8 ESCENARIO DE PRUEBAS

8.1 Introducción

En este capítulo pasamos a detallar las pruebas realizadas, teniendo en cuenta que pretendemos cumplir los siguientes objetivos:

- Transmitir imágenes de vídeo mediante Servicios Web XML partiendo de un servidor de vídeo existente.

- Utilizar Servicios Web XML en la J2ME, de modo que se pueda comunicar un servidor fijo con un cliente móvil.

- Integrar en nuestro programa cliente un descodificador JPEG ya desarrollado en un proyecto anterior.

- Encontrar una implementación que nos permita recibir imágenes de vídeo con una calidad aceptable, bajo tiempo de respuesta y consumo de memoria limitado.

8.2 Escenario de pruebas

Para cumplir los objetivos anteriores vamos a crear un escenario de pruebas en el que realizaremos las siguientes acciones:

1. Creación y despliegue de un servicio web XML que atienda peticiones por parte del cliente y se comunique con el servidor de vídeo para devolver la imagen.

2. Creación del cliente móvil, el cual realiza peticiones al servidor web, recibe la respuesta y representa por pantalla la imagen recibida.

3. Simulación del cliente móvil: Se realizarán peticiones de imágenes en formato PNG y en formato JPG. Asimismo, para cada uno de estos formatos de imagen utilizaremos la codificación Base64 y una codificación basada en un array de enteros.

4. Recogida de datos en las dos variables que más afectan al rendimiento del cliente: tiempo entre imágenes y consumo de memoria en tiempo de ejecución.

Toda esta secuencia de pruebas se realizará por separado para las dos tecnologías de servicios web en J2ME que hemos estudiado: kSOAP y JSR-172.

El entorno en que se desarrollarán todas estas pruebas está compuesto por un ordenador personal y una webcam. En el PC, que tendrá conectada la webcam, se ejecutará el servidor de vídeo, el servidor web Apache Tomcat y el emulador J2ME Wireless Toolkit. El servidor de vídeo se comunica con Apache por HTTP, mientras que el emulador se comunica con Apache mediante SOAP sobre HTTP. El montaje se corresponde con el siguiente esquema de comunicaciones:

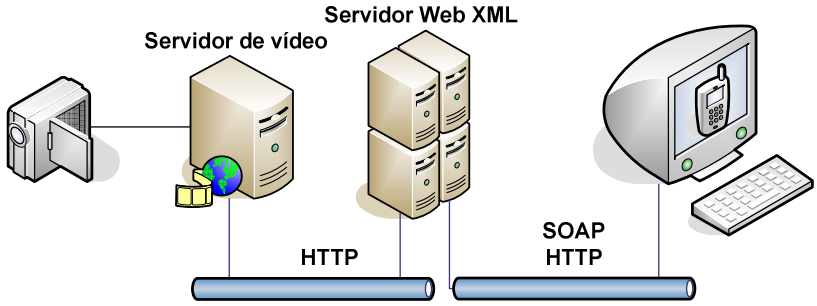


Figura 8.1: Escenario de pruebas

8.3 Escenario kSOAP

8.3.1 Desarrollo del servicio web

El servicio web desarrollado está compuesto de una sola clase, llamada *VideoKsoap* y cuatro métodos, dos de los cuales se utilizarán para escuchar las peticiones del cliente. El método más importante de la clase es el llamado *conectar*, que realiza la petición al servidor de vídeo especificando el formato y el tamaño de la imagen, y devuelve un array de bytes con la imagen solicitada.

```
public byte[] conectar(String formato, int tam) throws Exception{

        InputStream is = null;
        DataInputStream dis = null;
        int len = 0;
        byte datos[] = null;
```

```
        // Hacemos la petición al servidor de vídeo
        URL url = new URL("http", ip, puerto, "/?format=" + formato +
                "?resize=" + tam + "*" + tam + "?" +
getCalidad(formato));
        HttpURLConnection conexion =
(HttpURLConnection)url.openConnection();
        int rc = conexion.getResponseCode();
        if (rc != HttpURLConnection.HTTP_OK) {
            conexion.disconnect();
            throw new Exception("Codigo de respuesta HTTP: " + rc);
        }

        // Comprobamos el tipo de archivo devuelto
        String tipo = conexion.getContentType();
        if (!tipo.equals("image/png") && !tipo.equals("image/jpeg")) {
            throw new Exception("Se esperaba una imagen, " +
                    "pero se ha recibido " + tipo);
        }

        len = conexion.getContentLength();

        // Almacenamos la información recibida en un array de bytes
        if (len > 0) {
            is = conexion.getInputStream();
            dis = new DataInputStream(is);
            datos = new byte [len];
            dis.readFully(datos);
        }

        // Cerramos el flujo de entrada
        if (dis != null) {
            dis.close();
            dis = null;
        }

        // Cerramos la conexión
        if (conexion != null) {
            conexion.disconnect();
            conexion = null;
        }
        return datos;
    }
```

La petición al servidor de vídeo necesita un parámetro que le indique la calidad de la imagen. Este parámetro va a ser fijo para cada uno de los formatos, y se especifica en el método *getCalidad*:

```
public String getCalidad(String formato) {

        String cadena;
        if (formato == "jpg")
            cadena = "quality=50";
        else if (formato == "png")
            cadena = "comp=max";
```

```
        else
            cadena = "";

        return cadena;
    }
```

Los dos métodos restantes son los encargados de recibir la petición por parte del cliente. La diferencia entre ambos radica en que la codificación con la cual se enviará la imagen es distinta en cada caso.

En el primer caso vamos a enviar al cliente un array de bytes con la imagen solicitada. El módulo Axis en el lado del servidor y el paquete kSOAP en el lado del cliente se encargarán de codificar y descodificar automáticamente este array en formato Base64:

```
public byte[] getImagenBase64(String formato, int tam) throws
Exception{

        byte[] datos = conectar(formato, tam);

        return (datos);
    }
```

En el segundo de los casos enviaremos la imagen en un array de enteros. El fundamento de esta codificación es simple: cada byte que forma la imagen puede ser representado por un número entero de 8 bits. De este modo podemos formar un array de tantos enteros como bytes tenga la imagen. El método que realiza esta conversión es el siguiente:

```
public int[] getImagenIntArray(String formato, int tam) throws
Exception{

        byte datos[] = null;
        int cadena[] = null;
        int longitud;

        datos = conectar(formato, tam);
        longitud=datos.length;

        // Pasamos cada byte a su valor entero correspondiente
        cadena = new int[longitud];
        for (int i=0; i<longitud; i++)
            cadena[i]=(int)datos[i];

        return (cadena);
    }
```

Con esto ya tenemos preparado el código fuente de nuestro servicio web. El siguiente paso consiste en desplegarlo, es decir, integrarlo en el servidor web Apache Tomcat y hacer públicos los métodos disponibles.

8.3.2 Despliegue del servicio web

Para desplegar el servicio que acabamos de desarrollar debemos tener el servidor de aplicaciones Apache Tomcat con el módulo Axis correctamente instalado y configurado. Los pasos necesarios para llegar a este punto se explican detalladamente en el capítulo "Guía de instalación".

El PC en el que se realiza la instalación debe tener acceso a Internet, pues será el encargado de recibir y enviar las peticiones al teléfono móvil. Además es recomendable que Tomcat se ejecute en la misma máquina que el servidor de vídeo, aunque no es obligatorio. En este proyecto utilizaremos el mismo PC para implementar ambos servidores.

Una vez arrancado el servidor Apache Tomcat accedemos a la carpeta contenedora de servicios web XML, situada en la ruta *C:\Tomcat 4.1\webapps\axis\WEB-INF\classes.* Creamos aquí un directorio llamado *ServVideoKsoap* destinado a contener la clase del servicio web, así como la información de despliegue necesaria para la publicación. Dentro de este directorio ubicaremos los siguientes archivos:

- **VideoKsoap.java**: Clase que contiene el servicio web detallado anteriormente.

- **deploy.wsdd**: Archivo de despliegue del servicio web. Contiene la información necesaria para que el servidor de aplicaciones Apache Tomcat sepa cómo ejecutar y publicar el servicio web. En este documento XML se especifica el nombre con el que publicará el servicio (en nuestro caso VideoKsoap), el estilo de codificación de la petición (para kSOAP es obligatorio usar el estilo RPC), el espacio de nombres utilizado en los mensajes de petición/respuesta, la clase que implementa el servicio y por último, los métodos públicos que van a estar disponibles para realizar peticiones.

```
<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="VideoKsoap" provider="java:RPC">
      <parameter name="wsdlTargetNamespace"
value="http://video.samples/"/>
      <parameter name="className" value="ServVideoKsoap.VideoKsoap"/>
```

```
      <parameter name="allowedMethods" value="getImagenBase64
getImagenIntArray"/>
  </service>
</deployment>
```

- **undeploy.wsdd**: Archivo que elimina el despliegue del servicio web. Contiene la información necesaria para que el servidor de aplicaciones deje de publicar el servicio web. Este documento XML simplemente contiene el nombre del servicio web que se quiere dejar de publicar.

```
<undeployment
    xmlns="http://xml.apache.org/axis/wsdd/">
  <service name="VideoKsoap"/>
</undeployment>
```

Tras ubicar estos tres archivos en el directorio especificado, compilamos el archivo java:

```
javac VideoKsoap.java
```

A continuación realizamos el despliegue del servicio escribiendo en el intérprete de comandos:

```
java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient -
lhttp://localhost:8080/axis/services/AdminService deploy.wsdd
```

Llegados a este punto es recomendable reiniciar el servidor Apache Tomcat para que todos los cambios se apliquen correctamente. Podemos comprobar que el servicio web ya está disponible accediendo a la página web de servicios desplegados que ofrece Axis en la dirección: *http://127.0.0.1:8080/axis/servlet/AxisServlet*

En la figura siguiente se puede apreciar que además de los nombres de los servicios desplegados, aparecen los métodos que cada uno publica, así como un enlace al documento WSDL generado automáticamente por el servidor.

El documento WSDL es el que indica los métodos a través de los cuales se puede invocar el servici web, así como los parámetros que se necesitan para la petición y los que se devuelven como resultado. Este documento se utiliza sobre todo en programas que generan automáticamente un cliente de servicios web a partir del WSDL dado.

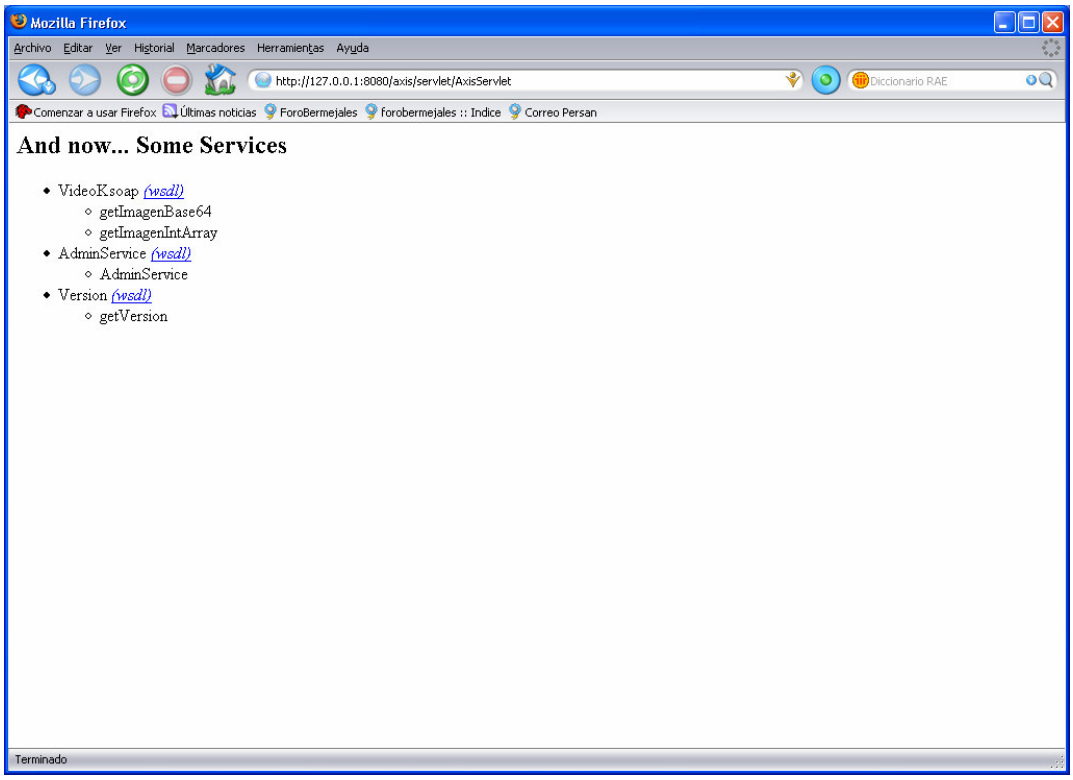


Figura 8.2: Servicios web desplegados en Axis

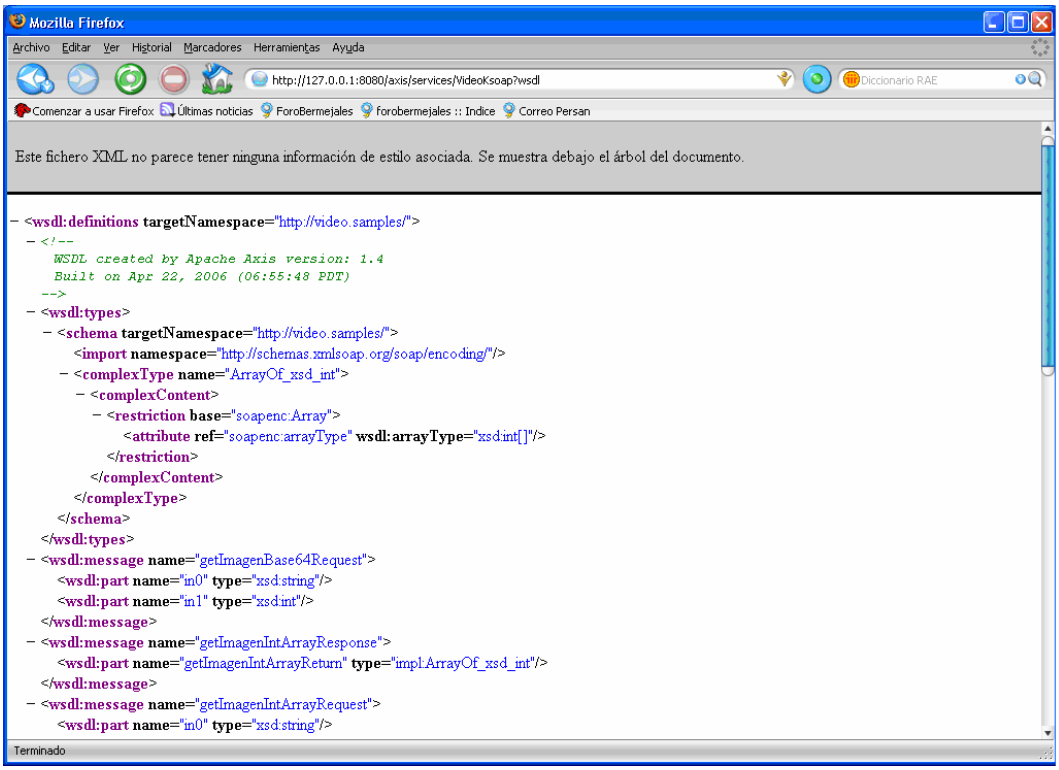


Figura 8.3: Parte del documento WSDL generado automáticamente por Axis

8.3.3 Desarrollo del cliente

El cliente de nuestro servicio web XML será un programa Java especialmente diseñado para dispositivos móviles utilizando J2ME. Su objetivo es el de realizar peticiones al servidor web XML a través de Internet mediante una conexión GPRS. Tras realizar la petición, el cliente queda a la espera de recibir una imagen como respuesta, la cual imprimirá por pantalla inmediatamente después de descodificarla. Mientras no se indique lo contrario, se realizarán peticiones indefinidamente con el fin de recibir una secuencia continua de imágenes, de modo que la representación por pantalla de dicha secuencia ofrezca al usuario una sensación de movimiento.

El cliente está formado por seis clases y la librería ksoap2, que ofrece el soporte de Servicios Web XML, abstrayéndonos de la construcción de mensajes SOAP y de la representación de la información en XML.

Detallamos a continuación cada una de las clases utilizadas:

- **Bienvenida.java**: Contiene una clase que presenta por pantalla un mensaje de bienvenida. Esta pantalla se visualizará nada más iniciarse el programa. El método constructor inicia las variables:

```
     public Bienvenida() {

       // Tipo de fuente utilizado
       fuente = null;

       // Texto de presentacion
       cadena1 = "PFC: Cliente J2ME";
       cadena2 = "Versión kSOAP";
       cadena3 = "Autor: F. Prieto";
       cadena4 = "Tutor: A. Sierra";


       // Altura y anchura de la pantalla del dispositivo
       CanvasWidth = getWidth();
       CanvasHeight = getHeight();
   }
```

Por su parte, el método *paint* representa por pantalla las cadenas de caracteres definidas anteriormente. Como se trata de una pantalla Canvas, es necesario especificar el punto exacto donde se deben ubicar cada una de las líneas de texto:

```
public void paint(Graphics g) {

       // Tipo de letra negrita y tamaño medio
```

```
        fuente = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,
Font.SIZE_MEDIUM);
        g.setFont(fuente);

        // Borramos la pantalla pintandola de blanco
        g.setColor(0xffffff);
        g.fillRect(0, 0, CanvasWidth, CanvasHeight);

        // Escribimos las lineas de texto en negro centradas en la
pantalla
        g.setColor(0);
        int x = CanvasWidth / 2;
        int y = CanvasHeight / 2 - 2*fuente.getHeight();
        g.drawString(cadena1, x, y, Graphics.TOP|Graphics.HCENTER);
        g.drawString(cadena2, x, y+fuente.getHeight(),
                Graphics.TOP|Graphics.HCENTER);

        // Tipo de letra pequeña
        fuente = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,
Font.SIZE_SMALL);
        g.setFont(fuente);

        g.drawString(cadena3, x, y+(3*fuente.getHeight()),
                Graphics.TOP|Graphics.HCENTER);
        g.drawString(cadena4, x, y+(4*fuente.getHeight()),
                Graphics.TOP|Graphics.HCENTER);
}
```

- **ClienteVideo.java**: Ésta es la clase principal del Midlet, ya que implementa los métodos *startApp*, *pauseApp* y *desproyApp*. En esta clase definimos todas las pantallas tipo Screen, destinadas a recopilar la información de usuario.

El método startApp es el que se encarga de inicializar los comandos (botones) y variables, y representar la primera pantalla de usuario, en este caso la pantalla de bienvenida:

```
public void startApp() {

        /* Inicializamos los comandos */
        okToFormDireccion = new Command("Cont", Command.OK, 1);
        okToListCodificacion = new Command("Cont", Command.OK, 0);
        okToImagen = new Command("Cont", Command.OK, 1);
        backToBienvenida = new Command("Atras", Command.BACK, 1);
        backToFormDireccion = new Command("Atras", Command.BACK, 1);
        backToListCodificacion = new Command("Atras", Command.BACK,
1);
        backToListFormato = new Command("Atras", Command.BACK, 1);
        exitCommand = new Command("Salir", Command.EXIT, 1);

        /* Inicializamos las variables */
        ip = "127.0.0.1";
        puerto = "8080";
        fin=false;
```

```
        pantallaImagen=null;

        /* Presentamos la pantalla de bienvenida */
        display = Display.getDisplay(this);
        display.setCurrent(get_bienvenida());
    }
```

Los siguientes métodos inicializan cada una de las pantallas en las que se pedirá información de usuario, como por ejemplo la dirección y puerto del servidor, el formato de la imagen o la codificación a utilizar. La información se recopila mediante listas, formularios y cuadros de texto:

```
/* Pantalla en la que se elige la direccion IP y el puerto del
servidor */
    public Form get_formDireccion() {
        if (formDireccion == null) {
            textFieldIP = new TextField("Direccion IP: ", ip, 15,
TextField.ANY);
            textFieldPuerto = new TextField("Puerto: ", puerto, 5,
TextField.NUMERIC);
            formDireccion = new Form("Direccion", new Item[] {
                textFieldIP, textFieldPuerto });
            formDireccion.addCommand(okToListCodificacion);
            formDireccion.addCommand(backToBienvenida);
            formDireccion.setCommandListener(this);
        }
        return formDireccion;
    }

    /* Pantalla en la que se elige la codificacion de la imagen.
     * Se puede elegien entre Base64 y array de enteros */
    public List get_listCodificacion() {
        if (listCodificacion == null) {
            listCodificacion = new List("Codificacion",
Choice.IMPLICIT,
                    new String[] { "Base64", "IntArray"}, null);
            listCodificacion.addCommand(backToFormDireccion);
            listCodificacion.setCommandListener(this);
        }
        return listCodificacion;
    }

    /* Pantalla en la que se elige el formato de la imagen (png o jpg)
*/
    public List get_listFormato() {
        if (listFormato == null) {
            listFormato = new List("Formato de imagen",
Choice.IMPLICIT,
                    new String[] { "PNG", "JPG" }, null);
            listFormato.addCommand(backToListCodificacion);
            listFormato.setCommandListener(this);
        }
        return listFormato;
    }
```

```
    /* Pantalla de resumen con todas las opciones elegidas */
    public Form get_formResumen() {
        stringResumen = new StringItem("","Serv: " + ip + ":" + puerto
+ "\n" +
                "Codif: " + codificacion + "\n" +
                "Formato: " + formato);
        formResumen = new Form("Preferencias");
        formResumen.append(stringResumen);
        formResumen.addCommand(okToImagen);
        formResumen.addCommand(backToListFormato);
        formResumen.setCommandListener(this);
        return formResumen;
    }
```

A continuación inicializamos las pantallas tipo Canvas, añadiéndole los botones o comandos:

```
/* Pantalla de bienvenida */
    public Bienvenida get_bienvenida() {
        if (pantallaBienvenida == null) {
            pantallaBienvenida = new Bienvenida();
            pantallaBienvenida.addCommand(okToFormDireccion);
            pantallaBienvenida.addCommand(exitCommand);
            pantallaBienvenida.setCommandListener(this);
        }
        return pantallaBienvenida;
    }

    /* Pantalla que presentará las imágenes */
    public Imagen get_Imagen() {
        if (pantallaImagen == null) {
            pantallaImagen = new Imagen();
            pantallaImagen.addCommand(exitCommand);
            pantallaImagen.setCommandListener(this);
        }
        return pantallaImagen;
    }
```

Las acciones que se realizarán al ejecutar un comando deben definirse dentro del método *CommandAction*:

```
/* Acciones que se ejecutaran al activar un comando */
    public void commandAction(Command command, Displayable
displayable) {
        if (command == okToFormDireccion) {
            display.setCurrent(get_formDireccion());
        }

        else if (command == okToListCodificacion) {
            ip = textFieldIP.getString();
            puerto = textFieldPuerto.getString();
            display.setCurrent(get_listCodificacion());
        }
```

```
        else if (command == okToImagen) {
            display.setCurrent(get_Imagen());
            (new Bucle()).start();
        }

        else if (command == backToBienvenida) {
            display.setCurrent(get_bienvenida());
        }

        else if (command == backToListCodificacion) {
            display.setCurrent(get_listCodificacion());
        }

        else if (command == backToListFormato) {
            display.setCurrent(get_listFormato());
        }

        else if (command == backToFormDireccion) {
            display.setCurrent(get_formDireccion());
        }

        else if (displayable == listCodificacion && command ==
listCodificacion.SELECT_COMMAND) {
            switch (get_listCodificacion().getSelectedIndex()) {
                case 0:
                    codificacion = "Base64";
                    display.setCurrent(get_listFormato());
                    break;
                case 1:
                    codificacion = "IntArray";
                    display.setCurrent(get_listFormato());
                    break;
            }
        }

        else if (displayable == listFormato && command ==
listFormato.SELECT_COMMAND) {
            switch (get_listFormato().getSelectedIndex()) {
                case 0:
                    formato = "png";
                    display.setCurrent(get_formResumen());
                    break;
                case 1:
                    formato = "jpg";
                    display.setCurrent(get_formResumen());
                    break;
            }
        }

        else if (command == exitCommand) {
            exitMIDlet();
        }
    }
```

Por último indicamos las acciones necesarias a realizar al salir del Midlet, tales como liberación de memoria:

```
/* Acciones que se ejecutarán al terminar la aplicación */
    public void exitMIDlet() {
        display.setCurrent(null);
        fin = true;
        destroyApp(true);
        notifyDestroyed();
    }

    public void pauseApp() {
    }

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

- **Bucle.java**: Tras la última pantalla de petición de datos, el Midlet ejecuta un bucle de peticiones al servidor web XML y tras recoger la imagen devuelta, utiliza la clase *Imagen* para representarla por pantalla. El métdo que realiza todo este proceso es el método *run*:

```
public void run() {
        try{
            do {
                // Dormimos entre peticiones
                do
                    sleep(intervalo);
                while(Imagen.cerrojo || kSOAP.capturando);

                // Petición al servicio web
                kSOAP peticion = new kSOAP();
                Imagen.datos = peticion.CapturaImagen(direccion);

                // Presentación por pantalla
                if (Imagen.datos != null)
                    ClienteVideo.pantallaImagen.repaint();
                else
                    ClienteVideo.fin=true;

            } while (ClienteVideo.fin == false);
        } catch (InterruptedException e) {
            Alert a = new Alert("Error ", e.toString(), null, null);
            a.setTimeout(Alert.FOREVER);
            ClienteVideo.display.setCurrent(a);
        }
    }
```

- **kSOAP.java**: Esta clase realiza la petición al servidor web con los parámetros que se le han pedido al usuario en las pantallas anteriores.

Primero debemos crear el sobre SOAP, y posteriormente se crea el objeto SOAP que se enviará a través de la red. Tras recibir la imagen, distinguimos el tipo de codificación: si ésta es Base64, devolvemos el array de bytes directamente, en caso contrario debemos pasar el array de enteros a un array de bytes. El método *CapturaImagen* es el que realiza todas estas operaciones:

```
byte[] CapturaImagen(String direccion) {

        Image foto = null;
        DataInputStream dis = null;
        byte data[];
        try {

            capturando = true;
            Runtime.getRuntime().gc();

            // Peticion al servidor web

            //Creamos el SOAP Envelope
            SoapSerializationEnvelope envelope = new
SoapSerializationEnvelope(SoapEnvelope.VER11);

            /* Creamos el objeto SOAP, indicando el
             * espacio de nombres, método y parámetros */
            SoapObject client = new SoapObject( "urn:video.samples",
"getImagen" + ClienteVideo.codificacion);
            client.addProperty("formato", new
String(ClienteVideo.formato));
            client.addProperty("tam", new Integer(ClienteVideo.TAM));

            // Enviamos el sobre al servidor
            envelope.setOutputSoapObject(client);
            ht = new HttpTransport(direccion);
            ht.call("",envelope);

            // Si la codificación es Base64, devolvemos el array
directamente
            if (ClienteVideo.codificacion == "Base64"){
                data =
Base64.decode(envelope.getResponse().toString());
            }

            // Si la codificación es IntArray, pasamos los enteros a
bytes
            else {
                Vector resObj = new Vector();
                resObj =(Vector)envelope.getResponse();
                int numBytes=resObj.size();

                //Transformamos los enteros a bytes y los almacenamos
en un array
                data = new byte[numBytes];
                for (int j=0;j<numBytes;j++)
                    data [j] =
(byte)Integer.parseInt(resObj.elementAt(j).toString());
            }
```

```
            capturando = false;

            return data;

        }catch(Exception exception) {
            Alert a = new Alert("Error E/S", exception.toString(),
null, null);
            a.setTimeout(Alert.FOREVER);
            ClienteVideo.display.setCurrent(a);
            return null;
        }
    }
```

- **Imagen.java**: Clase que representa por pantalla la imagen recibida del servidor web. Si el formato es JPG, se descodifica utilizando la clase DecodificadorJPEG.

El método constructor inicializa las variables relacionadas con el tamaño de la pantalla, así como el descodificador JPG en caso de que la imagen recibida tenga ese formato:

```
public Imagen() {
        datos = null;

        // Calculamos el centro de la pantalla
        CanvasWidth = getWidth();
        CanvasHeight = getHeight();
        centro_x = (CanvasWidth - ClienteVideo.TAM) / 2;
        centro_y = (CanvasHeight - ClienteVideo.TAM) / 2;
        if(centro_x < 0)
            centro_x = 0;
        if(centro_y < 0)
            centro_y = 0;

        // Iniciamos la clase DecodificadorJPEG
        iniciaJPEG();
    }

    void iniciaJPEG(){
        if (ClienteVideo.formato == "jpg") {
            // Inicializo el decodificador JPEG
            decod = new DecodificadorJPEG();
            decod.setAncho(ClienteVideo.TAM);
            decod.setAlto(ClienteVideo.TAM);
        }
    }
```

El método *paint* representa la imagen por pantalla. Mientras se efectúa la representación se activa un cerrojo para que el bucle no efectúe peticiones a la vez, ya que el rendimiento de la aplicación disminuye considerablemente. Si el formato es PNG se llama al método *createImage* incluído en J2ME que se encarga de descodificar dicho

formato. En caso contrario, si la imagen es de tipo JPG, utilizaremos la clase *DecodificadorJPEG.*

```
public void paint(Graphics g) {
        cerrojo = true;

        // Borramos la pantalla pintandola de blanco
        g.setColor(0xffffff);
        g.fillRect(0, 0, CanvasWidth, CanvasHeight);

        // Presenta la imagen por pantalla
        if(datos != null){
            int numBytes = datos.length;

            // Si el formato es png, representamos directamente
            if (ClienteVideo.formato != "jpg"){
                foto = Image.createImage(datos,0,numBytes);
            }

            // Si el formato es jpg, utilizamos el decodificador
            else {

                // Crea el buffer ARGB para la imagen decodificada
                imagenARGB = new
int[ClienteVideo.TAM*ClienteVideo.TAM];

                // Decodifica en imagenARGB[]
                decod.decodifica(datos,imagenARGB, numBytes);

                // Creamos la imagen
                foto =
Image.createRGBImage(imagenARGB,ClienteVideo.TAM,
                        ClienteVideo.TAM,false);

            }
            g.drawImage(foto, centro_x, centro_y, 0);
        }
        cerrojo = false;
    }
```

- **DecodificadorJPEG.java**: Clase que toma un array de bytes con una imagen JPG y la descodifica para poder representarla en pantalla.

Esta clase fue desarrollada en un proyecto final de carrera anterior y no vamos a realizar su análisis detallado debido a su extensión y complejidad. El código fuente de esta clase puede consultarse en el capítulo “Planos de código”.

8.3.4 Simulación del cliente

Para simular el comportamiento de un teléfono móvil real, utilizamos el emulador J2ME Wireless Toolkit 2.2. Con él podemos probar el funcionamiento del cliente, así como realizar medidas de parámetros tales como tiempos de respuesta y consumo de memoria. La instalación y configuración de este programa se detalla en el capítulo "Guía de instalación".

Para simular el programa que hemos desarrollado, ejecutamos el J2ME Wireless Toolkit y creamos un nuevo proyecto, al que llamaremos *ClienteVideoKsoap*. En la siguiente pantalla de configuración tenemos que indicar que el perfil debe ser MIDP 2.0 y la configuración CDLC 1.1. Asimismo es necesario especificar el nombre de la clase principal del Midlet, en nuestro caso *cliente.ClienteVideo*. Automáticamente se habrá creado un directorio con el mismo nombre del proyecto en la ruta *C:\WTK22\apps*. Dentro de la carpeta *src* creamos una a la que llamaremos *cliente*, que corresponde con el paquete en el que contendremos el código fuente. Es en esta carpeta donde ubicaremos los archivos java con las clases especificadas anteriormente.

A continuación situamos el paquete *ksoap2-j2me-core-2.1.1.jar* en el directorio *lib*. Este paquete, como hemos comentado anteriormente, aporta la funcionalidad SOAP al cliente, realizando las tareas necesarias para intercambiar documentos XML y realizar las peticiones al servidor. Puede descargarse desde el siguiente enlace web: http://sourceforge.net/projects/ksoap2/.

Una vez hecho esto, compilamos el proyecto con la opción *Build* y creamos el paquete *jar* precompilado accediendo al menú *Project→ Package→ Create Package.* Esta opción crea tres archivos en el directorio *C:\WTK22\apps\ClienteVideoKsoap\bin*:

- **ClienteVideoKsoap.jar**: Fichero que contiene todas las clases necesarias para la ejecución del cliente.

- **ClienteVideoKsoap.jad**: Fichero descriptor de aplicaciones Java. Se encarga de verificar que su MIDlet Suite asociado se ajusta convenientemente al dispositivo sobre el que será descargado.

- **MANIFEST.MF**: Contiene información sobre el cliente, como por ejemplo el nombre y tamaño de la clase principal, la versión de MIDP y CLCD, nombre del fabricante, etc.

Estos tres ficheros son los que se deben descargar a un teléfono móvil real para ejecutar la aplicación.

8.3.4.1 Descarga OTA

Para la instalación del cliente en un terminal móvil vamos a utilizar el método de descarga vía OTA, aprovechando el hecho de que el servidor de vídeo es también servidor de ficheros y viene además configurado para soportar este tipo de descargas. Para alojar nuestro Midlet-Suite en el servidor es necesario realizar los siguientes pasos:

5. Crear una carpeta dentro del directorio de trabajo del servidor de vídeo y alojar allí los tres archivos que hemos obtenido. En nuestro caso, el directorio de trabajo es *C:*, así que creamos la carpeta *C:/descarga* y copiamos los ficheros mencionados.

6. Editar el archivo *ClienteVideoKsoap.jad*, para que la línea que contiene información sobre la URL muestre lo siguiente:

```
MIDlet-Jar-URL: http://127.0.0.1/descarga/ClienteVideoKsoap.jar
```

7. Crear una sencilla página web con el enlace al archivo *jad.* El archivo debe situarse en el directorio de trabajo del servidor de vídeo, y el contenido se muestra a continuación:

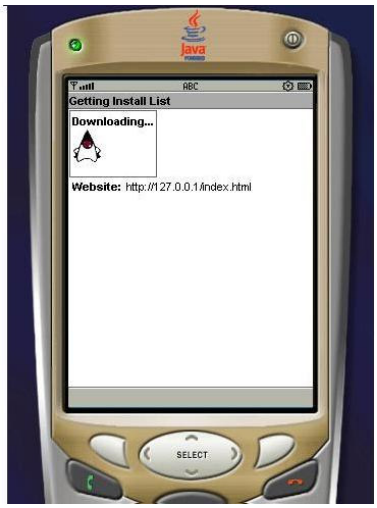
```
<html>

<head>
<title>Cliente Video - Descarga</title>
</head>

<body>
<a href="http://127.0.0.1/descarga/ClienteVideoKsoap.jad">Instalar
Cliente Video Ksoap</a>
</body>

</html>
```

8. Ejecutar el servidor de vídeo, ya que hasta que el servidor no comience a capturar imágenes, los archivos no van a estar disponibles para su descarga.

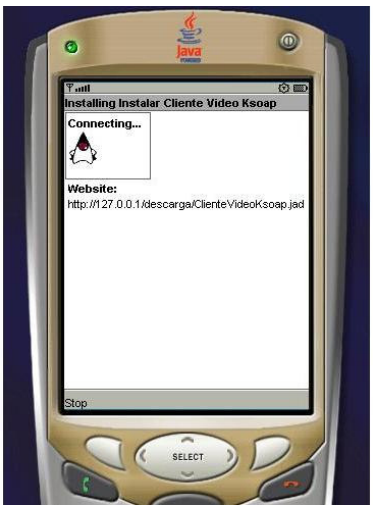
Para simular una descarga vía OTA, utilizamos el emulador *OTA Provisioning* que viene incluido en el paquete *J2ME Wireless Toolkit 2.2.* Al ejecutarlo, se muestran los paquetes ya instalados y la opción de instalar un nuevo paquete. Si seleccionamos esa opción pasamos a una pantalla en la que se nos pide la URL del Midlet. En nuestro caso, la dirección será: *http://127.0.0.1/index.html.* Tras descargarse la página web que creamos anteriormente, se visualiza por pantalla el enlace "Instalar Cliente Video Ksoap". Si pulsamos sobre el botón "Install", comenzará la descarga e instalación del Midlet. En las siguientes capturas de pantalla se muestra el proceso:

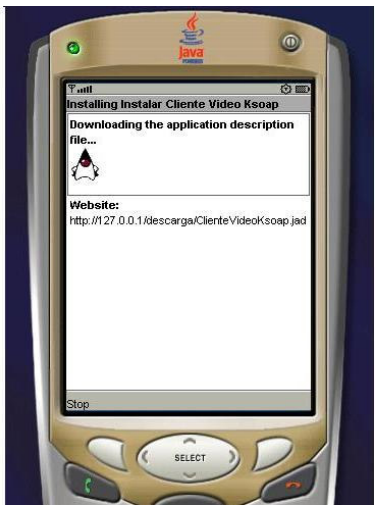

Applications
Install Application
Back
Menu
SELECT


Enter a website to Install From:
http://127.0.0.1/index.html
Back
Menu
SELECT

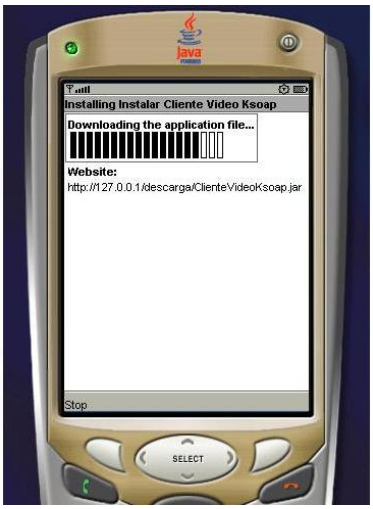

Getting Install List
Downloading...
Website: http://127.0.0.1/index.html
SELECT

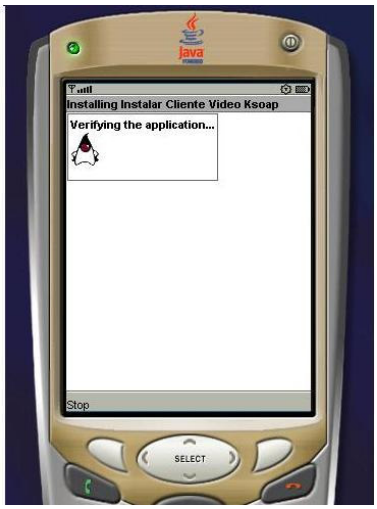

Select one to install:
Instalar Cliente Video Ksoap
Back
Install
SELECT


Installing Instalar Cliente Video Ksoap
Connecting...
Website:
http://127.0.0.1/descarga/ClienteVideoKsoap.jad
Stop
SELECT


Installing Instalar Cliente Video Ksoap
Downloading the application description file...
Website:
http://127.0.0.1/descarga/ClienteVideoKsoap.jad
Stop
SELECT

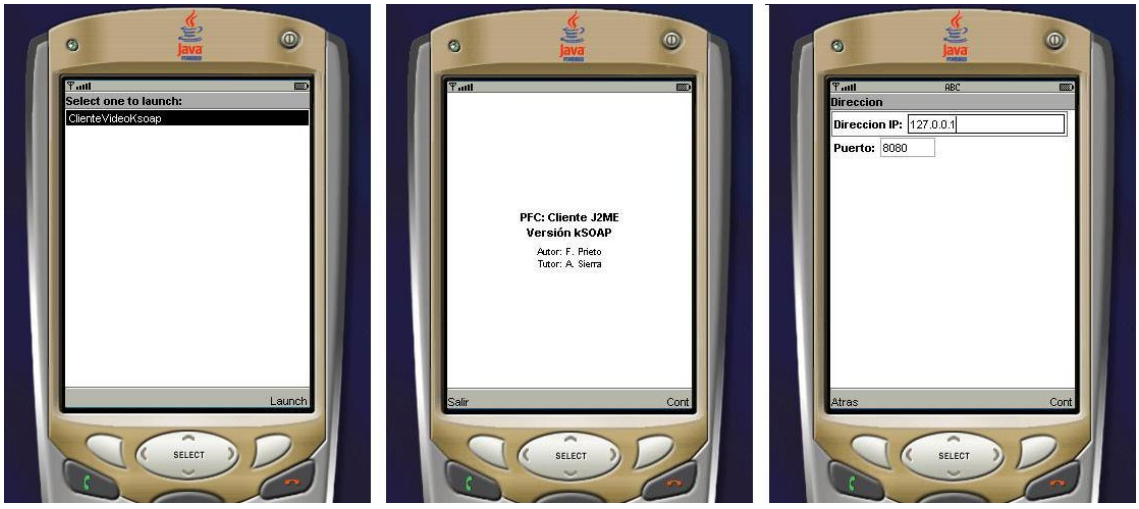

Confirmation
Are you sure you want to install "ClienteVideoKsoap"?
Size: 56 K
Version: 1.0
Vendor: Unknown
Website: ClienteVideoKsoap.jar
Advanced: Untrusted
Cancel
Install
SELECT

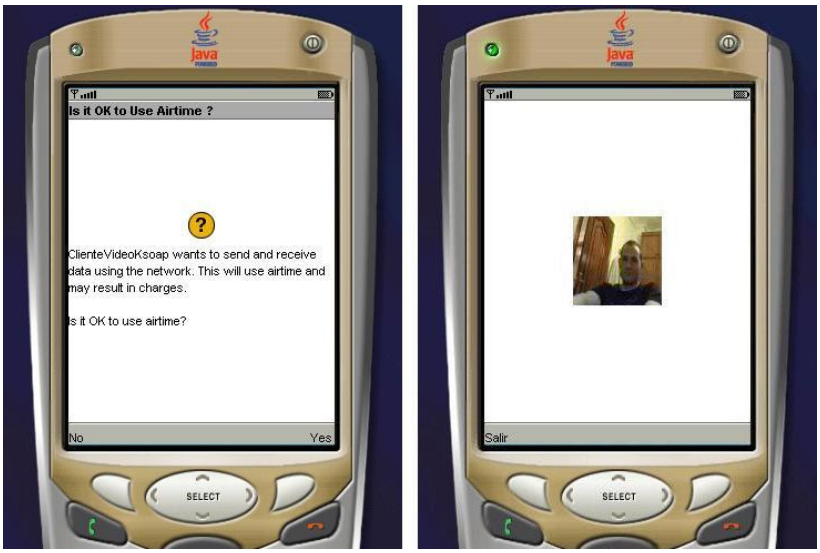

Installing Instalar Cliente Video Ksoap
Downloading the application file...
Website:
http://127.0.0.1/descarga/ClienteVideoKsoap.jar
Stop
SELECT


Installing Instalar Cliente Video Ksoap
Verifying the application...
Stop
SELECT

8.3.4.2 Ejecución del cliente

Por último ejecutamos el cliente. A continuación mostramos una captura de cada una de las pantallas que componen la interfaz de usuario.





8.4 Escenario JSR-172

8.4.1 Desarrollo del servicio web

El servicio web desarrollado está compuesto de una sola clase, llamada *VideoJSR* y cuatro métodos, dos de los cuales se utilizarán para escuchar las peticiones del cliente. Esta clase es muy parecida a la utilizada para el caso Ksoap, aunque tiene algunas variaciones. El método más importante de la clase es el llamado *conectar*, que realiza la

petición al servidor de vídeo especificando el formato y el tamaño de la imagen, y devuelve un array de bytes con la imagen solicitada.

```
public byte[] conectar(String formato, int tam) throws Exception{

        InputStream is = null;
        DataInputStream dis = null;
        int len = 0;
        byte datos[] = null;

        // Hacemos la petición al servidor de vídeo
        URL url = new URL("http", ip, puerto, "/?format=" + formato +
                "?resize=" + tam + "*" + tam + "?" +
getCalidad(formato));
        HttpURLConnection conexion =
(HttpURLConnection)url.openConnection();
        int rc = conexion.getResponseCode();
        if (rc != HttpURLConnection.HTTP_OK) {
            conexion.disconnect();
            throw new Exception("Codigo de respuesta HTTP: " + rc);
        }

        // Comprobamos el tipo de archivo devuelto
        String tipo = conexion.getContentType();
        if (!tipo.equals("image/png") && !tipo.equals("image/jpeg")) {
            throw new Exception("Se esperaba una imagen, " +
                    "pero se ha recibido " + tipo);
        }

        len = conexion.getContentLength();

        // Almacenamos la información recibida en un array de bytes
        if (len > 0) {
            is = conexion.getInputStream();
            dis = new DataInputStream(is);
            datos = new byte [len];
            dis.readFully(datos);
        }

        // Cerramos el flujo de entrada
        if (dis != null) {
            dis.close();
            dis = null;
        }

        // Cerramos la conexión
        if (conexion != null) {
            conexion.disconnect();
            conexion = null;
        }
        return datos;
    }
```

La petición al servidor de vídeo necesita un parámetro que le indique la calidad de la imagen. Este parámetro va a ser fijo para cada uno de los formatos, y se especifica en el método *getCalidad*:

```
public String getCalidad(String formato) {

        String cadena;
        if (formato == "jpg")
            cadena = "quality=50";
        else if (formato == "png")
            cadena = "comp=max";
        else
            cadena = "";

        return cadena;
    }
```

Los dos métodos restantes son los encargados de recibir la petición por parte del cliente. La diferencia entre ambos radica en que la codificación con la cual se enviará la imagen es distinta en cada caso.

En el primer caso vamos a enviar al cliente un array de bytes con la imagen solicitada. En este escenario el módulo Axis en el lado del servidor y las librerías de la especificación JSR-172 en el lado del cliente no codifican ni descodifican automáticamente este array en formato Base64, ya que como veremos más adelante, la codificación de los mensajes intercambiados no es RPC. No obstante podemos obligar al servidor web a codificar el array de bytes en Base64, con lo que devolverá el String resultante.

```
    public String getImagenBase64(String formato, int tam) throws
Exception{

        byte[] datos = conectar(formato, tam);

        return (Base64.encode(datos));
    }
```

En el segundo de los casos enviaremos la imagen en un array de enteros. El fundamento de esta codificación es simple: cada byte que forma la imagen puede ser representado por un número entero de 8 bits. De este modo podemos formar un array de tantos enteros como bytes tenga la imagen. El método que realiza esta conversión es el siguiente:

```
public int[] getImagenIntArray(String formato, int tam) throws
Exception{

        byte datos[] = null;
        int cadena[] = null;
```

```
        int longitud;

        datos = conectar(formato, tam);
        longitud=datos.length;

        // Pasamos cada byte a su valor entero correspondiente
        cadena = new int[longitud];
        for (int i=0; i<longitud; i++)
            cadena[i]=(int)datos[i];

        return (cadena);
    }
```

Con esto ya tenemos preparado el código fuente de nuestro servicio web. El siguiente paso consiste en desplegarlo, es decir, integrarlo en el servidor web Apache Tomcat y hacer públicos los métodos disponibles.

8.4.2 Despliegue del servicio web

Para desplegar el servicio que acabamos de desarrollar debemos tener el servidor de aplicaciones Apache Tomcat correctamente instalado y configurado con el módulo Axis. Los pasos necesarios para llegar a este punto se explican detalladamente en el capítulo "Guía de instalación". El PC en el que se realiza la instalación debe tener acceso a Internet, pues será el encargado de recibir y enviar las peticiones al teléfono móvil. Además es recomendable que Tomcat se ejecute en la misma máquina que el servidor de vídeo por motivos de rendimiento, aunque no es obligatorio. En este proyecto utilizaremos el mismo PC para implementar ambos servidores.

Una vez arrancado el servidor Apache Tomcat accedemos a la carpeta contenedora de servicios web XML, situada en la ruta *C:\Tomcat 4.1\webapps\axis\WEB-INF\classes.* Creamos aquí un directorio llamado *ServVideoJSR* destinado a contener la clase del servicio web, así como la información de despliegue necesaria para la publicación. Dentro de este directorio ubicaremos los siguientes archivos:

- **VideoJSR.java**: Clase que contiene el servicio web detallado anteriormente.

- **deploy.wsdd**: Archivo de despliegue del servicio web. Contiene la información necesaria para que el servidor de aplicaciones Apache Tomcat sepa cómo ejecutar y publicar el servicio web.

En este documento XML se especifica el nombre con el que publicará el servicio (en nuestro caso VideoJSR), el estilo de codificación de la petición (para JSR-172 es

obligatorio usar el estilo literal/wrapped), el espacio de nombres utilizado en los mensajes de petición/respuesta, la clase que implementa el servicio y los métodos públicos que van a estar disponibles para realizar peticiones.

Para este escenario hemos ampliado la información contenida en el documento WSDD. Se han añadido las etiquetas *operation*, que aportan flexibilidad a la hora de definir el servicio web que implementamos. En este caso podemos observar que hemos especificado, para cada método, los tipos de datos que se esperan recibir como parámetros en la petición, así como el tipo de parámetro devuelto en la respuesta. Otra variación con respecto a kSOAP es la utilización del estilo *wrapped*, único soportado por JSR-172.

```
<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="VideoJSR" provider="java:RPC" style="wrapped"
use="literal">
      <parameter name="wsdlTargetNamespace"
value="http://video.samples/"/>
      <parameter name="className" value="ServVideoJSR.VideoJSR"/>

      <operation name="getImagenBase64" qname="operNS:getImagenBase64"
                xmlns:operNS="http://video.samples/"
                returnQName="retNS:getImagenBase64Result"
                xmlns:retNS="http://video.samples/"
                returnType="rtns:string"
                xmlns:rtns="http://www.w3.org/2001/XMLSchema" >
        <parameter qname="pns:formato"
xmlns:pns="http://video.samples/"
                   type="tns:string"
xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
        <parameter qname="pns:tam" xmlns:pns="http://video.samples/"
                   type="tns:int"
xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
      </operation>

      <operation name="getImagenIntArray"
qname="operNS:getImagenIntArray"
                xmlns:operNS="http://video.samples/"
                returnQName="retNS:getImagenIntArrayResult"
                xmlns:retNS="http://video.samples/"
                returnType="rtns:int[]"
                xmlns:rtns="http://www.w3.org/2001/XMLSchema" >
        <parameter qname="pns:formato"
xmlns:pns="http://video.samples/"
                   type="tns:string"
xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
        <parameter qname="pns:tam" xmlns:pns="http://video.samples/"
                   type="tns:int"
xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
      </operation>

      <parameter name="allowedMethods" value="getImagenBase64
getImagenIntArray"/>
```

```
  </service>

</deployment>
```

- **undeploy.wsdd**: Archivo que elimina el despliegue del servicio web. Contiene la información necesaria para que el servidor de aplicaciones deje de publicar el servicio web. Este documento XML simplemente contiene el nombre del servicio web que se quiere dejar de publicar.

```
<undeployment
    xmlns="http://xml.apache.org/axis/wsdd/">
  <service name="VideoJSR"/>
</undeployment>
```

Tras ubicar estos tres archivos en el directorio especificado, compilamos el archivo java:

```
javac VideoJSR.java
```

A continuación realizamos el despliegue del servicio escribiendo en el intérprete de comandos:

```
java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient -
lhttp://localhost:8080/axis/services/AdminService deploy.wsdd
```

Llegados a este punto es recomendable reiniciar el servidor Apache Tomcat para que todos los cambios se apliquen correctamente. Podemos comprobar que el servicio web ya está disponible accediendo a la página web de servicios desplegados que ofrece Axis en la dirección: *http://127.0.0.1:8080/axis/servlet/AxisServlet*

En la figura siguiente se puede apreciar que además de los nombres de los servicios desplegados, aparecen los métodos que cada uno publica, así como un enlace al documento WSDL generado automáticamente por el servidor.

El documento WSDL es el que indica los métodos a través de los cuales se puede invocar el servici web, así como los parámetros que se necesitan para la petición y los que se devuelven como resultado. Este documento se utiliza sobre todo en programas que generan automáticamente un cliente de servicios web a partir del WSDL dado.

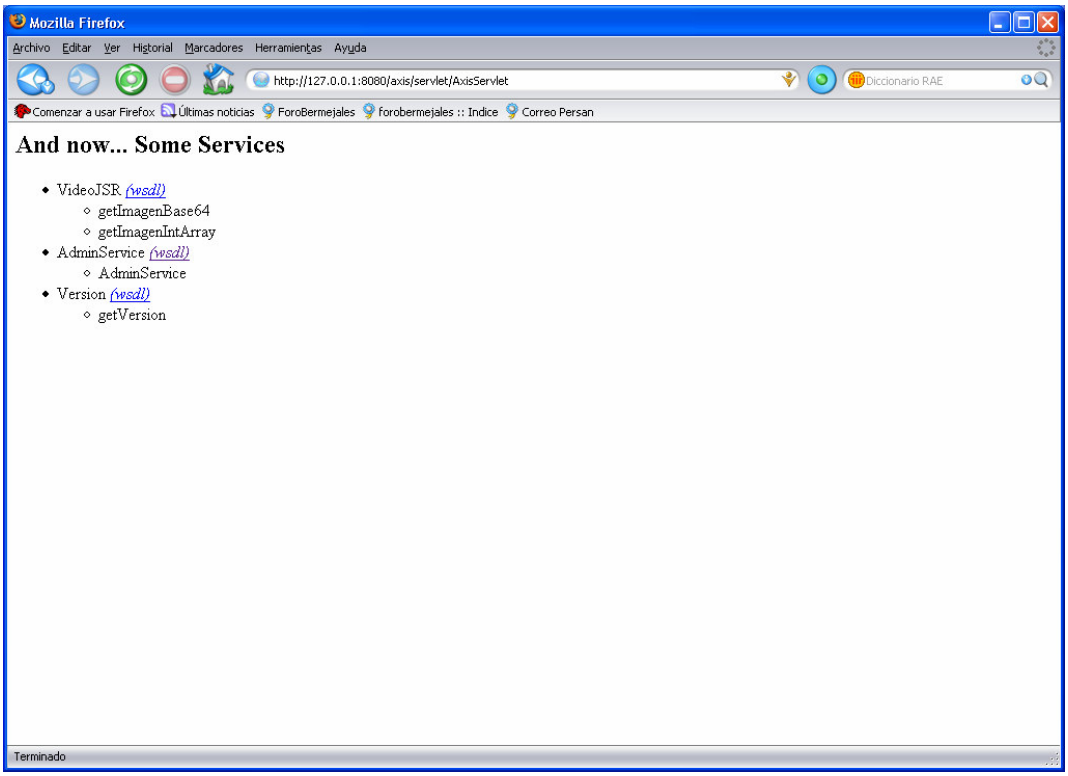


Figura 8.4: Servicios web desplegados en Axis

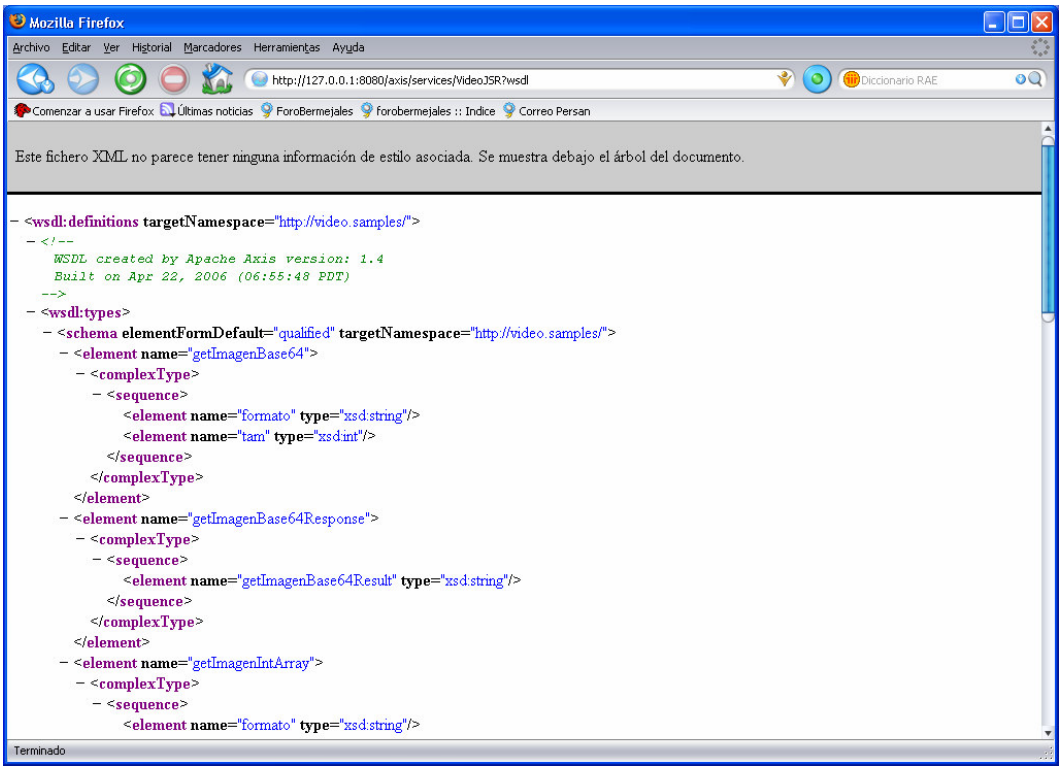


Figura 8.5: Parte del documento WSDL generado automáticamente por Axis

8.4.3 Desarrollo del cliente

El cliente de nuestro servicio web XML será un programa Java especialmente diseñado para dispositivos móviles utilizando J2ME. Su objetivo es el de realizar peticiones al servidor web XML a través de Internet mediante una conexión GPRS. Tras realizar la petición, el cliente queda a la espera de recibir una imagen como respuesta, la cual imprimirá por pantalla inmediatamente después de descodificarla. Mientras no se indique lo contrario, se realizarán peticiones indefinidamente con el fin de recibir una secuencia continua de imágenes, de modo que la representación por pantalla de dicha secuencia ofrezca al usuario una sensación de movimiento.

El cliente está formado por seis clases que conforman el programa principal y seis clases más que forman parte del stub, ofreciendo el soporte de Servicios Web XML y abstrayéndonos de la construcción de mensajes SOAP y de la representación de la información en XML.

Detallamos a continuación cada una de las clases utilizadas:

- **Bienvenida.java**: Contiene una clase que presenta por pantalla un mensaje de bienvenida. Esta pantalla se visualizará nada más iniciarse el programa. El método constructor inicia las variables:

```
     public Bienvenida() {

       // Tipo de fuente utilizado
       fuente = null;

       // Texto de presentacion
       cadena1 = "PFC: Cliente J2ME";
       cadena2 = "Versión JSR-172";
       cadena3 = "Autor: F. Prieto";
       cadena4 = "Tutor: A. Sierra";


       // Altura y anchura de la pantalla del dispositivo
       CanvasWidth = getWidth();
       CanvasHeight = getHeight();
   }
```

Por su parte, el método *paint* representa por pantalla las cadenas de caracteres definidas anteriormente. Como se trata de una pantalla Canvas, es necesario especificar el punto exacto donde se deben ubicar cada una de las líneas de texto:

```
public void paint(Graphics g) {

       // Tipo de letra negrita y tamaño medio
```

```
        fuente = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,
Font.SIZE_MEDIUM);
        g.setFont(fuente);

        // Borramos la pantalla pintandola de blanco
        g.setColor(0xffffff);
        g.fillRect(0, 0, CanvasWidth, CanvasHeight);

        // Escribimos las lineas de texto en negro centradas en la
pantalla
        g.setColor(0);
        int x = CanvasWidth / 2;
        int y = CanvasHeight / 2 - 2*fuente.getHeight();
        g.drawString(cadena1, x, y, Graphics.TOP|Graphics.HCENTER);
        g.drawString(cadena2, x, y+fuente.getHeight(),
                Graphics.TOP|Graphics.HCENTER);

        // Tipo de letra pequeña
        fuente = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,
Font.SIZE_SMALL);
        g.setFont(fuente);

        g.drawString(cadena3, x, y+(3*fuente.getHeight()),
                Graphics.TOP|Graphics.HCENTER);
        g.drawString(cadena4, x, y+(4*fuente.getHeight()),
                Graphics.TOP|Graphics.HCENTER);
}
```

- **ClienteVideo.java**: Ésta es la clase principal del Midlet, ya que implementa los métodos *startApp*, *pauseApp* y *desproyApp*. En esta clase definimos todas las pantallas tipo Screen, destinadas a recopilar la información de usuario.

El método startApp es el que se encarga de inicializar los comandos (botones) y variables, y representar la primera pantalla de usuario, en este caso la pantalla de bienvenida:

```
public void startApp() {

        /* Inicializamos los comandos */
        okToListCodificacion = new Command("Cont", Command.OK, 0);
        okToImagen = new Command("Cont", Command.OK, 1);
        backToBienvenida = new Command("Atras", Command.BACK, 1);
        backToListCodificacion = new Command("Atras", Command.BACK,
1);
        backToListFormato = new Command("Atras", Command.BACK, 1);
        exitCommand = new Command("Salir", Command.EXIT, 1);

        /* Inicializamos las variables */
        fin=false;
        pantallaImagen=null;

        /* Presentamos la pantalla de bienvenida */
        display = Display.getDisplay(this);
```

```
        display.setCurrent(get_bienvenida());
    }
```

Los siguientes métodos inicializan cada una de las pantallas en las que se pedirá información de usuario, como por ejemplo el formato de la imagen o la codificación a utilizar. En este escenario el usuario no puede elegir la dirección ni el puerto del servidor, ya que el stub contiene esta información y no puede ser modificada en tiempo de ejecución. La información se recopila mediante listas, formularios y cuadros de texto:

```
/* Pantalla en la que se elige la codificacion de la imagen.
     * Se puede elegien entre Base64 y array de enteros */
    public List get_listCodificacion() {
        if (listCodificacion == null) {
            listCodificacion = new List("Codificacion",
Choice.IMPLICIT,
                    new String[] { "Base64", "IntArray"}, null);
            listCodificacion.addCommand(backToBienvenida);
            listCodificacion.setCommandListener(this);
        }
        return listCodificacion;
    }

    /* Pantalla en la que se elige el formato de la imagen (png o jpg)
*/
    public List get_listFormato() {
        if (listFormato == null) {
            listFormato = new List("Formato de imagen",
Choice.IMPLICIT,
                    new String[] { "PNG", "JPG" }, null);
            listFormato.addCommand(backToListCodificacion);
            listFormato.setCommandListener(this);
        }
        return listFormato;
    }

    /* Pantalla de resumen con todas las opciones elegidas */
    public Form get_formResumen() {
        stringResumen = new StringItem("",
                "Codif: " + codificacion + "\n" +
                "Formato: " + formato);
        formResumen = new Form("Preferencias");
        formResumen.append(stringResumen);
        formResumen.addCommand(okToImagen);
        formResumen.addCommand(backToListFormato);
        formResumen.setCommandListener(this);
        return formResumen;
    }
```

A continuación inicializamos las pantallas tipo Canvas, añadiéndole los botones o comandos:

```
/* Pantalla de bienvenida */
    public Bienvenida get_bienvenida() {
        if (pantallaBienvenida == null) {
            pantallaBienvenida = new Bienvenida();
            pantallaBienvenida.addCommand(okToListCodificacion);
            pantallaBienvenida.addCommand(exitCommand);
            pantallaBienvenida.setCommandListener(this);
        }
        return pantallaBienvenida;
    }

    /* Pantalla que presentará las imágenes */
    public Imagen get_Imagen() {
        if (pantallaImagen == null) {
            pantallaImagen = new Imagen();
            pantallaImagen.addCommand(exitCommand);
            pantallaImagen.setCommandListener(this);
        }
        return pantallaImagen;
    }
```

Las acciones que se realizarán al ejecutar un comando deben definirse dentro del método *CommandAction*:

```
/* Acciones que se ejecutaran al activar un comando */
    public void commandAction(Command command, Displayable
displayable) {

        if (command == okToListCodificacion) {
            display.setCurrent(get_listCodificacion());
        }

        else if (command == okToImagen) {
            display.setCurrent(get_Imagen());
            (new Bucle()).start();
        }

        else if (command == backToBienvenida) {
            display.setCurrent(get_bienvenida());
        }

        else if (command == backToListCodificacion) {
            display.setCurrent(get_listCodificacion());
        }

        else if (command == backToListFormato) {
            display.setCurrent(get_listFormato());
        }

        else if (displayable == listCodificacion && command ==
listCodificacion.SELECT_COMMAND) {
            switch (get_listCodificacion().getSelectedIndex()) {
                case 0:
                    codificacion = "Base64";
                    display.setCurrent(get_listFormato());
```

```
                    break;
                case 1:
                    codificacion = "IntArray";
                    display.setCurrent(get_listFormato());
                    break;
            }
        }

        else if (displayable == listFormato && command ==
listFormato.SELECT_COMMAND) {
            switch (get_listFormato().getSelectedIndex()) {
                case 0:
                    formato = "png";
                    display.setCurrent(get_formResumen());
                    break;
                case 1:
                    formato = "jpg";
                    display.setCurrent(get_formResumen());
                    break;
            }
        }

        else if (command == exitCommand) {
            exitMIDlet();
        }
    }
```

Por último indicamos las acciones necesarias a realizar al salir del Midlet, tales como liberación de memoria:

```
/* Acciones que se ejecutarán al terminar la aplicación */
    public void exitMIDlet() {
        display.setCurrent(null);
        fin = true;
        destroyApp(true);
        notifyDestroyed();
    }

    public void pauseApp() {
    }

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

- **Bucle.java**: Tras la última pantalla de petición de datos, el Midlet ejecuta un bucle de peticiones al servidor web XML y tras recoger la imagen devuelta, utiliza la clase *Imagen* para representarla por pantalla. El métdo que realiza todo este proceso es el método *run*:

```
public void run() {
        try{
            do {
```

```
                // Dormimos entre peticiones
                do
                    sleep(intervalo);
                while(Imagen.cerrojo || JSR172.capturando);

                // Petición al servicio web
                JSR172 peticion = new JSR172();
                Imagen.datos = peticion.CapturaImagen();

                // Presentación por pantalla
                if (Imagen.datos != null)
                    ClienteVideo.pantallaImagen.repaint();
                else
                    ClienteVideo.fin=true;

            } while (ClienteVideo.fin == false);
        } catch (InterruptedException e) {
            Alert a = new Alert("Error ", e.toString(), null, null);
            a.setTimeout(Alert.FOREVER);
            ClienteVideo.display.setCurrent(a);
        }
    }
```

- **JSR172.java**: Esta clase realiza la petición al servidor web con los parámetros que se le han pedido al usuario en las pantallas anteriores. Gracias al stub, realizar una petición a un servicio web remoto es exactamente igual que invocar a un método local.

Tras recibir la imagen, distinguimos el tipo de codificación: si ésta es Base64, descodificamos el String recibido en el array de bytes original ayudándonos de una librería externa. En caso contrario debemos pasar el array de enteros a un array de bytes. El método *CapturaImagen* es el que realiza todas estas operaciones:

```
byte[] CapturaImagen() {

        Image foto = null;
        DataInputStream dis = null;
        byte data[];
        try {

            capturando = true;
            Runtime.getRuntime().gc();

            // Peticion al servidor web

            // Si la codificación es Base64, pasamos el String a array
de bytes
            if (ClienteVideo.codificacion == "Base64"){

                /* Hacemos la petición al servidor web invocando
                 * el método correspondiente del stub */
```

```
                String cadena =
proxy_VideoJSR_Stub.getImagenBase64(ClienteVideo.formato,ClienteVideo.
TAM);

                /* JSR-172 no codifica los arrays de bytes a Base64,
                 * de modo que tenemos que forzar la decodificación */
                data = Base64.decode(cadena);
            }

            // Si la codificación es IntArray, pasamos los enteros a
bytes
            else {

                /* Hacemos la petición al servidor web invocando
                 * el método correspondiente del stub */
                int cadena[] =
proxy_VideoJSR_Stub.getImagenIntArray(ClienteVideo.formato,ClienteVide
o.TAM);
                int numBytes = cadena.length;

                //Transformamos los enteros a bytes y los almacenamos
en un array
                data = new byte[numBytes];
                for (int j=0;j<numBytes;j++)
                    data [j] = (byte)cadena[j];
            }

            capturando = false;

            return data;

        }catch(Exception exception) {
            Alert a = new Alert("Error E/S", exception.toString(),
null, null);
            a.setTimeout(Alert.FOREVER);
            ClienteVideo.display.setCurrent(a);
            return null;
        }
    }
```

- **Imagen.java**: Clase que representa por pantalla la imagen recibida del servidor web. Si el formato es JPG, se descodifica utilizando la clase DecodificadorJPEG.

El método constructor inicializa las variables relacionadas con el tamaño de la pantalla, así como el descodificador JPG en caso de que la imagen recibida tenga ese formato:

```
public Imagen() {
        datos = null;

        // Calculamos el centro de la pantalla
        CanvasWidth = getWidth();
        CanvasHeight = getHeight();
```

```
        centro_x = (CanvasWidth - ClienteVideo.TAM) / 2;
        centro_y = (CanvasHeight - ClienteVideo.TAM) / 2;
        if(centro_x < 0)
            centro_x = 0;
        if(centro_y < 0)
            centro_y = 0;

        // Iniciamos la clase DecodificadorJPEG
        iniciaJPEG();
    }

    void iniciaJPEG(){
        if (ClienteVideo.formato == "jpg") {
            // Inicializo el decodificador JPEG
            decod = new DecodificadorJPEG();
            decod.setAncho(ClienteVideo.TAM);
            decod.setAlto(ClienteVideo.TAM);
        }
    }
```

El método *paint* representa la imagen por pantalla. Mientras se efectúa la representación se activa un cerrojo para que el bucle no efectúe peticiones a la vez, ya que el rendimiento de la aplicación disminuye considerablemente. Si el formato es PNG se llama al método *createImage* incluído en J2ME que se encarga de descodificar dicho formato. En caso contrario, si la imagen es de tipo JPG, utilizaremos la clase *DecodificadorJPEG.*

```
public void paint(Graphics g) {
        cerrojo = true;

        // Borramos la pantalla pintandola de blanco
        g.setColor(0xffffff);
        g.fillRect(0, 0, CanvasWidth, CanvasHeight);

        // Presenta la imagen por pantalla
        if(datos != null){
            int numBytes = datos.length;

            // Si el formato es png, representamos directamente
            if (ClienteVideo.formato != "jpg"){
                foto = Image.createImage(datos,0,numBytes);
            }

            // Si el formato es jpg, utilizamos el decodificador
            else {

                // Crea el buffer ARGB para la imagen decodificada
                imagenARGB = new
int[ClienteVideo.TAM*ClienteVideo.TAM];

                // Decodifica en imagenARGB[]
                decod.decodifica(datos,imagenARGB, numBytes);

                // Creamos la imagen
```

```
                foto =
Image.createRGBImage(imagenARGB,ClienteVideo.TAM,
                        ClienteVideo.TAM,false);

            }
            g.drawImage(foto, centro_x, centro_y, 0);
        }
        cerrojo = false;
    }
```

- **DecodificadorJPEG.java**: Clase que toma un array de bytes con una imagen JPG y la descodifica para poder representarla en pantalla.

Esta clase fue desarrollada en un proyecto final de carrera anterior y no vamos a realizar su análisis detallado debido a su extensión y complejidad. El código fuente de esta clase puede consultarse en el capítulo "Planos de código".

- **Stub**: El stub es un conjunto de clases generadas automáticamente a partir del documento WSDL proporcionado por el servidor web, y cuyo objetivo es el de abstraer al programador del cliente tanto de la conexión con el servidor web como de la comunicación entre éste y el cliente.

La generación del stub es muy sencilla. Utilizaremos la utilidad "Stub Generator" incluida en el J2ME Wireless Toolkit 2.2 de Sun. Una vez ejecutada se nos piden 4 parámetros:

- **URL del documento WSDL**

  En nuestro caso, http://127.0.0.1:8080/axis/services/VideoJSR?wsdl

- **Ruta de salida**

  Aquí especificaremos C:\WTK22\apps\ClienteVideoJSR\src\cliente

- **Paquete de salida**

  Indicaremos "Stub"

- **Configuración**

Elegimos “CLDC 1.1”

Una vez hecho esto, se creará un paquete en la ruta especificada que contiene las seis clases que conforman el stub.

8.4.4 Simulación del cliente

Para simular el comportamiento de un teléfono móvil real, utilizamos el emulador J2ME Wireless Toolkit 2.2. Con él podemos probar el funcionamiento del cliente, así como realizar medidas de parámetros tales como tiempos de respuesta y consumo de memoria. La instalación y configuración de este programa se detalla en el capítulo “Guía de instalación”.

Para simular el programa que hemos desarrollado, ejecutamos el J2ME Wireless Toolkit y creamos un nuevo proyecto, al que llamaremos *ClienteVideoJSR*. En la siguiente pantalla de configuración tenemos que indicar que el perfil debe ser MIDP 2.0 y la configuración CDLC 1.1. Asimismo es necesario especificar el nombre de la clase principal del Midlet, en nuestro caso *cliente.ClienteVideo*, e indicar que se añadan las librerías JSR-172. Automáticamente se habrá creado un directorio con el mismo nombre del proyecto en la ruta *C:\WTK22\apps*. Dentro de la carpeta *src* creamos una llamada *cliente*, que corresponde con el paquete en el que contendremos el código fuente. Es en esta carpeta donde ubicaremos los archivos java con las clases especificadas anteriormente.

A continuación situamos el paquete *ksoap2-j2me-core-2.1.1.jar* en el directorio *lib*. Aunque para JSR-172 no es necesario, este paquete nos permite descodificar un String Base64 en un array de bytes. Puede descargarse desde el siguiente enlace web: http://sourceforge.net/projects/ksoap2/.

Una vez hecho esto, compilamos el proyecto con la opción *Build* y creamos el paquete jar precompilado accediendo al menú *Project→ Package→ Create Package.* Esta opción crea tres archivos en el directorio *C:\WTK22\apps\ClienteVideoJSR\bin*:

- **ClienteVideoJSR.jar**: Fichero que contiene todas las clases necesarias para la ejecución del cliente.

- **ClienteVideoJSR.jad**: Fichero descriptor de aplicaciones Java. Se encarga de verificar que su MIDlet Suite asociado se ajusta convenientemente al dispositivo sobre el que será descargado.

- **MANIFEST.MF**: Contiene información sobre el cliente, como por ejemplo el nombre y tamaño de la clase principal, la versión de MIDP y CLCD, nombre del fabricante, etc.

Estos tres ficheros son los que se deben descargar a un teléfono móvil real para ejecutar la aplicación.

8.4.4.1 Descarga OTA

Para la instalación del cliente en un terminal móvil vamos a utilizar el método de descarga vía OTA, aprovechando el hecho de que el servidor de vídeo es también servidor de ficheros y viene además configurado para soportar este tipo de descargas. Para alojar nuestro Midlet-Suite en el servidor es necesario realizar los siguientes pasos:

1. Crear una carpeta dentro del directorio de trabajo del servidor de vídeo y alojar allí los tres archivos que hemos obtenido. En nuestro caso, el directorio de trabajo es *C:*, así que creamos la carpeta *C:/descarga* y copiamos los ficheros mencionados.

2. Editar el archivo *ClienteVideoJSR.jad*, para que la línea que contiene información sobre la URL muestre lo siguiente:

```
MIDlet-Jar-URL: http://127.0.0.1/descarga/ClienteVideoJSR.jar
```

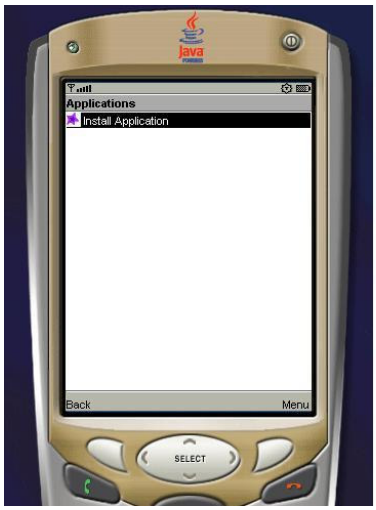
3. Crear una sencilla página web con el enlace al archivo *jad.* El archivo debe situarse en el directorio de trabajo del servidor de vídeo, y el contenido se muestra a continuación:

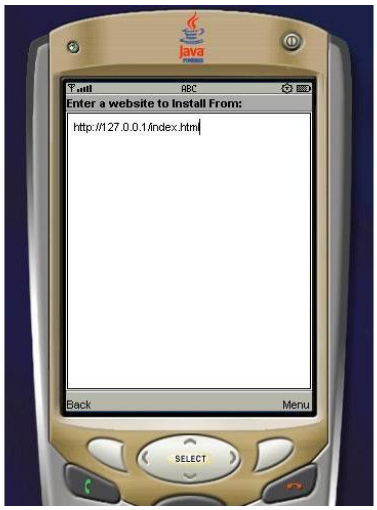
```
<html>

<head>
<title>Cliente Video - Descarga</title>
</head>

<body>
<a href="http://127.0.0.1/descarga/ClienteVideoJSR.jad">Instalar
Cliente Video JSR</a>
</body>

</html>
```

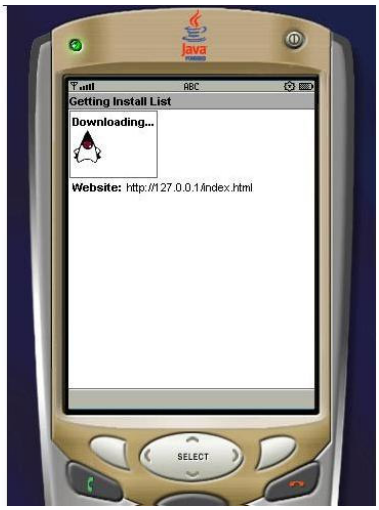
4. Ejecutar el servidor de vídeo, ya que hasta que el servidor no comience a capturar imágenes, los archivos no van a estar disponibles para su descarga.

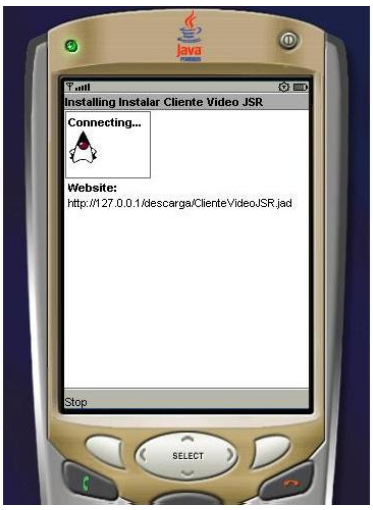
Para simular una descarga vía OTA, utilizamos el emulador *OTA Provisioning* que viene incluido en el paquete *J2ME Wireless Toolkit 2.2.* Al ejecutarlo, se muestran los paquetes ya instalados y la opción de instalar un nuevo paquete. Si seleccionamos esa opción pasamos a una pantalla en la que se nos pide la URL del Midlet. En nuestro caso, la dirección será: *http://127.0.0.1/index.html.* Tras descargarse la página web que creamos anteriormente, se visualiza por pantalla el enlace "Instalar Cliente Video JSR". Si pulsamos sobre el botón "Install", comenzará la descarga e instalación del Midlet.

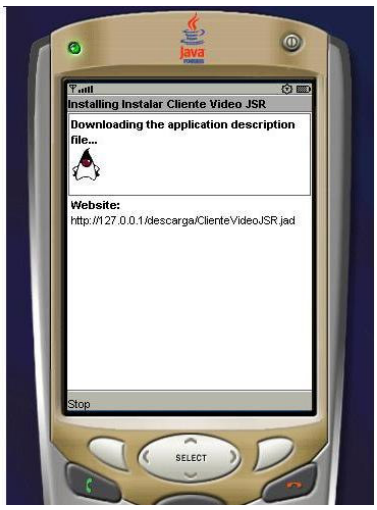
En las siguientes capturas de pantalla se muestra el proceso:

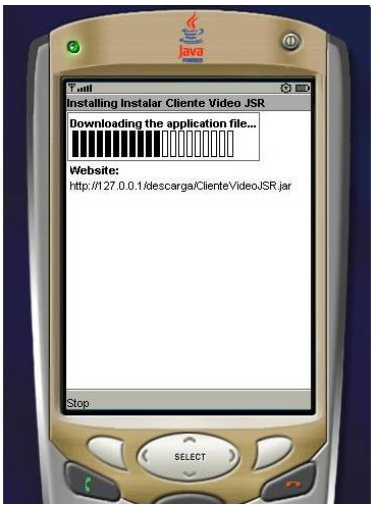


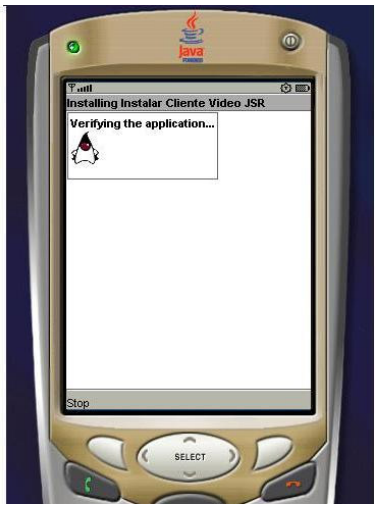






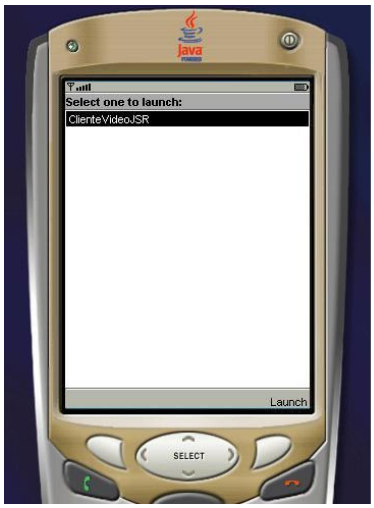


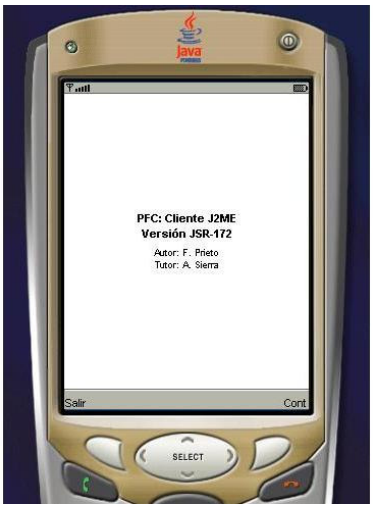




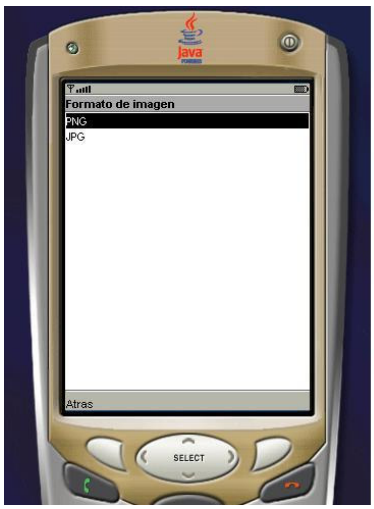
8.4.4.2 Ejecución del cliente

Por último ejecutamos el cliente. A continuación mostramos una captura de cada una de las pantallas que componen la interfaz de usuario. Se puede apreciar la ausencia de la pantalla que pedía en kSOAP la dirección y el puerto del servidor.

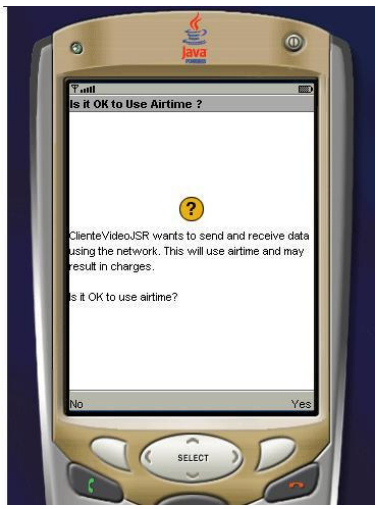

Select one to launch:
ClienteVideoJSR
Launch
SELECT


PFC: Cliente J2ME
Versión JSR-172
Autor: F. Prieto
Tutor: A. Sierra
Salir
Cont
SELECT


Codificacion
Base64
IntArray
Atras
SELECT


Formato de imagen
PNG
JPG
Atras
SELECT


Preferencias
Codif: IntArray
Formato: png
Atras
Cont
SELECT


Is it OK to Use Airtime ?
ClienteVideoJSR wants to send and receive data using the network. This will use airtime and may result in charges.
Is it OK to use airtime?
No
Yes
SELECT


Salir
SELECT

8.5 Consideraciones finales

En este capítulo hemos desarrollado un escenario que nos permite simular la utilización de un teléfono móvil para recibir imágenes de vídeo a través de servicios web.

Una vez que hemos probado que el software funciona adecuadamente y que la comunicación entre el cliente, el servidor web XML y el servidor de vídeo es correcta, podemos pasar a realizar la batería de pruebas que nos aportarán la información necesaria para valorar el rendimiento que ofrece esta implementación.