Desarrollo de un servidor HTTP para dispositivos móviles en J2ME

Guillermo Diez-Andino Sancho, Rosa Mª García Rioja y Celeste Campo Vázquez
Departamento. Ingeniería Telemática - Universidad Carlos III de Madrid
Avda. Universidad 30 Leganés (Madrid), España
{gdandino, rgrioja, celeste}@it.uc3m.es

Abstract

El protocolo HTTP viene siendo ampliamente utilizado desde el nacimiento de la web, en donde es indispensable para la creación de aplicaciones que hacen uso de ésta. Con la llegada de las tecnologías móviles pueden darse diversas situaciones en las que la creación de un servidor HTTP en dispositivos limitados pueda resultar de gran utilidad. Un ejemplo, es la compartición de la información almacenada en las agendas electrónicas de los asistentes a una reunión para poder concertar otra reunión en las próximas semanas.

La plataforma J2ME dispone del soporte necesario para la creación de todo tipo de clientes HTTP. Estos clientes HTTP están pensados en un principio para la utilización y comunicación con aplicaciones residentes en PCs, de modo que los dispositivos móviles puedan aprovechar todas los servicios ofrecidos por estas máquinas y tener acceso a todo tipo de información almacenada en ellas (páginas Web, aplicaciones para su descarga, etc).

En este artículo se describe de forma detallada tanto el diseño como el desarrollo de un servidor HTTP en dispositivos limitados basado en la tecnología J2ME. Para que de esta forma, sean los propios dispositivos móviles los que proporcionen estos servicios, permitiendo acceder a ellos, tanto a otros dispositivos móviles como a los que no lo son, sin necesidad de un servidor intermedio.

I. Introducción

El principal objetivo a conseguir es construir un servidor HTTP sobre el perfil J2ME MIDP, de manera que diferentes aplicaciones residentes en distintos dispositivos puedan comunicarse directamente a través del protocolo HTTP. El servidor residente en el dispositivo móvil será el encargado de proporcionar los datos solicitados por las diferentes aplicaciones, de modo que no sea necesario un elemento intermedio (típicamente un PC).

En primer lugar se van a tratar los principales aspectos de la comunicación mediante sockets en MIDP, se planteará la problemática existen a la hora de utilizar los *sockets pasivos* así como las soluciones planteadas.

Una vez superada la problemática de utilización de sockets se describe en detalle la construcción de un servidor HTTP. El servidor desarrollado soporta las características básicas del protocolo HTTP 1.0 (método GET, HEAD y OPTIONS), permitiendo varias conexiones simultáneas e informando con los códigos de error producidos en las diferentes situaciones que se puedan producir.

A continuación se describen en profundidad cada uno de los principales módulos, siendo el primero de ellos el de **configuración**, mediante el cual se definen las opciones básicas de funcionamiento (temporizadores, número de conexiones simultáneas, puertos, tipos MIME soportados, etc).

A través del **sistema de archivos** (segundo módulo) se van a poder gestionar los recursos a albergar por el servidor así como resolver las peticiones de los usuarios. Ha sido necesario desarrollar un sistema de archivos en MIDP ya que RMS[1] no proporciona ni mecanismos de acceso a ficheros, ni granularidad de permisos a la hora de acceder a los recursos existentes (archivos de sonido, imágenes,...). Por último, es mediante el **servicio de tratamiento de peticiones** (tercer módulo) que las diversas solicitudes HTTP podrán ser tratadas.

Una vez tratada la arquitectura del sistema en detalle se procede a describir la funcionalidad implementada en el servidor, las cabeceras de petición/respuesta existentes, los métodos y códigos de error

[1]Record Management System

soportados así como las principales opciones de configuración existentes para finalmente exponer las principales conclusiones obtenidas.

II. Problemática de los sockets

Un paso previo a la construcción de un servidor HTTP sobre MIDP consiste en la activación de los `serversockets` a nivel MIDP. Esto se debe a que el entorno de desarrollo empleado, J2ME Wireless Toolkit de Sun, únicamente soporta comunicación en modo cliente HTTP, lo cual supone una barrera para la creación de un servidor HTTP.

El principal problema reside en el hecho de que por defecto no se pueden utilizar los sockets en la implementación MIDP 1.0. Aunque sí se encuentran implementados a nivel CLDC, no se tiene acceso a ellos desde el perfil MIDP, únicamente se emplean a través de determinadas clases que hacen uso de ellos, pero nunca directamente.

Debido a esto, es necesario encontrar una solución que pasa por crear un nuevo perfil que sí tenga acceso a ellos o bien activar alguna característica de configuración que nos proporcione el acceso deseado.

A. Soluciones planteadas

Entre las diversas soluciones planteadas a la hora de utilizar los sockets bajo la implementación de Sun se pueden destacar las siguientes:

- Existe una característica no documentada que permite ejecutar un programa que utilice UDP o un socket, estableciendo para ello la variable de entorno `com.sun.midp.io.enable_extra_protocols` a true. Esta variable se encuentra disponible en el fichero `internal.config` del directorio bin del Wireless Toolkit.

Activando esta variable en el Wireless Toolkit debería existir acceso a los sockets, tanto activos como pasivos. No obstante, esta solución es algo relativa, ya que en los entornos de ejecución en donde no se puedan realizar este tipo de modificaciones (por ejemplo la máquina virtual para Palm OS) la utilización de sockets seguiría sin estar disponible.

Esta solución se puede utilizar de manera temporal y a efectos de desarrollo, ya que sería necesario buscar una nueva implementación del perfil MIDP que sí soporte los sockets.

- Una segunda solución a este problema consiste en ejecutar el MIDlet con el parámetro `-D` indicando que se desea tener accesibles los protocolos adicionales (en caso de que existan) a nivel de perfil. Al igual que con la solución anterior, va a existir la misma problemática en aquéllos entornos de ejecución en donde no sea posible pasar parámetros al ejecutar los MIDlets.
- Otra posibilidad consiste en engañar a la clase `Connection` de manera que crea que socket sea un protocolo válido. Esto se puede hacer creando una clase `Protocol.class` y situándola en `com/sun/midp/io/j2me/socket`, que es donde la clase `Connector` buscará los protocolos soportados.

Esta clase `Protocol` debería heredar de `com.sun.cldc.io.j2me.socket.Protocol`. Relacionada con esta solución existe la variable `javax.microedition.io.Connector.protocolpath` del fichero `system.config` con la que se puede especificar la ruta donde se encuentran las clases con los protocolos disponibles en el sistema.

- Por último existe la posibilidad de recompilar el perfil con las opciones de soporte de sockets activadas de manera que el nuevo perfil soporte la utilización de sockets. Este cambio habría que realizarlo para todas y cada una de las diferentes máquinas virtuales existentes (Palm OS, Pocket PC, etc).

Cada uno de estas soluciones planteadas lleva asociada una complejidad diferente. De este modo se ha considerado la solución de activación de los sockets mediante la variable `com.sun.midp.io.enable_extra_protocols` la más sencilla y viable durante la fase de desarrollo.

Es de destacar que en la versión 2.0 del MIDP los sockets se encuentran disponibles por defecto, no siendo necesario llevar a cabo ningún tipo de modificación en el perfil o de configuración mediante parámetros.

III. Arquitectura

El servidor está compuesto por tres módulos principalmente. El primero de ellos es la configuración mediante la cual se definen las opciones básicas de funcionamiento. A través del sistema de archivos (segundo módulo) se van a poder gestionar los recursos a albergar por el servidor así como resolver las peticiones de los usuarios. Por último es mediante el servicio de tratamiento de peticiones que las diversas solicitudes HTTP podrán ser tratadas.

La arquitectura se puede ver de manera esquemática en la figura 1.

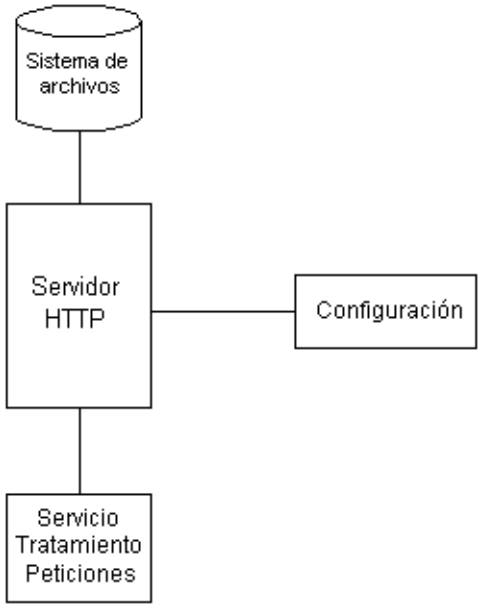

Fig. 1. Arquitectura del Servidor HTTP sobre J2ME

En los siguientes apartados se describe en detalle cada uno de los módulos que componen el servidor desarrollado.

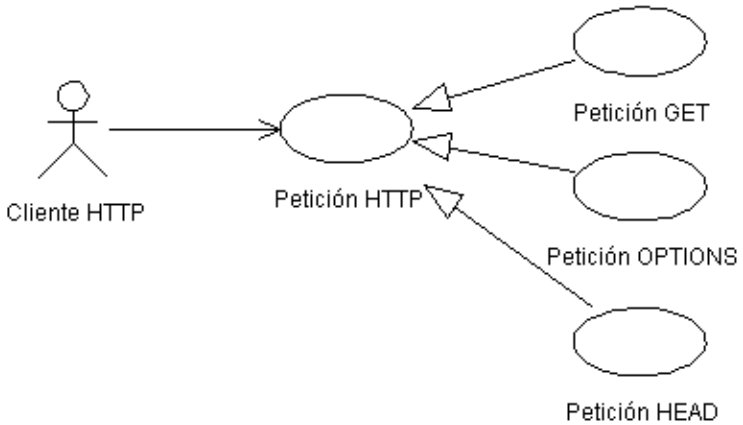

Fig. 2. Diagrama de casos de uso del servidor HTTP.

Las clases y los interfaces existentes se muestran en el diagrama de clases de la figura 3.

El servidor ha sido diseñado de modo que se minimice la complejidad a la hora de añadir futuras ampliaciones y mejoras. La utilización de patrones de diseño ha sido un factor determinante a la hora de lograrlo, ya que simplifican en extremo los cambios a realizar en el código a la hora de modificarlo.

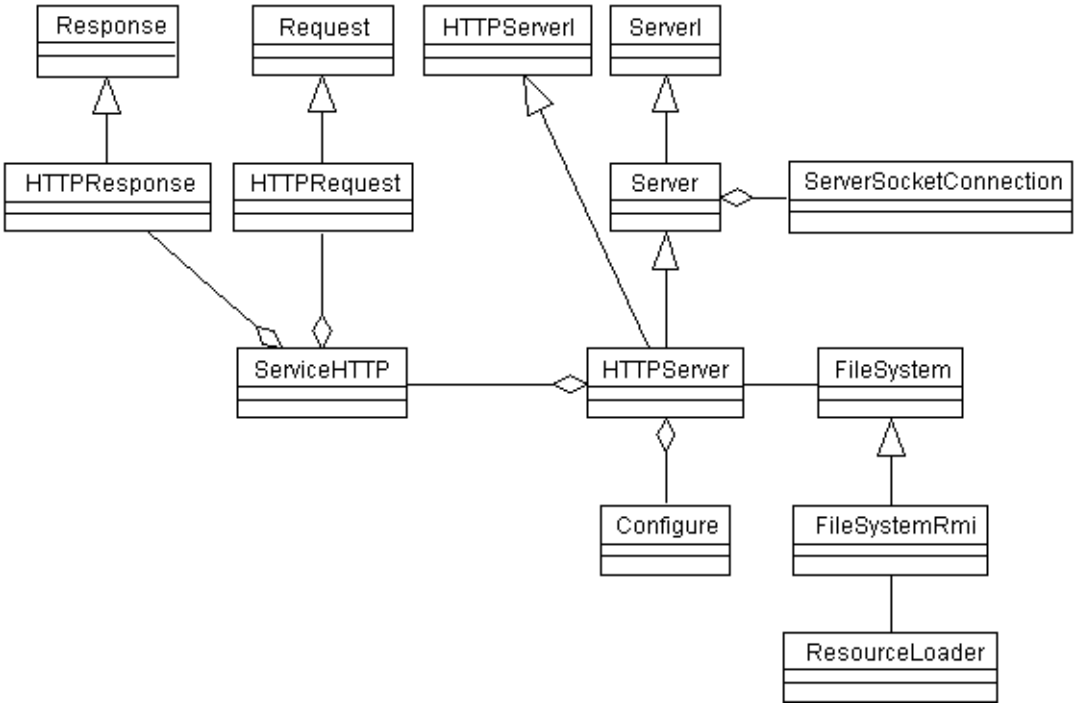

Fig. 3. Diagrama de clases servidor HTTP.

A. Configuración

Como ya se ha mencionado es la configuración el primer módulo del servidor HTTP. A fin de controlar las diversas opciones de funcionamiento el servidor incluye un fichero de configuración que parametriza el comportamiento del servidor y controla los tipos MIME reconocidos por el servidor.

Lo primero a especificar en el fichero de configuración es el `DefaultPort` o puerto por defecto en el que se lanzará el servidor. A continuación el campo `Server` indica el nombre del servidor, que será el valor indicado en la cabecera de respuesta HTTP `Server`. Mediante `maxProcessors` se permite seleccionar el número máximo de conexiones que va a poder atender simultáneamente[2]. Por último a través de `connectionTimeOut` se controla el tiempo que puede pasar inoperativa una conexión abierta.

Una vez establecidas las variables de configuración del servidor se indican los tipos MIMEs del servidor, indicando en primer lugar el tipo y a continuación la extensión de los archivos de ese tipo.

Un ejemplo de fichero de configuración podría ser el siguiente:

El servidor buscará en esta lista el tipo MIME del documento solicitado, estableciendo como tipo por defecto[3] .html.

B. Sistema de archivos

J2ME MIDP define una simple base de datos de registros denominada Record Management System (RMS) con el objetivo de poder almacenar información una vez que el MIDlet finalice.

La unidad de almacenamiento básica dentro de RMS es el `Record` que es almacenado en una base de datos especial denominada `Record Store`. Un `Record` (registro) como su nombre indica es un simple array de bytes, mientras que un `Record Store` es un fichero binario que contiene una colección de registros.

La implementación MIDP proporciona diversas funcionalidades para el manejo de estos registros, aunque no incluye ningún soporte para el manejo de ficheros en su modo tradicional.

[2]Este dato es relativo, ya que existe una variable interna del MIDP que indica el número máximo de conexiones permitidas
[3]Esto se dará en la situación de que el tipo solicitado sea desconocido por el servidor

```
DefaultPort: 80
Server: Servidor Experimental J2ME
maxProcessors: 6
connectionTimeOut: 60000

text/plain .txt
application/pdf .pdf
application/msword .doc
application/pdf: .pdf
application/postscript: .ps
application/vnd.ms-excel: .xls
application/vnd.ms-powerpoint: .ptt
aplication/x-gzip: .gzip
application/x-java-archive: .jar
application/x-java-serialized-object: .rms
application/x-java-vm: .class
audio/basic .snd
audio/x-aiff .aif
audio/x-wav .wav
audio/midi .mid
text/html .html
text/plain .txt
image/gif .gif
image/jpeg .jpeg
image/png .png
image/tiff .tiff
image/x-xbitmap .bmp
video/mpeg .mpg
video/quicktime .qtm
```

Fig. 4. Ejemplo fichero de configuración

Para poder proporcionar los recursos solicitados por los clientes HTTP ha sido necesario crear una pseudo sistema de archivos sobre RMS. Este sistema de archivos se ha implementado siguiendo los patrones **Strategy**[4] y el patrón **Singleton**[5].

Mediante la implementación realizada se podría cambiar el sistema de ficheros de RMS a cualquier tipo soportado por el dispositivo (XML, etc) simplemente realizando una nueva clase que implemente su funcionalidad concreta.

El problema por el que se hace necesario el desarrollo de un sistema de ficheros en J2ME reside en el hecho de que no existe un acceso directo a ficheros sobre MIDP. Si a esto se añade que entre los clientes que posiblemente utilicen un servidor de este tipo se encuentran los PCs, que manejan un sistema de rutas jerárquico, se ha necesitado **mapear** el formato de rutas manejado por los navegadores a un nuevo formato establecido sobre RMS.

Las rutas de un sistema de archivos cualquiera forma una estructura jerárquica de varios niveles de anidamiento. Al carecer de esta característica en MIDP se ha considerado que cualquier recurso solicitado va a corresponder a un `Record`, siendo la estructura jerárquica en la cual se encuentra este

[4]Patrón que proporciona gran flexibilidad a la hora de implementar diferentes estrategias, en este caso políticas de almacenamiento y recuperación de datos

[5]Garantiza la existencia de una única instancia de una determinada clase en el sistema, esto puede ser útil por ejemplo, para asegurar que existe un único fichero de configuración

recurso un `RecordStore`. Esto se puede ilustrar con la figura 5.

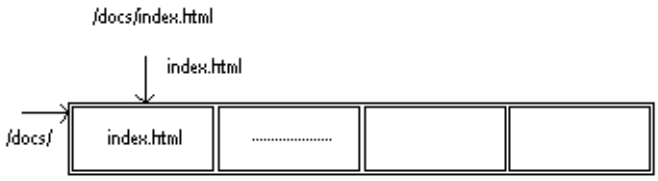

Fig. 5. Ejemplo de mapeado de archivos

Si el recurso solicitado es `/docs/index.html`, el documento `index.html` corresponderá a un record almacenado dentro del `RecordStore` denominado `/docs/`. Cualquier petición del tipo `/docs/documento` se buscará dentro de `/docs/`.

Se ha considerado que peticiones del tipo `/documento` equivalen a solicitar un recurso que se encuentra en la carpeta `/docs/`, es decir, la "carpeta" por defecto es `/docs/`.

Inicialmente existen dos carpetas dentro del servidor, siendo la primera la encargada de almacenar los documentos HTML a servir (se ha llamado a esta carpeta `/docs/`). En esta carpeta se almacenan todas las páginas con formato HTML. En segundo lugar está `/images/` donde se encuentran las imágenes que puedan incluir los documentos de la carpeta `/docs/` o cualquier otra imagen. Estas carpetas han sido creadas a modo de ejemplo, ya que los usuarios podrán crear las carpetas y ficheros necesarios acorde a su estructura de directorios.

Un aspecto importante es el hecho de que cada archivo está compuesto por tres atributos (nombre, longitud y permiso) y un campo de bytes que corresponde al documento en sí. La longitud máxima de un registro es de 65000 bytes y está definida por la implementación.

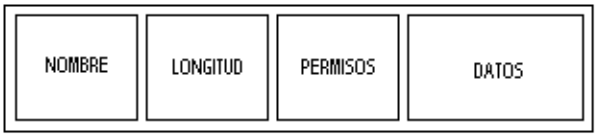

Fig. 6. Atributos de un archivo

El atributo `nombre` sirve para identificar cada fichero de manera única, de modo que no pueden existir dos con el mismo nombre dentro de una misma carpeta. Aunque el atributo `longitud` se puede calcular a partir del tamaño de un registro de RMS, se ha optado por incluirlo para agilizar la recuperación de los archivos. Por último el atributo `permisos` sirve para que no todos los ficheros incluidos en el servidor puedan ser solicitados por clientes HTTP. Esto último se debe a que en RMS no existe un control de acceso establecido con el que proteger determinados registros, con lo que de no existir el atributo `permisos`, cualquier documento existente dentro del servidor podría ser proporcionado. El permiso puede ser *público* si el documento es accesible mediante HTTP o *privado* en caso contrario.

Cabe mencionar que al establecer los permisos a nivel de fichero, no existe ningún control sobre las carpetas, puede haber documentos tanto públicos como privados dentro de una misma carpeta.

Respecto a la granularidad de los permisos, puede seguirse cualquier política de acceso simplemente tratando el atributo permiso del fichero. Se podrían crear grupos de usuarios, propietarios del archivo, etc. Inicialmente sólo se han establecido permisos "público" y "privado", aunque en futuras revisiones cabría ampliar el rango de permisos existentes.

Fig. 7. Sistema de Archivos sobre RMS

C. Tratamiento de peticiones

El tercer y último módulo que compone el servidor J2ME va a ser el encargado de llevar a cabo el tratamiento de las peticiones propiamente dichas. Este módulo reconocerá las conexiones entrantes y llevará a cabo las tareas pertinentes para resolver las peticiones HTTP recibidas.

En los siguientes apartados se van a comentar cuáles son los métodos soportados por el servidor, así como las cabeceras de respuesta con las que va a poder responder.

C.1 Métodos soportados

Inicialmente se encuentran implementados los métodos GET, HEAD y OPTIONS.

Mediante el método GET se va a poder solicitar cualquier recurso que se encuentre en el sistema de archivos del servidor HTTP J2ME. Las peticiones realizadas por el servidor van a realizarse con "tipo de acceso" *público* no pudiendo acceder a aquéllos recursos con acceso *privado* [6].

Método	**Descripción**
GET	Solicitud de un documento
HEAD	Información de un documento
OPTIONS	Métodos disponibles

TABLE I
MÉTODOS HTTP SOPORTADOS

El método OPTIONS ofrece información sobre los métodos disponibles en el servidor y mediante HEAD se puede obtener información relativa a un recurso específico.

C.2 Cabeceras de petición

El servidor captura todas las cabeceras recibidas en las peticiones de los clientes. Estas cabeceras son almacenadas para que en futuras versiones de la aplicación se realice un tratamiento adecuado de las mismas.

[6]En posteriores versiones se implementará un mayor nivel de granularidad, creando una mayor variedad de tipos de acceso

Por el momento ninguna cabecera es tratada. Es la línea de petición lo único que va a tener sentido para el servidor, que analizará el método, el recurso y la versión HTTP del cliente que la realice.

C.3 Cabeceras de respuesta

Las posibles cabeceras de respuesta del servidor HTTP van a ser las mostradas en la tabla II.

Cabecera	**Valor**
Server	Servidor J2ME Experimental
Connection	close
Language	en
Cache-Control	no-cache
Content-Type	tipo mime obtenido del fichero de configuración
Content-Length	tamaño del documento solicitado

TABLE II
CABECERAS DE RESPUESTA DEL SERVIDOR HTTP

C.4 Errores soportados

Los errores que devuelve el servidor son del tipo 2XX, 4XX y 5XX, ya que no existen respuestas parciales ni redirecciones.

Los códigos devueltos son los mostrados en la tabla III.

Resultado	**Código**	**Descripción**
HTTP_OK	200	La petición es correcta, se muestra el resultado
HTTP_BAD_REQUEST	400	La petición es incorrecta
HTTP_FORBIDDEN	403	No se tiene acceso al recurso solicitado
HTTP_NOT_FOUND	404	El recurso solicitado no existe
HTTP_BAD_METHOD	405	El método empleado no existe
HTTP_INTERNAL_ERROR	500	Se ha producido algún error interno en el servidor
HTTP_NOT_IMPLEMENTED	501	El método especificado no se encuentra implementado
HTTP_UNAVAILABLE	503	No es posible atender la solicitud debido a sobrecarga del sistema. Si se repite la solicitud más adelante es posible que pueda ser atendida
HTTP_VERSION	505	El servidor no entiende la versión HTTP del cliente

TABLE III
CÓDIGOS DE RESPUESTA DEL SERVIDOR HTTP

En futuras versiones del servidor cabe la posibilidad de contemplar un mayor rango de errores. Inicialmente se contemplan los errores relativos a comandos incorrectos, fallos del servidor así como

accesos ilegales a recursos del sistema (inexistencia de un documento o acceso no permitido a un recurso).

C.5 Threads

Aunque J2ME no soporta grupos de threads, los threads indivualmente sí son admitidos por esta plataforma. J2ME incluye diferentes posibilidades de crear hilos de ejecución así como métodos de establecimiento de prioridades y control de los mismos.

La clase encargada de realizar el tratamiento de las peticiones se ha implementado heredando de la clase `java.lang.Thread` e incluyendo el método `run` de modo que cada vez que se acepte una conexión, el servidor creará un nuevo hilo que, de manera independiente, tratará dicha conexión, finalizando su ejecución una vez atendida la solicitud.

IV. Interfaces y clases desarrolladas

En los siguientes apartados se comentan de manera resumida las interfaces y clases que conforman la estructura del servidor HTTP.

A. Interfaces

Los interfaces existentes son los siguientes:

- **ServerI** Define la interfaz básica de cualquier servidor.
- **HTTPServerI** Contiene las variables con los tipos de respuesta HTTP que va a poder devolver el servidor.
- **Request** Define los métodos comunes a cualquier tipo de petición.
- **Response** Define los métodos comunes a cualquier tipo de respuesta.

B. Clases

A continuación se describen brevemente las clases empleadas en el servidor HTTP J2ME:

- **ServiceHTTP**

Es la clase encargada de realizar todo el procesamiento de las peticiones de los clientes. Cada vez que se recibe una conexión se crea un objeto de esta clase que se lanza como un thread independiente, finalizando una vez atendida la petición.
El número de servicios HTTP que se pueden lanzar va a estar limitado por la variable `maxProcessors` del fichero de configuración así como la variable interna `com.sun.midp.io.http.max_persistent_connections`[7]

- **ServerSocketConnection**

Clase encargada de tratar todo lo relacionado con los sockets tipo servidor o pasivos. Permanece a la escucha y devuelve las conexiones recibidas.

- **Server**

Clase genérica que representa un servidor cualquiera e implementa el interfaz ServerI. Cualquier servicio deberá heredar de ella e implementar su funcionalidad concreta[8].

- **HTTPServer**

Representa un servidor HTTP completo que acepta peticiones y lanza hilos de ejecución para atender las solicitudes.

- **HTTPRequest**

Realiza todas las tareas relacionadas con el tratamiento de las peticiones HTTP (lectura, análisis de las cabeceras, comprobación de los métodos, etc).

- **HTTPResponse**

[7]La variable interna del sistema es la que controla realmente el número de conexiones simultáneas que pueden abrirse, aunque a nivel de aplicación la variable del fichero de configuración es la que permite un número determinado de conexiones simultáneas, siempre que no se superen las establecidas internamente

[8]La clase HTTPServer hereda de Server y añade la funcionalidad específica de un servidor HTTP.

Lleva a cabo la creación de una respuesta HTTP 1.0, creando las cabeceras de respuesta pertinentes e incluyendo en su caso el recurso solicitado por el usuario.

- **FileSystem**

Esta clase define la funcionalidad de un sistema de archivos básico. Siguiendo el patrón *Strategy*[9] va a permitir que el sistema de archivos pueda ser de cualquier tipo definido por el programador (RMS, XML, bases de datos, etc).

- **FileSystemRms**

Representa un sistema de archivos concreto. Éste se va a basar en la funcionalidad proporcionada por el paquete `javax.microedition.rms` que incluye una serie de clases e interfaces con el objetivo de permitir el almacenamiento persistente en dispositivos MIDP.

- **Configure**

Contiene la configuración del servidor HTTP obtenida a través de un fichero de texto incluido junto al servidor. Esta clase proporciona los métodos necesarios para consultar información relativa a la situación actual del servidor así como sobre los tipos MIME soportados.

- **ResourceLoader**

Clase encargada de obtener recursos externos como pueden ser imágenes, archivos de texto, sonidos, etc. Va a emplearse para realizar la carga inicial de los recursos disponibles en el servidor HTTP.

- **Servidor**

Clase principal del sistema, que a modo de prueba establece una serie de recursos en el servidor y lanza un servidor HTTP.

V. Ejemplo de funcionamiento

En este apartado se muestra un sencillo ejemplo del tratamiento de una petición HTTP empleando para ello el emulador incluido en el Wireless Toolkit.

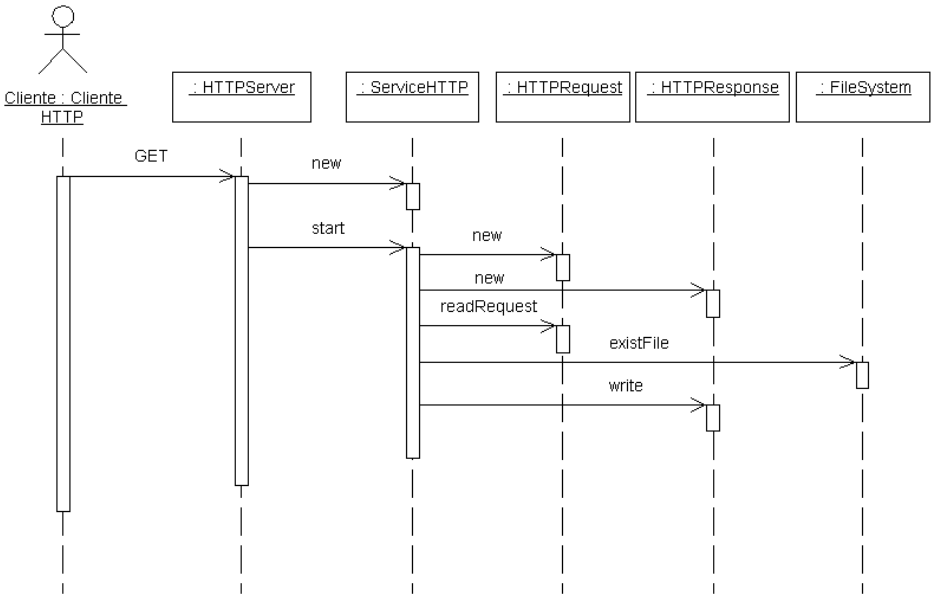

Fig. 8. Diagrama de secuencia del servidor HTTP

A. Descripción

El formato de distribución de la aplicación va a ser un archivo jar a que junto a su descriptor va a ser todo lo que necesita el emulador para ejecutarlo.

[9]Este patrón permite implementar diferentes algoritmos de manera transparente al usuario así como minimizar los cambios a realizar en el caso de querer añadir incluir nuevas "estrategias"

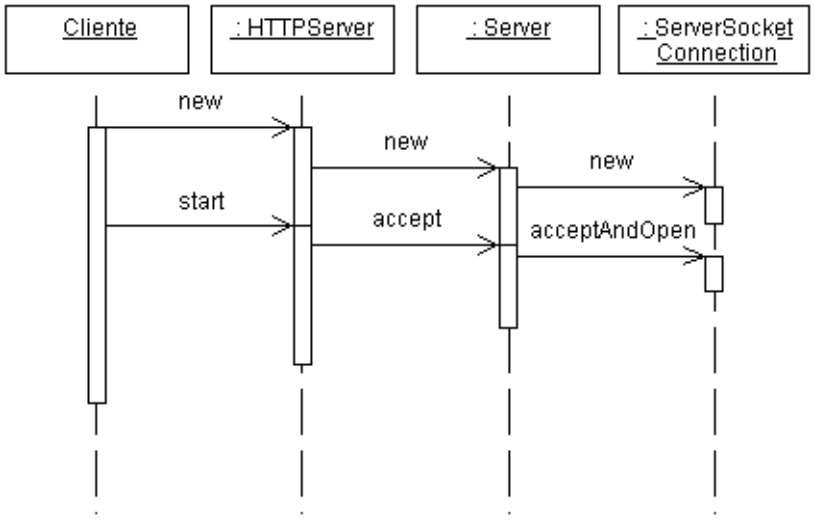

Fig. 9. Diagrama de secuencia (II) del servidor HTTP

Dentro del archivo jar se van a incluir no sólo las clases necesarias, si no aquellos recursos (archivos HTML e imágenes en este caso) que va a incluir el servidor HTTP en su sistema de archivos.

B. Inclusión de recursos en el servidor

En primer lugar y dentro de la aplicación se han incluido una serie de imágenes y archivos HTML que constituirán los recursos que el servidor va a poder proporcionar[10] inicialmente. Estos recursos se incluyen en el MIDlet Suite a través de las clases `ResourceLoader` y `FileSystem`.

Para la inclusión de estos archivos se han utilizado los métodos proporcionados por el sistema de archivos, `FileSystem` así como por el cargador de recursos `ResourceLoader`

C. Funcionamiento

Al lanzar la aplicación aparece una pantalla como la mostrada en la figura 10.

El servidor ha sido lanzado y permanece a la escucha en el puerto 80 (puerto por defecto), cualquier cliente que se conecte al dispositivo y en este puerto será atendido por el servidor.

A continuación se plantean tres escenarios diferentes siendo el primero el de un usuario que se conecta mediante telnet al puerto 80 y realiza una solicitud incorrecta. El resultado obtenido se muestra en la figura 11.

Como se puede observar, el cliente ha especificado un método no existente, GETH. El servidor informa del error producido con un código del tipo 4XX.

Si el método empleado es correcto y el documento solicitado, `index.html`, existe en el servidor, se obtendría el resultado mostrado en la figura 12.

Por último, al solicitar una imagen el resultado es el mostrado en la figura 13.

En esta sección se han incluido algunas de las pantallas obtenidas con la puesta en funcionamiento del servidor en el Emulador del Wireless Tookit. Adicionalmente el servidor ha sido probado en dispositivos con sistema operativo Palm OS. Para ello se ha convertido el archivo `jar` del servidor a su equivalente `prc`[11]. Una vez se ha dispuesto de la versión del servidor para Palm OS (archivo prc) ha bastado con instalarlo en una Palm con la máquina virtual J2ME de IBM conocida como J9. Las pruebas realizadas sobre estos dispositivos reales han resultado totalmente satisfactorias.

[10]El J2ME Wireless Toolkit situa todos los recursos que se desee estén disponibles en el MIDlet, en la carpeta `res/icons` del directorio de la aplicación del servidor

[11]Constituye el formato de las aplicaciones Palm

Fig. 10. Pantalla de lanzamiento del servidor HTTP

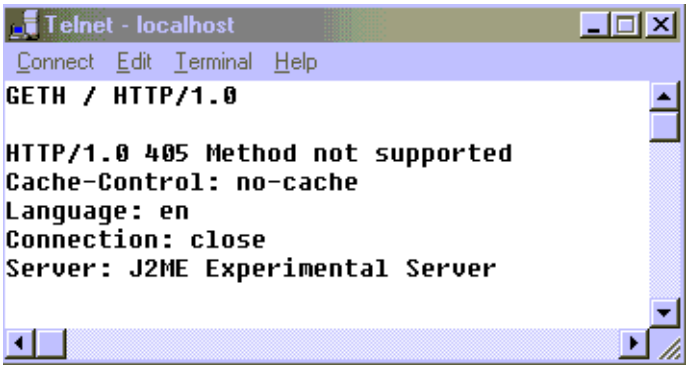

Fig. 11. Solicitud incorrecta mediante telnet.

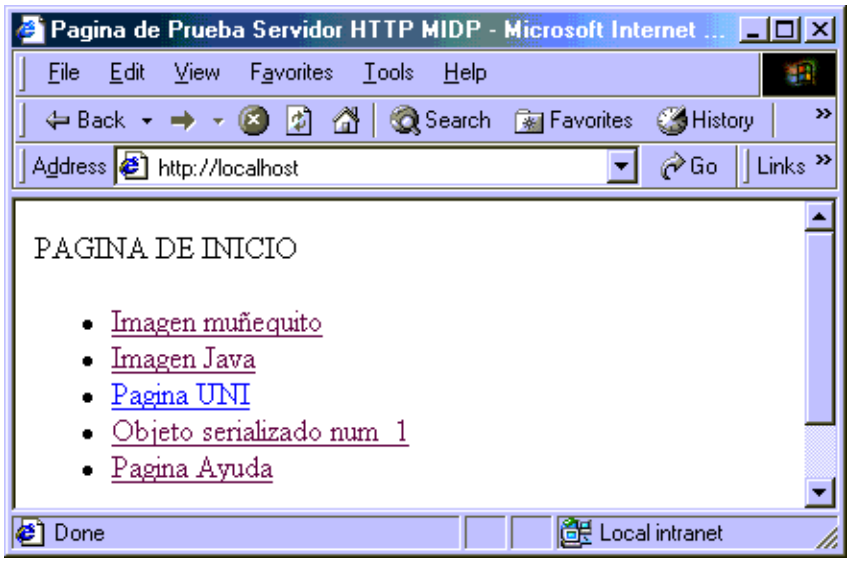

Fig. 12. Solicitud index.html mediante navegador

El servidor desarrollado va a poder emplearse en cualquier aplicación que requiera comunicación entre dispositivos limitados. Posibles ejemplos de aplicaciones serían sistemas de chat, aplicaciones de concertación de citas (sin tener necesidad de disponer de un servidor central ubicado en un PC) o cualquier otra aplicación que se nos pueda ocurrir.

Inicialmente este servidor fue pensado para desarrollar una plataforma de agentes móviles en dispositivos limitados J2ME. Este servidor evitaría la dependencia existente hoy en día en plataformas de agentes móviles sobre dispositivos limitados (LEAP, por ejemplo) de un servidor central en un PC. Este servidor puede no estar disponible en ocasiones (se encuentra caido, no es alcanzable, ...) o simplemente no nos interesa debido al coste de comunicación que supone, ya que deseamos tener una plataforma **autónoma**.

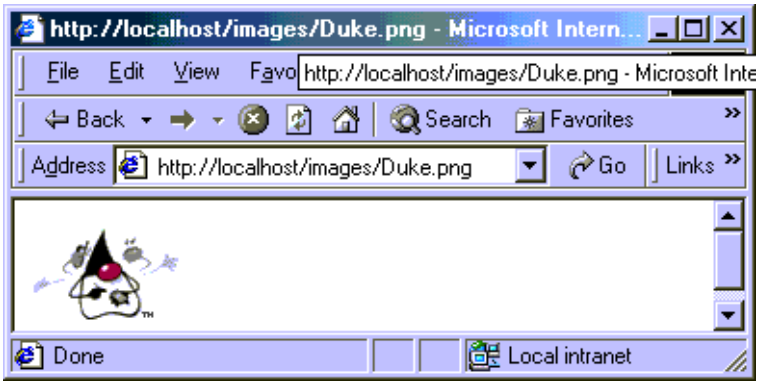

Fig. 13. Solicitud imagen mediante navegador

VI. Conclusiones

El desarrollo de un servidor HTTP sobre J2ME supone una innovación en cuanto al tipo de aplicaciones a desarrollar sobre la nueva plataforma de Java J2ME. El hecho de disponer de un mecanismo de comunicación directa entre dispositivos J2ME sin la necesidad de que existan servidores HTTP intermedios residentes en PCs supone un cambio en el rol que este tipo de dispositivos venía desempeñando hasta ahora, es decir, el de cliente.

Con este servidor HTTP los dispositivos J2ME van a poder ofrecer contenidos así como servicios a otros dispositivos limitados y no limitados. Esta nueva concepción va a permitir desde la compartición de información sobre redes ad-hoc formadas por los dispositivos, hasta la utilización de un terminal móvil J2ME para la consulta del estado de los electrodomésticos J2ME en nuestro hogar.

Agradecimientos

Este trabajo ha sido parcialmente subencionado por el marco de la cátedra Nokia de la Universidad Carlos III de Madrid.

References

[1] E. Giguere. Making connections with the cldc, Feb. 2001. http://www.ericgiguere.com.
[2] P. R. Jan-Peter Stromann, Stephan Hartwig. Wireless microservers. *Pervasive Computing*, pages 58–69, Apr.-June 2002.
[3] G. H. Mahmoud. Secure java midp programming using https, June 2002.
[4] Q. Mahmoud. Midp network programming using http and the connection framework. page 11, Nov. 2000.
[5] Q. H. Mahmoud. Tranporting objects over sockets, Dec. 2002. http://developer.java.sun.com.
[6] S. Microsystems. Connected, limited device configuration.specification version 1.0a, May 2000.
[7] S. Microsystems. Building the mobile information device profile, 2001. http://www.sun.com.
[8] S. Microsystems. Mobile information device profile build configuration, 2001.
[9] S. Microsystems. Running the mobile information device profile, 2001.
[10] S. Microsystems. Java technology and the new world of wireless portals and mcommerce, Feb. 2002.
[11] S. Microsystems. The k virtual machine datasheet, 2002.
[12] M. Morrison. *Wireless Java with J2ME*. SAMS, 2001. Libro de aprendizaje.
[13] E. Ort. Opening a socket connection and midp, Jan. 2000. http://wireless.java.sun.com/midp/questions/rawsocket.