugar y en cualquier momento:

[9] GPRS: el despegue de la Internet móvil. http://www.uoc.edu/web/esp/art/uoc/0105021/berbel.html

- Velocidad de transferencia de hasta 9,6 Kbps.
- Tiempo de establecimiento de conexión, de 15 a 30 segundos.
- Pago por tiempo de conexión.

La baja velocidad de transferencia limita la cantidad de servicios que Internet nos ofrece. Por ejemplo, a 9,6 Kbps no se puede navegar por Internet de una manera satisfactoria. Si además tenemos en cuenta que estamos pagando por tiempo de conexión, los costos se disparan. La combinación de estos tres factores negativos hace que GSM sea una tecnología mayoritariamente utilizada para la voz y no para los datos.
GPRS (*Global Packet Radio Service*), es una evolución no traumática de la actual red GSM: no conlleva grandes inversiones y reutiliza parte de las infraestructuras actuales de GSM. Por este motivo, GPRS tendrá, desde sus inicios, la misma cobertura que la actual red GSM. GPRS es una tecnología que subsana las deficiencias de GSM:

- Velocidad de transferencia de hasta 144 Kbps.
- Conexión permanente. Tiempo de establecimiento de conexión inferior al segundo.
- Pago por cantidad de información transmitida, no por tiempo de conexión.

GPRS puede combinar hasta 8 canales para transferir datos, y cada canal puede transferir a una velocidad de 10 Kbps, aproximadamente. El uso de GPRS no se limita sólo a los teléfonos móviles; existen tarjetas PCMCIA GPRS para conectar portátiles a Internet, tarjetas para conectar el ordenador de escritorio, etc.

Cuando se accede a servicios directamente desde un dispositivo móvil, la velocidad de transferencia, a diferencia de lo que puede parecer, no es el factor determinante. Tener una conexión permanente a Internet (lo que se traduce en acceder a servicios de manera casi instantánea) o el hecho de no estar pagando por tiempo de conexión son factores más relevantes que la velocidad de transmisión. Pero, por encima de todo, lo más importante es, sin duda, poder disponer de todos esos servicios.

La evolución natural de GPRS es UMTS (*Universal Mobile Telephony System*). UMTS requiere una nueva tecnología de radio (grandes inversiones en infraestructuras), una red

de mayor capacidad, debido a que las velocidades de transferencia varían de 384 Kbps a 2 Mbps, y nuevos terminales. Estos factores hacer prever que UMTS tardará un cierto tiempo en establecerse y que GPRS, dada su mayor cobertura, mantendrá un uso elevado. Hay que destacar que ninguna tecnología es excluyente entre sí. La aparición de GPRS no excluye GSM; igualmente, UMTS no implica la anulación de GPRS.

GSM, GPRS y UMTS son tecnologías de transmisión de información inalámbrica que se caracterizan por la velocidad que soportan, el tipo de pago y el tiempo de establecimiento de la conexión. WAP (*Gíreles Access Protocol*), sin embargo, es un protocolo; no es comparable a GSM, GPRS y UMTS. Son cosas totalmente distintas. Actualmente, cuando hablamos de WAP, aun siendo un protocolo, en realidad nos estamos refiriendo a la capacidad de mostrar contenidos y servicios directamente en la pantalla de nuestro dispositivo móvil. Por tanto, WAP utiliza la tecnología actual de GSM y utilizará, sin duda, GPRS y UMTS como tecnologías de base.

Debido a las limitaciones de velocidad, teclado, pantalla y capacidad de proceso, es necesario un nuevo lenguaje más ligero y efectivo para mostrar contenidos y servicios en los dispositivos móviles. Este lenguaje se llama *WML* (*Wireless Markup Language*).

Hay dos factores determinantes en el hecho de que WAP no haya tenido el éxito comercial que se esperaba, que pasamos a analizar a continuación:

- Por un lado, es una tecnología joven: si miramos la historia de la tecnología, no encontraremos muchos ejemplos de novedades tecnológicas que se hayan impuesto en menos de 10 años.

- Por el momento, funciona sobre GSM, y el problema no es la velocidad de GSM sino los 30 segundos, aproximadamente, que se tarda en establecer la conexión y el pago por tiempo de conexión.

De WAP e *I-mode* se ha aprendido que los servicios, el pago y el tiempo de establecimiento de conexión son más importantes que la velocidad de transmisión. También hemos visto que las necesidades del usuario móvil no son las del usuario fijo y

que, por tanto, hay que readaptar los contenidos, porque siempre tendremos pantallas pequeñas.

1.6. DESARROLLO DE APLICACIONES PARA DISPOSITIVOS MÓVILES

El desarrollo de aplicaciones destinadas a dispositivos móviles, desde el punto de vista de la Ingeniería del Software, no debe diferir sustancialmente de los pasos a dar cuando se construyen aplicaciones para ordenadores de escritorio o estaciones de trabajo[10]. Así, podríamos establecer los siguientes pasos previos:

1.6.1. Análisis de requerimientos

El analista deberá determinar normalmente con varias entrevistas con los usuarios, las necesidades que estos tienen y los requerimientos que se le pedirán a la aplicación. Por ejemplo, en el caso de un análisis para una aplicación que se ejecutará en un dispositivo móvil, algunos de estos requerimientos generales pueden ser la facilidad de uso, que se pueda ejecutar en teléfonos móviles, PDAs y paginadores, que permita una conexión a una entidad mayor para obtener datos actualizados o devolver otros, o también, que sea capaz de almacenar cierta información de manera persistente.

1.6.2. Diseño de la aplicación

Es muy importante en este tipo de aplicaciones el crear programas separados por cada uno de los posibles usos que se le dé a la aplicación. De esta manera cada programa será más pequeño y se adaptará mucho mejor a las características de los dispositivos móviles. Por tanto, a la hora del diseño nos plantearemos esta tarea seriamente, pues finalmente serán varias las ventajas de hacerlo así. Ya en la fase de implementación se tendrá que establecer un mecanismo que controle las diferentes aplicaciones.

---

[10] Programación de dispositvos móviles con Java. http://leo.ugr.es/~fjgc/INTRO/intro_9.htm

En cuanto al diseño de la interfaz de usuario, debemos decidir la correspondencia entre la aplicación y la pantalla. Los diseñadores en esta fase no deben considerar cómo los usuarios operarán con el dispositivo para llevar a cabo una tarea, o cómo se notificará a la aplicación las acciones del usuario. Se deben concentrar sólo en el objetivo de la pantalla y en la tarea que permitirá llevar a cabo. Sun recomienda en esta etapa que se haga un *story board* conteniendo en cada viñeta los requerimientos para la pantalla correspondiente. En otra fase se decidirá qué tipo de controles vamos a utilizar para realizar entradas de datos y cómo vamos a presentar la información. En este punto, las características generales en cuanto a pantalla del dispositivo pueden marcar claramente el tipo de diseño de interfaz: lo que en uno se puede disponer en una única pantalla, en otro podremos necesitar varias.

El almacenamiento persistente es un aspecto a tener en cuenta en nuestro diseño. La pregunta a responder es: ¿qué datos deben sobrevivir a la finalización de la aplicación y estar disponibles para la siguiente vez que se vaya a ejecutar? Otra cuestión, que no se debe plantear en esta fase sino en la de implementación es qué utilizar para realizar ese almacenamiento. Una primera respuesta es aquel formato que se emplee para enviar y recibir datos entre el dispositivo J2ME y el sistema externo. Con esto evitamos una fase de conversión de formatos. Si el dispositivo posee sistemas de archivos, entonces podemos optar por la creación de un archivo con una estructura más o menos compleja y usar las bibliotecas de Java para acceder a ellos. Otra alternativa también puede ser emplear sistemas de gestión de bases de datos relacionales, aunque en este caso es difícil evitar realizar gran cantidad de accesos al almacenar un gran volumen de datos.

Finalmente, debemos tener en cuenta dentro del diseño, aspectos relacionados con la conectividad y con la entrada/salida, ya que son puntos muy importantes que van a determinar la portabilidad de la aplicación. Por tanto, en este momento deberemos tomar decisiones en un nivel de abstracción alto, que luego se concretarán cuando determinemos claramente el tipo de dispositivo y sus prestaciones.

1.6.3. Implementación de la aplicación

Esta etapa vendrá marcada por la elección del lenguaje, plataforma y herramientas de desarrollo (depuradores, entornos integrados, etc.). Con J2ME, teniendo en cuenta tipo de dispositivo con el que vamos a trabajar, decidiremos la configuración y perfil más adecuados.

Los pasos a seguir en esta fase hasta instalar el programa en el dispositivo serían los siguientes:

1. Escritura del código.

2. Compilación de la aplicación.

3. Eliminación de información de clases innecesarias (*obfuscate*). Esta etapa es opcional y en ella se renombran clases, métodos, interfaces, con objeto de hacerlo ambiguo. Un paquete obtenido de esta fase lo protege de la descompilación y de la ingeniería inversa. Además, reduce el tamaño de los archivos de clase, dando lugar a archivos JAR más pequeños.

4. Ejecución del preverificador para añadir la información de *clase verificada* a los archivos de clase.

5. Empaquetamiento de la aplicación: creación de los archivos JAR y JAD.

6. Ejecución en un emulador apropiado.

7. Instalación en el dispositivo y ejecución. En este caso existen dos modos de hacerlo: en el primero, se descargará la aplicación a través de una conexión de red, se cargará en memoria, se ejecutará la aplicación, y finalmente se eliminará cualquier traza de ésta en el dispositivo; en la segunda, y siempre que el dispositivo lo permita, se instalará físicamente. En el entorno J2ME, *Java Application Manager* (JAM) es un

gestor que controla la descargar, instalación, lanzamiento y desinstalación de aplicaciones en el dispositivo.

Algunos aspectos a tener en cuenta en el momento de desarrollar una aplicación J2ME son los siguientes:

1. Evitar ejecutar tareas computacionalmente intensivas en el dispositivo. Cuando sea posible, la alternativa más sencilla es hacer que una posible aplicación servidora en una máquina servidora sea la que haga el cálculo y que la aplicación que corre en el dispositivo sólo se encargue de la gestión de la interfaz y de mínimas operaciones. Simplificar la aplicación, es decir, dejarla tan simple como sea posible, eliminando características superfluas. Esta decisión básicamente debe realizarse en tiempo de diseño. Si se va a necesitar un requerimiento sólo de vez en cuando, entonces lo mejor es desarrollar una segunda aplicación auxiliar que la contenga, dando la oportunidad a los usuarios a que puedan eliminar dicha aplicación si no la necesitan.

2. Escribir aplicaciones más pequeñas (s*maller is better*) ya que consumirá menos memoria, requerirá menos tiempo de instalación y de inicialización al ser ejecutada. Un aspecto para alcanzar este fin es el de reducir el número de clases de la aplicación.

3. Utilizar menos memoria en tiempo de ejecución. Para ello podemos considerar el uso de tipos escalares en sustitución de objetos más complejos siempre que sea posible (como son *int* y *boolean*, por ejemplo), ya que cada vez que construimos un tipo, el constructor entra en funcionamiento reservando el espacio de memoria requerido para alojarlo. También otro buen hábito en este sentido es alojar objetos cuando realmente sea necesario (*lazy instantiation*). Así, cuando vayamos a utilizar un objeto podemos preguntar previamente si ha sido creado o simplemente apunta a *null*.

4. Liberar los recursos del programa tan pronto como se acaben de utilizar. Una vez que se finalice el uso de conexiones a bases de datos, a la red, a ficheros, etc., liberarlos, ya que dejaremos libre la memoria que implica su gestión y se pondrán a disposición de otras aplicaciones.

5. Reutilizar objetos en lugar de continuamente crearlos y abandonarlos. La idea, por tanto, es escribir métodos de inicialización de objetos y también de dejarlos en un estado neutro para posteriormente poder emplearlos en otros menesteres.

6. Evitar excepciones cuando se pueda, ya que redundará en la reducción del tamaño de los archivos de clase y el número de objetos que se alojen en memoria. Siempre que se pueda solventar el problema de otra forma es mejor.

7. Utilizar variables locales, ya que es más lento acceder a un miembro de una clase que a una variable local. Por ejemplo, si se accede a una variable miembro dentro de un bucle una y otra vez, será conveniente asignar el miembro a una variable local, que se almacenará en la pila, y acceder continuamente a ella. También es útil esta técnica para operar con elementos de un vector en vez de acceder directamente por medio del vector.

8. Evitar concatenación de cadenas de caracteres debido a que se emplea un par de llamadas a métodos de la clase *String* más la del constructor correspondiente. Esto se tiene que tener más en cuenta, aún si cabe, si se hace en un bucle.

9. Emplear hilos. Por norma general, todas aquellas operaciones que lleven más de una décima de segundo deberían ejecutarse en un hilo separado, de tal forma que no se bloquee la interfaz de usuario, aspecto muy importante en los dispositivos móviles.

2. PROTOTIPO

2.1. DESCRIPCIÓN DEL PROBLEMA

Los administradores de servidores – *web, correo, etc.* – diariamente se enfrentan al riesgo de caídas o bloqueos de servicios, fallas en procesos, errores inesperados en tareas específicas, etc., cuya solución muchas veces requiere de unos pocos pasos, o incluso de sólo reiniciar el elemento que falló. En ocasiones, se documentan los errores frecuentes y se procura tener a la mano herramientas o procedimientos de solución eficaz de estas situaciones.

Si, por ejemplo, el servicio de entrega de correos se bloquea, muchas veces la solución más rápida es simplemente reiniciarlo; entonces, el administrador posiblemente irá a una consola en su servidor y ejecutará una pequeña serie de comandos para reestablecer el servicio (tal vez ya tiene un archivo de ejecución por lotes que realiza la tarea). Pero ¿qué ocurrirá si el administrador no está en su lugar de trabajo? ¿si no se encuentra cerca de un computador conectado a Internet? Él sabe que bastará con ejecutar *aquel* archivo por lotes que tiene alojado en el servidor, pero no podrá hacerlo desde su ubicación actual.

2.2. ANÁLISIS DE REQUERIMIENTOS

2.2.1. Requerimientos de Hardware

Para implementar el cliente en un dispositivo móvil, los requerimientos mínimos de hardware son:

- Versión de java: CLDC 1.0 + MIDP 1.0

- Memoria: 128 KB de memoria no volátil para componentes MIDP, 8KB de memoria no volátil para datos persistentes creados por la aplicación, 32KB de memoria volátil para el *Runtime* de Java, y 5 KB de memoria disponible para almacenamiento de aplicación.
- Tamaño pantalla: 96 x 54 Pixeles.
- Profundidad de Display: 1 bit.
- Relación de aspecto de pixeles (*Píxel shape)*: 1:1
- Red: Dos vías, inalámbrico, con ancho de banda limitado.
- Entrada: Una o más de las siguientes: teclado one-handed, teclado two-handed, touch screen.

2.2.2. Requerimientos funcionales

Se trata de una aplicación típica cliente / servidor, donde el cliente se alojará en un dispositivo móvil que enviará órdenes a la aplicación servidor, por medio de una comunicación HTTP, y el servidor se encargará de ejecutar unos comandos previamente configurados a través de un *script* PHP, y de enviar una confirmación al cliente, ya sea de suceso o de error. Por lo tanto, para el prototipo se escogieron cuatro comandos sencillos que son:

1. *Tamaño de Disco*. Muestra el tamaño del disco duro del servidor.
2. *Nombre Servidor*. Muestra el nombre del equipo servidor.
3. *Sistema Operativo*. Muestra el nombre del Sistema operativo del servidor.
4. *Procesador*. Muestra la arquitectura del procesador del servidor.

Es posible configurar comandos más complejos y se podrán ejecutar de igual manera, sin riesgo de aumentar el consumo de transferencia en el dispositivo móvil, ya que siempre van a correr en el lado del servidor.

2.2.3. Casos de uso

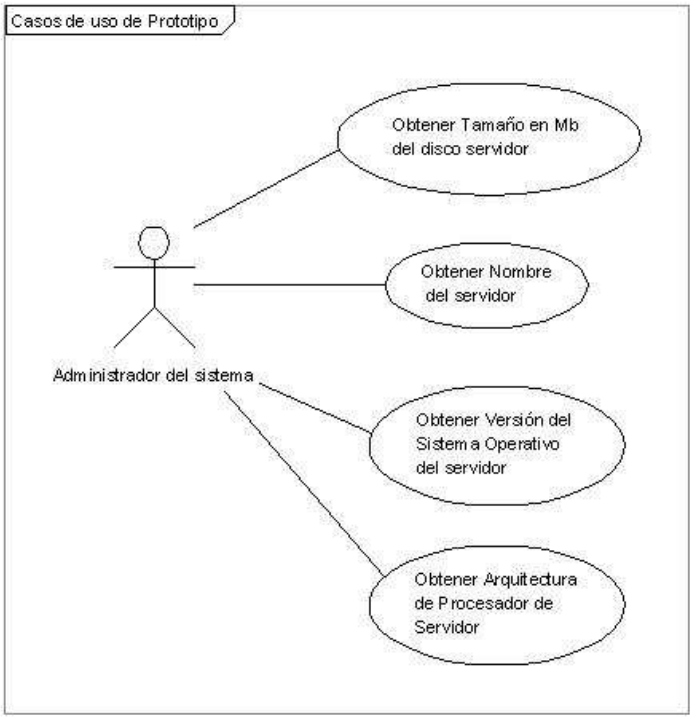

Figura 8. Diagrama de casos de uso

- Requerimiento que satisface: Las cuatro opciones del menú

- Precondición: Que el dispositivo tenga conexión a Internet y que se haya seleccionado una opción del menú.

- Actor: Administrador del sistema.

- Flujo normal de eventos:
  (1) Se construye en el móvil la URL a la que se va a comunicar
  (2) Se establece la conexión con el servidor
  (3) Se envía la instrucción al servidor como un HTTP *Request*
  (4) El servidor ejecuta la tarea correspondiente
  (5) El servidor envía respuesta al cliente mediante un *HTTP Response*
  (6) Se muestra la respuesta del servidor en el dispositivo móvil

- Flujo alternativo:
  (2a) No es posible establecer una conexión con el servidor. Se muestra un mensaje de error y se muestra la opción de regresar a la pantalla de comandos.

El funcionamiento de los cuatro casos de uso es el mismo. La diferencia consiste únicamente en la respuesta que se muestra en el cliente dependiendo de la opción seleccionada; así:

- Para la opción *"Tamaño de Disco"*, la respuesta obtenida es: 'Tamaño: <tamaño_disco> MB'
- Para la opción *"Nombre Servidor"*, la respuesta obtenida es: 'Nombre: <nombre_servidor> '
- Para la opción *"Sistema Operativo"*, la respuesta obtenida es: 'OS: <sistema_operativo>'
- Para la opción *"Procesador"*, la respuesta es: 'Proc: <arquitectura_procesador>'

2.2.4. Diagrama de Clases

El prototipo está constituido principalmente por dos clases: **Prototipo** y **Peticion**.

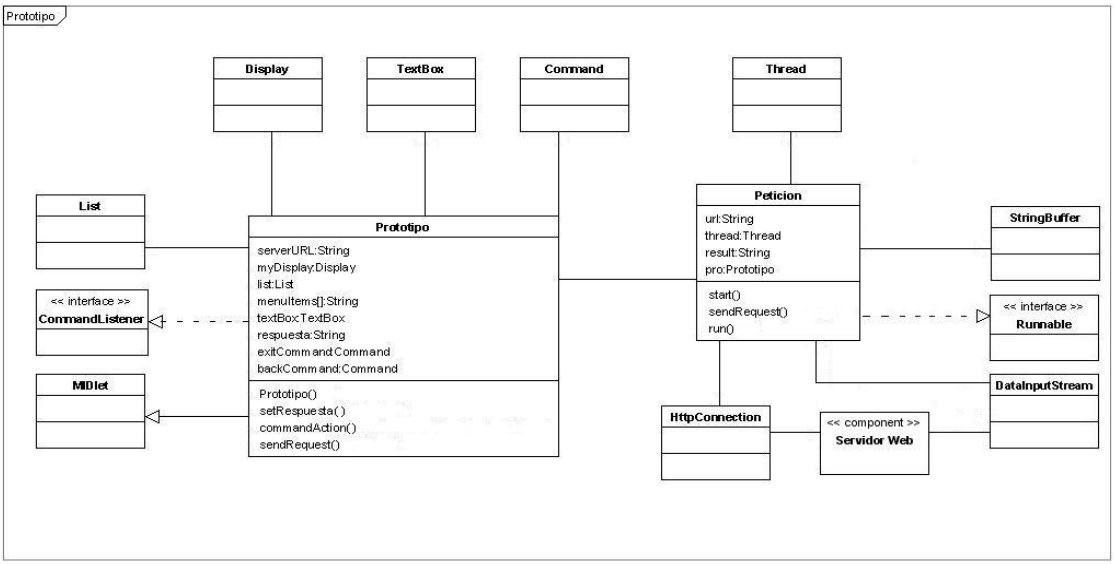

Figura 9. Diagrama de clases

La clase **Prototipo** extiende a la clase **MIDlet**, la cual posee tres métodos abstractos que deben implementarse: *startApp(), pauseApp()* y *destroyApp()*; a su vez, implementa la Interfaz CommandListener, que contiene un método abstracto llamado *commandAction(Command c, Displayable d)*, en donde se indicará la acción que realizará cuando se produzca un evento en el *Command c* que se encuentra en el objeto *Displayable d.*

Los atributos que componen a la clase **Prototipo** son:

- *serverURL*, una constante *String* que guarda la URL a enviar al servidor sin la petición GET construida.
- *myDisplay*, un objeto de tipo *Display* el cual guarda la referencia a la pantalla del dispositivo.
- *List*, un objeto de tipo *List*, que representa la lista de comandos a seleccionar.
- *menuItem[]*, un arreglo *String* que contiene el listado de las opciones a mostrar en *list*.
- *textBox*, un objeto de tipo *TextBox* que representa el área de texto donde se mostrará la respuesta recibida del servidor.
- Respuesta, un *String*, que guardará la cadena recibida por el servidor.
- *exitCommand*, un objeto de tipo *Command* que mantiene información del evento *SALIR* de la aplicación.
- *backCommand*, un objeto de tipo *Command* que mantiene información del evento *VOLVER* de la aplicación

A su vez, los métodos que componen esta clase son:

- *Prototipo():void*. Es el constructor de la clase y se usa para crear la referencia a la pantalla del dispositivo, inicializar los objetos gráficos, crear y delegar los comandos *SALIR* y *VOLVER* y establecer el manejador de eventos (*setCommandListener()* de la lista).
- *setRespuesta(r:String):void*. Este método permite crear un objeto de tipo *TextBox* para visualizar la respuesta enviada por el servidor. Le asigna la pantalla, y el manejador para el comando *VOLVER*.

- *commandAction(com:Command,disp:Displayable):void*. Como se dijo anteriormente, es un método abstracto de la interfaz *CommandListener* que se implementa en esta clase y permite manejar las acciones a realizar cuando se seleccione los comandos *VOLVER*, *SALIR* y las opciones de la lista *list*.
- *sendRequest(pet:int):void*. Este método termina de construir la URL que se enviará al servidor, añadiéndole la petición como un *query string*. Por cada tipo de petición crea un hilo de ejecución *Peticion* y lo ejecuta.

Por su parte, la clase **Peticion** es la encargada de establecer la conexión con el servidor, enviar la petición HTTP, recibir la respuesta y notificar al objeto **Prototipo** de esta respuesta. Implementa la interfaz *Runnable* para permitir crear, por cada comando escogido por el usuario, un hilo de ejecución para procesar dicho comando y devolver la respuesta.

Los atributos de esta clase son:

- *url*, un *String* que guarda la URL ya construida (con el *query string*) que se enviará al servidor.
- *thread*, un objeto de tipo *Thread* que representa un hilo de ejecución.
- *result*, un *String* que guarda la respuesta del servidor una vez se haya procesado el comando.
- pro, un objeto Prototipo que guarda la referencia al objeto *Prototipo* quien llama a la clase ***Peticion***.

Además, está compuesta por los siguientes métodos:

- *Petición(u:String, Prototipo p):void*. Método constructor cuya función es inicializar los atributos de la clase. Recibe como parámetros la URL ya construida para enviar al servidor y el objeto *Prototipo*.
- *start():void*. Crea y comienza el hilo de ejecución. Este es un método abstracto de la interfaz *Runnable* que se implementa en esta clase.
- *sendRequest(u:String):void*. Este método establece una conexión con el servidor mediante la creación de un objeto *HttpConection*. Por medio de este objeto se

envía la URL mediante un *HTTP request (GET)*, se crea un flujo de entrada de tipo *DataInputStream* para manejar la respuesta del servidor y esta respuesta se va almacenando en un objeto de tipo *StringBuffer*. Al final, se establece el atributo *result* con la respuesta recibida.

- *run():void*. Otro método abstracto de la interfaz *Runnable* que se implementa en esta clase y es responsable de las acciones del hilo de ejecución. Este método hace una llamada a *sendRequest*(u:String) y luego al método *setRespuesta(r:String)* del objeto **Prototipo**, en donde se notifica la respuesta recibida.

2.3. MODELO DE DISEÑO

2.3.1. Diagrama de despliegue

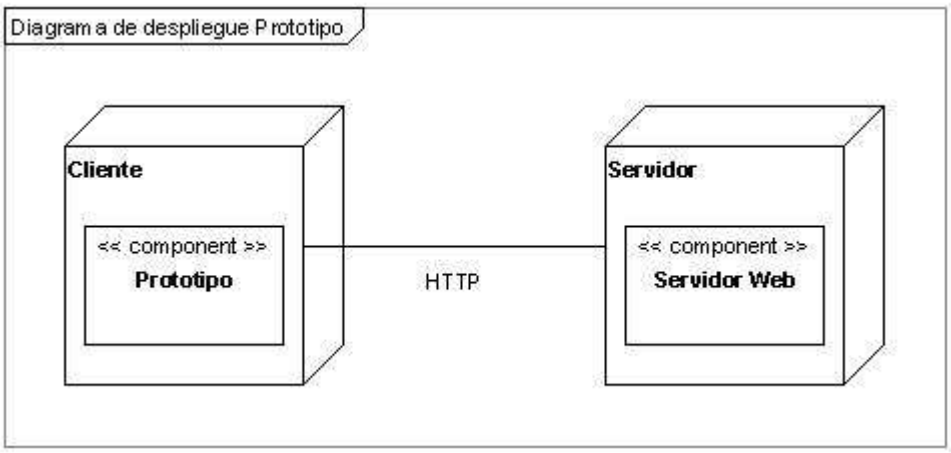

Figura 10. Diagrama de despliegue

En este diagrama se describe la relación entre el cliente (dispositivo móvil) y el servidor por medio de una comunicación HTTP. En el lado del cliente, el componente representa la aplicación prototipo, compuesta por sus dos clases principales ***Prototipo*** y ***Peticion***; y en el lado del servidor, el ***Servidor web*** en el que se ejecutarán las peticiones mediante un script PHP.

2.3.2. Diagrama de clases detallado

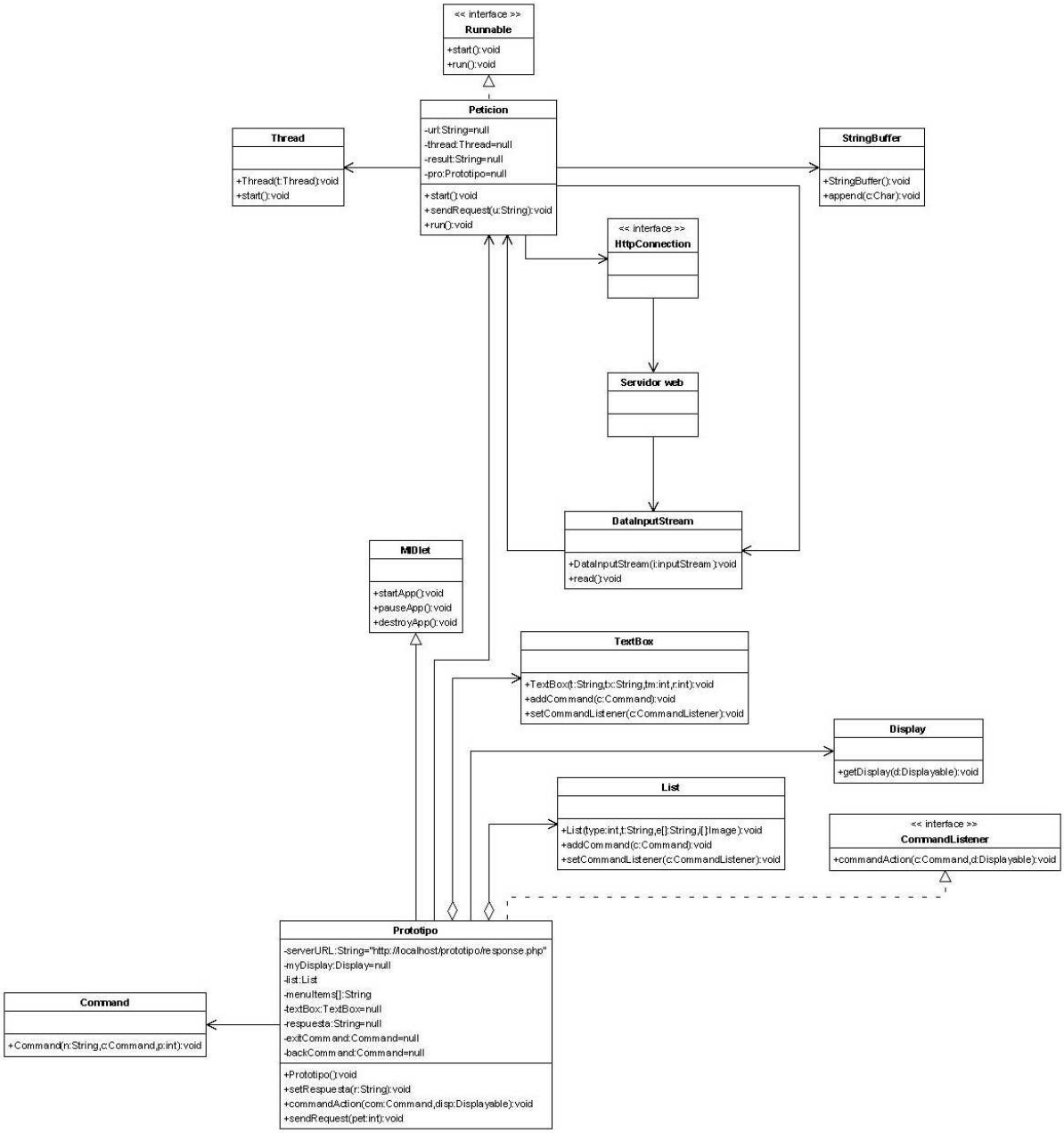

Figura 11. Diagrama de clases detallado

2.3.3. Diagrama de secuencia

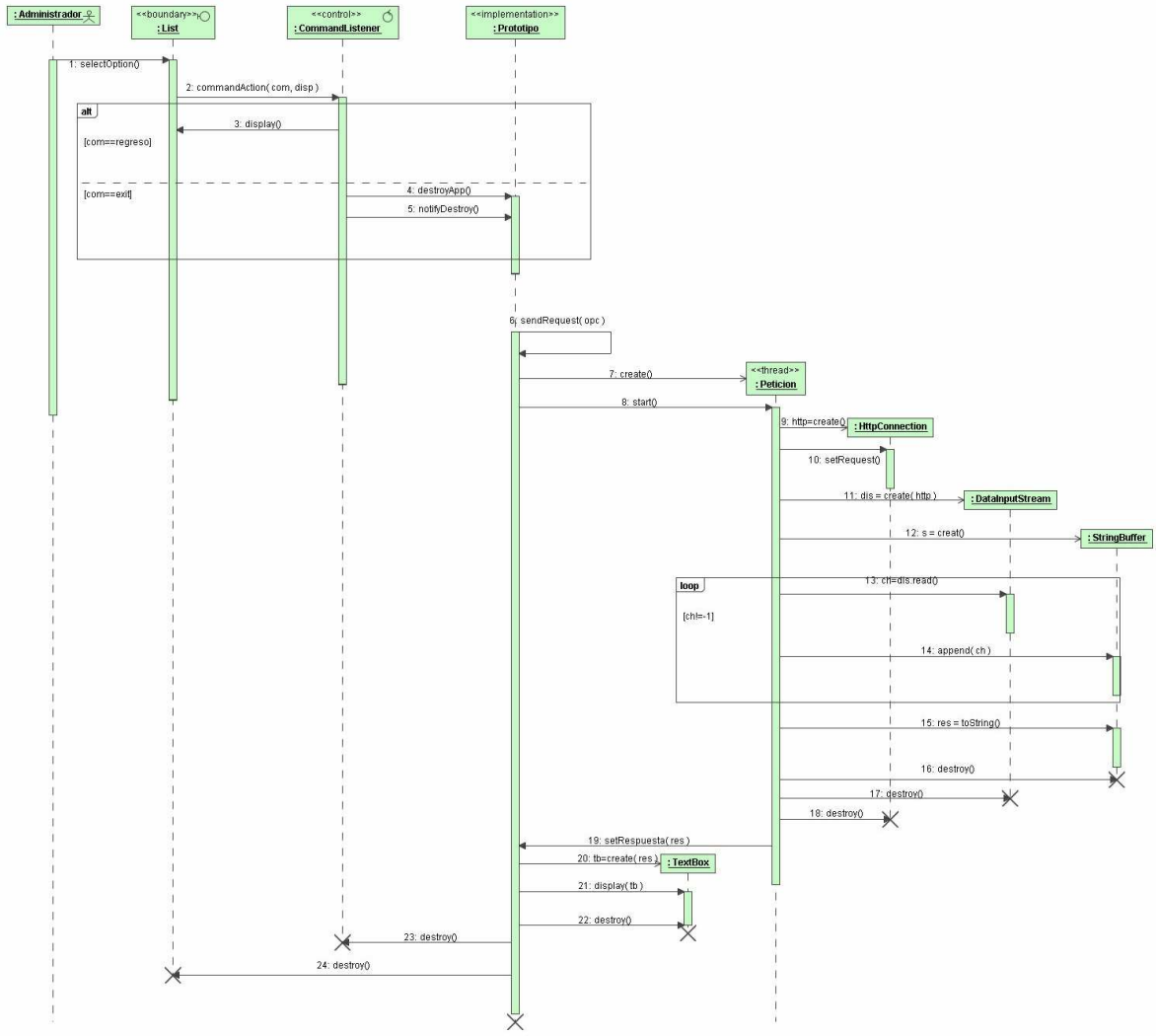

Figura 12. Diagrama de secuencia

CONCLUSIÓN

Es innegable que las tendencias de tecnología informática que se analizan en el cercano futuro son la movilidad, la conectividad, la convergencia y la miniaturización. Todas estas cuatro se representan en dispositivos móviles.

Dado al auge que estos están teniendo en todo el mundo y particularmente en nuestro país, se requiere de una nueva generación de profesionales especializados en el desarrollo de aplicaciones para esta plataforma y de usuarios capaces de entender cuáles son las potencialidades de estas nuevas tecnologías. No hace falta más que echar una mirada en las calles o en reuniones sociales para ver que prácticamente todas las personas, de cualquier clase social, con los más diversos gustos y de todos los sexos tienen al menos un dispositivo móvil: un celular.

En esta nueva realidad, de la que afortunadamente hoy somos parte, se presenta una gran gama de posibilidades para desarrollo de aplicaciones que solucionen problemas o satisfagan necesidades, de maneras antes impensadas.

Ya no es indispensable estar en frente de una estación de trabajo o un servidor para ejecutar tareas en ellos. Basta con que estén conectados a Internet y tener un dispositivo capaz de accederlos de manera remota.

El prototipo elaborado en la monografía es una muestra fiel de cómo pueden satisfacerse todos los requerimientos anteriormente mencionados, utilizando las tendencias de tecnología informática de las que ya hemos hablado, y nos muestra al final la aplicación de las mismas en nuestro trabajo y diario vivir.

Para la elaboración del prototipo, se utilizó el entorno de desarrollo J2ME Wireless toolkit 2.2 de la Sun Microsystems. Este entorno permitió ejecutar la preverificación de todas las clases de la parte cliente desarrolladas, depurar el MIDlet y crear los archivos JAR y el

descriptor JAD. En este mismo entorno, se realizó la emulación de un dispositivo Móvil y se probó la funcionalidad del prototipo.

Para la parte del servidor, se utilizó el *Internet Information Server* como servidor web y un script en PHP para la ejecución de las peticiones del cliente.

RECOMENDACIONES

Una vez concluida la monografía, se considera interesante investigar sobre otros aspectos relacionados con el desarrollo de aplicaciones para dispositivos móviles y se propone:

- Estudiar la forma en que J2ME maneja las sesiones y las *cookies*.

- Emplear técnicas criptográficas que garanticen la confidencialidad, la seguridad y la autenticación en el prototipo.

- Implementar un sistema de *login* para el prototipo.

- Investigar otros sistemas operativos para dispositivos móviles existentes en el mercado.

- Investigar sobre el entorno de desarrollo *.NET Compact Framework*, competidor de *J2ME*.

BIBLIOGRAFÍA

BERBEL NAVARRO, Genís. GPRS: el despegue de la Internet móvil. <http://www.uoc.edu/web/esp/art/uoc/0105021/berbel.html>

GÁLVEZ ROJAS, Sergio y ORTEGA DÍAZ, Lucas. Java a tope: J2ME (Java 2 Micro Edition). Universidad de Málaga. <http://www.lcc.uma.es/~galvez/J2ME.htm>

GIGUERE, Eric. Java 2 Micro Edition: Professional Developer's Guide. Capítulo 3. <http://www.j2medeveloper.com/j2mebook/Chapter3.pdf>

J2ME Wireless Toolkit 2.0 User's Guide – SUN Microsystems <http://java.sun.com/j2me/docs/wtk2.0/UserGuide.pdf>

KNUDSEN, Jonathan. Wireless Java: Developing with J2ME, Second Edition. <http://www.apress.com/ApressCorporate/supplement/1/138/1590590775-898.pdf>

NÁJERA, Gilberto. Desarrollo de aplicaciones para dispositivos inalámbricos (J2ME). <http://www.mailxmail.com/curso/informatica/dispositivosinalambricos>

FERNÁNDEZ LUNA, Juan Manuel. Programación de dispositivos móviles con Java. <http://leo.ugr.es/~fjgc/INTRO/index2.htm>

Todo Symbian – Iniciación a J2ME. <http://www.todosymbian.com/secart36.html>

Herramientas web para la enseñanza de protocolos de comunicación. El modelo cliente – servidor. <http://neo.lcc.uma.es/evirtual/cdd/tutorial/aplicacion/cliente-servidor.html>