Tema 5: Programación de dispositivos móviles con Java (J2ME)

Parte III: Entrada/Salida

Tecnologías Web

1. Conexión HTTP

Acceso a la red

Tecnologías Web

GCF

- GCF = *Generic Connection Framework*
  - Marco de conexiones genéricas, en `javax.microedition.io`
  - Permite establecer conexiones de red independientemente del tipo de red del móvil (circuitos virtuales, paquetes, etc)
- Cualquier tipo conexión se establece con un único método genérico

```
Connection con = Connector.open(url);
```

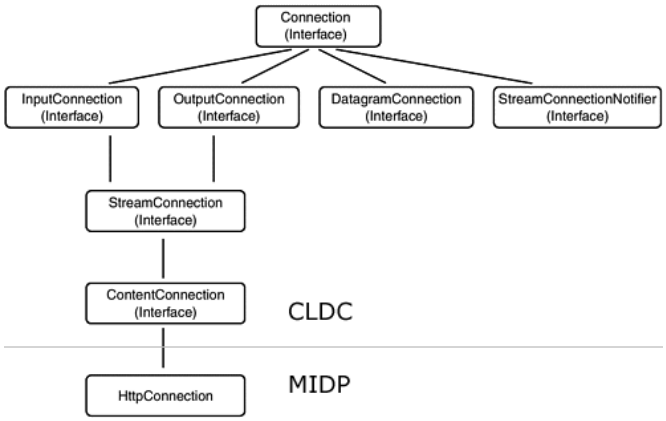
- Según la URL podemos establecer distintos tipos de conexiones

```
http://j2ee.ua.es/pdm                HTTP
datagram://192.168.0.4:6666          Datagramas
socket://192.168.0.4:4444            Sockets
comm:0;baudrate=9600                 Puerto serie
file:/fichero.txt                    Ficheros
```

Tecnologías Web

Tipos de conexiones

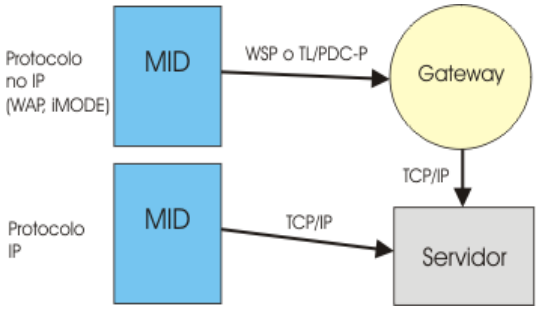
- En CLDC se implementan conexiones genéricas
- En MIDP y APIs opcionales se implementan los protocolos concretos



Tecnologías Web

Conexión HTTP

- El único protocolo que se nos asegura que funcione en todos los móviles es HTTP
  - Funcionará siempre de la misma forma, independientemente del tipo de red que haya por debajo



Tecnologías Web

Leer de una URL

- Abrimos una conexión con la URL

```
HttpConnection con = (HttpConnection)Connector.open(
                        "http://j2ee.ua.es/index.htm");
```

- Abrimos un flujo de entrada de la conexión

```
InputStream in = con.openInputStream();
```

- Podremos leer el contenido de la URL utilizando este flujo de entrada
  - Por ejemplo, en caso de ser un documento HTML, leeremos su código HTML
- Cerramos la conexión

```
in.close();
con.close();
```

Tecnologías Web

Mensaje de petición

- Podemos utilizar distintos métodos
  - `HttpConnection.GET`
  - `HttpConnection.POST`
  - `HttpConnection.HEAD`
- Para establecer el método utilizaremos:

```
con.setRequestMethod(HttpConnection.GET);
```

- Podemos añadir cabeceras HTTP a la petición

```
con.setRequestProperty(nombre, valor);
```

- Por ejemplo:

```
con.setRequestProperty("User-Agent",
          "Profile/MIDP-1.0 Configuration/CLDC-1.0");
```

Tecnologías Web

Mensaje de respuesta

- A parte de leer el contenido de la respuesta, podemos obtener
  - Código de estado

```
int cod = con.getResponseCode();
String msg = con.getResponseMessage();
```

  - Cabeceras de la respuesta

```
String valor = con.getHeaderField(nombre);
```

  - Tenemos métodos específicos para cabeceras estándar

```
getLength()
getType()
getLastModified()
```

Tecnologías Web

Enviar datos

- Utilizar parámetros
  - GET o POST
  - Parejas <*nombre, valor*>

```
HttpConnection con = (HttpConnection)Connector.open(
    "http://localhost:8080/registra?nombre=Pedro&edad=23");
```

  - No será útil para enviar estructuras complejas de datos
- Añadir los datos al bloque de contenido de la petición
  - Deberemos decidir la codificación a utilizar
  - Por ejemplo, podemos codificar en binario con `DataOutputStream`

Tipos de contenido

- Para enviar datos en el bloque de contenido debemos especificar el tipo MIME de estos datos
  - Lo establecemos mediante la cabecera `Content-Type`

```
con.setRequestProperty("Content-Type", "text/plain");
```

  - Por ejemplo, podemos usar los siguientes tipos:

`application/x-www-form-urlencoded`	Formulario POST
`text/plain`	Texto ASCII
`application/octet-stream`	Datos binarios

Codificación de los datos

- Podemos codificar los datos a enviar en binario
  - Establecemos el tipo MIME adecuado

```
con.setRequestProperty("Content-Type",
                       "application/octet-stream");
```

  - Utilizaremos un objeto `DataOutputStream`

```
DataOutputStream dos = con.openDataOutputStream();
dos.writeUTF(nombre);
dos.writeInt(edad);
dos.flush();
```

- Podemos definir una serialización de nuestros objetos para enviarlos por la red

Tecnologías Web

Serialización manual

- CLDC no soporta serialización de objetos
  - Conversión de un objeto en una secuencia de bytes
  - Nos permite enviar y recibir objetos a través de flujos de E/S
- Necesitaremos serializar objetos para
  - Hacer persistente la información que contengan
  - Enviar esta información a través de la red
- Podemos serializar manualmente nuestros objetos
  - Definiremos métodos `serialize` y `deserialize`
  - Utilizaremos los flujos `DataOutputStream` y `DataInputStream` para codificar y descodificar los datos del objeto en el flujo

Tecnologías Web

Serializar

- Escribimos las propiedades del objeto en el flujo de salida

```
public class Punto2D {
  int x;
  int y;
  String etiqueta;
  ...
  public void serialize(DataOutputStream dos) throws IOException {
    dos.writeInt(x);
    dos.writeInt(y);
    dos.writeUTF(etiqueta);
    dos.flush();
  }
}
```

Leer datos de la respuesta

- Contenido de la respuesta HTTP
  - No sólo se puede utilizar HTML
  - El servidor puede devolver contenido de cualquier tipo
  - Por ejemplo, XML, ASCII, binario, etc
- Si el servidor nos devuelve datos binarios, podemos descodificarlos mediante `DataInputStream`

```
DataInputStream dis = con.openDataInputStream();
String nombre = dis.readUTF();
int precio = dis.readInt();
dis.close();
```

- Podría devolver objetos serializados
  - Deberíamos deserializalos con el método adecuado

Deserializar

- Leemos las propiedades del objeto del flujo de entrada
- Debemos leerlas en el mismo orden en el que fueron escritas

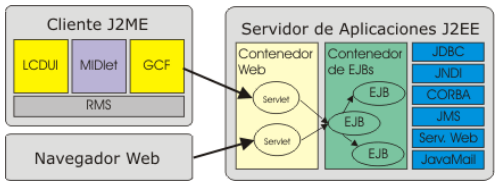
```
public class Punto2D {
  ...
  public static Punto2D deserialize(DataInputStream dis)
                                        throws IOException {
    Punto2D p = new Punto2D();
    p.x = dis.readInt();
    p.y = dis.readInt();
    p.etiqueta = dis.readUTF();

    return p;
  }
}
```

Tecnologías Web

Comunicación con el servidor

- El MIDlet cliente utilizará:
  - GCF para comunicarse con el servidor web
  - LCDUI para la interfaz con el usuario
  - RMS para almacenar datos de forma local en el móvil
- En la aplicación web J2EE utilizaremos:
  - Un servlet que se comunique con el cliente J2ME
    - Debe proporcionar sólo información en un formato entendible por la aplicación
  - Podemos definir otro servlet para acceder mediante una interfaz web
    - Debe proporcionar un documento web para ser visualizado en un navegador
  - Podemos reutilizar desde ambos servlets la misma lógica de negocio implementada mediante componentes Java



Tecnologías Web

Codificación de los datos

- En la comunicación con el servidor (servlet) se debe acordar una codificación de los mensajes que ambos entiendan.
- Binario
  - Mensajes compactos y fáciles de analizar.
  - Alto acoplamiento.
  - Podemos utilizar la serialización de objetos definida en MIDP
    - Asegurarse de que el objeto es compatible con J2ME y J2EE
    - Tanto en el cliente como en el servidor se deberán utilizar los mismos métodos de serialización
- XML
  - Mensajes extensos y complejos de analizar por un móvil.
  - Bajo acoplamiento.

Mantenimiento de sesiones

- Las sesiones normalmente se mantienen con elementos que gestionan los navegadores web como las cookies
- Para poder utilizar sesiones deberemos implementar en nuestro cliente alguno de los métodos existentes
  - Cookies
  - Reescritura de URLs
- Las cookies en algunos casos son filtradas por gateways
  - Será más conveniente utilizar reescritura de URLs

Reescritura de URLs

- En el lado del servidor debemos obtener la URL rescrita

```
String url_con_ID = response.encodeURL(url);
```

- Se adjunta un identificador a dicha URL que identifica la sesión en la que nos encontramos
- Devolvemos la URL al cliente
  - Por ejemplo, codificada en la respuesta HTTP

```
dos.writeUTF(url_con_ID);
```

- La próxima vez que nos conectemos al servidor deberemos utilizar la URL rescrita
  - De esta forma el servidor sabrá que la petición la realiza el mismo cliente y podrá mantener la sesión

2. Conexión de mensajes

Envíar y recibir SMSs

Conexión de mensajes

- Con WMA podremos crear conexiones para enviar y recibir mensajes de texto SMS
- Utilizaremos una URL como

```
sms://telefono:[puerto]
```

- Creamos la conexión

```
MessageConnection mc = (MessageConnection)
       Connector.open("sms://+34555000000");
```

Tecnologías Web

Envío de mensajes

- Componemos el mensaje

```
String texto =
   "Este es un mensaje corto de texto";
TextMessage msg = mc.newMessage(mc.TEXT_MESSAGE);
msg.setPayloadText(texto);
```

- El mensaje no deberá pasar de 140 bytes
  - Si se excede, podría ser fraccionado
  - Si no puede ser fraccionado, obtendremos un error
- Enviamos el mensaje

```
mc.send(msg);
```

Tecnologías Web

Recepción de mensajes

- Creamos conexión de mensajes entrantes

```
MessageConnection mc = (MessageConnection)
    Connector.open("sms://:6226");
```

- Recibimos el mensaje

```
Message msg = mc.receive();
```

- Esto nos bloqueará hasta la recepción
  - Para evitar estar bloqueados, podemos utilizar un listener
  - Con un `MessageListener` se nos notificará de la llegada de mensajes
  - Deberemos utilizar un hilo para la recepción del mensaje

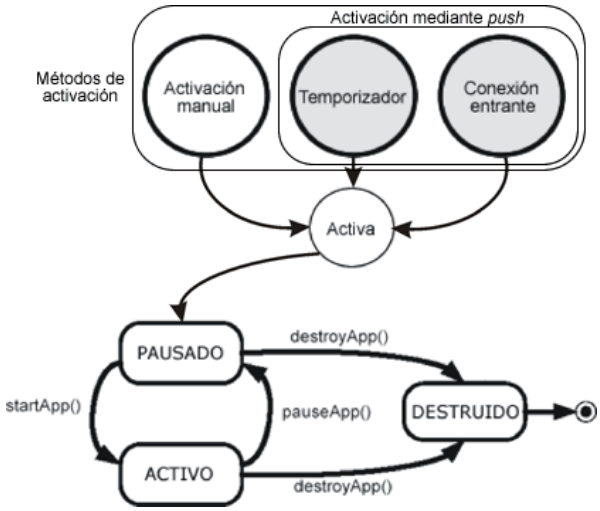
3. Push

Activación de aplicaciones por push

Push vs Pull

- Modelo pull
  - Cuando el usuario quiera obtener datos, debe “tirar” de ellos
    - Debe entrar en la aplicación y solicitar dichos datos
  - Por ejemplo, si hemos publicado un mensaje en un foro y queremos ver si alguien nos ha respondido, deberemos abrir la aplicación de foros para consultar las respuestas
- Modelo push
  - Los datos son “empujados” hacia el usuario
    - Puede recibir datos en el momento en que éstos estén disponibles, sin necesidad de solicitarlos
  - Por ejemplo, podríamos hacer que el foro anterior nos enviase un aviso cuando alguien haya publicado una respuesta a nuestro mensaje

Activación por push



Conexiones entrantes

- Podemos hacer que la aplicación se active cuando se produzca una conexión entrante
  - Sockets, datagramas, mensajes
- Normalmente en el móvil no tendremos una IP fija, por lo que los sockets y los datagramas no son adecuados
  - El número de teléfono si es fijo, podemos usar SMS
- Podemos registrar la conexión push de dos formas:
  - Estática, en el fichero JAD

```
MIDlet-Push-1: sms://:6226,tw.subastas.micro.MIDletPrincipal,*
```

  - Dinámica, utilizando la API de `PushRegistry`

```
PushRegistry.registerConnection(url, nombreClaseMIDlet,
  remitentesPermitidos);
```

Tecnologías Web

4. RMS

Almacenamiento persistente

Tecnologías Web

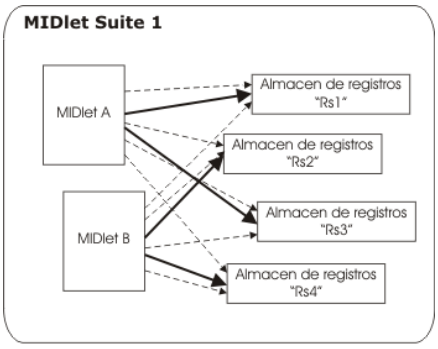
RMS

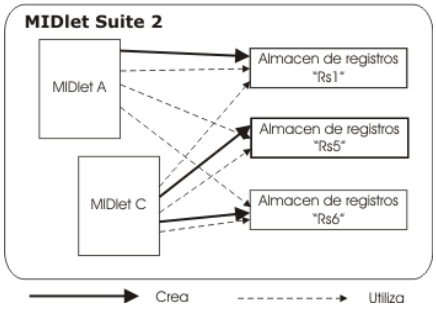
- RMS = *Record Management System*
  - Nos permite almacenar datos de forma persistente
  - Esta API se encuentra en `javax.microedition.rms`
- No se especifica la forma en la que se guardan realmente los datos
  - Deben guardarse en cualquier memoria no volátil
- Los datos se guardan en almacenes de registros
  - Un almacén de registros contiene varios registros
  - Cada registro contiene
    - Un identificador
    - Un *array* de *bytes* como datos

Tecnologías Web

Almacenes de registros

- Un MIDlet puede crear y acceder a varios almacenes
- Los almacenes son privados de cada *suite*





Tecnologías Web

Operaciones con los almacenes

- Abrir/crear un almacén

```
RecordStore rs = RecordStore.open(nombre, true);
```

- Cerrar un almacén

```
rs.close();
```

- Listar los almacenes disponibles

```
String [] nombres = RecordStore.listRecordStores();
```

- Eliminar un almacén

```
RecordStore.deleteRecordStore(nombre);
```

Tecnologías Web

Conjunto de registros

- Cada almacén contendrá un conjunto de registros
- Cada registro tiene
  - Identificador
    - Valor entero
  - Datos
    - *Array* de *bytes*

Identificador	Datos
1	A4 5D 12 09 ...
2	32 3E 1A 98 ...
3	FE 26 3B 45 ...

- El identificador se autoincrementará con cada inserción
- Deberemos codificar los datos en binario para añadirlos en un registro
  - Utilizar objetos `DataInputStream` y `DataOutputStream`
  - Podemos utilizar los métodos de serialización de los objetos

Tecnologías Web

Añadir datos

- Codificar los datos en binario

```
ByteArrayOutputStream baos =
    new ByteArrayOutputStream();
DataOutputStream dos =
    new DataOutputStream(baos);
dos.writeUTF(nombre);
dos.writeInt(edad);
byte [] datos = baos.toByteArray();
```

- Añadir los datos como registro al almacén

```
int id = rs.addRecord(datos, 0, datos.length);
```

```
rs.setRecord(id, datos, 0, datos.length);
```

Tecnologías Web

Enumeración de registros

- Normalmente no conoceremos el identificador del registro buscado *a priori*
  - Podremos recorrer el conjunto de registros para buscarlo
  - Utilizaremos un objeto `RecordEnumeration`

```
RecordEnumeration re =
    rs.enumerateRecords(null, null, false);
```

  - Recorremos la enumeración

```
while(re.hasNextElement()) {
  int id = re.nextRecordId();
  byte [] datos = rs.getRecord(id);
  // Procesar datos obtenidos
  ...
}
```

Tecnologías Web

Consultar y borrar datos

- Leemos el registro del almacén

```
byte [] datos = rs.getRecord(id);
```

- Descodificamos los datos

```
ByteArrayInputStream bais =
    new ByteArrayInputStream(datos);
DataInputStream dis = DataInputStream(bais);
String nombre = dis.readUTF();
String edad = dis.readInt();
```

- Eliminar un registro

```
rs.deleteRecord(id);
```

Tecnologías Web