9. APLICACIÓN DESARROLLADA II

9.1 Introducción

En el capítulo que se desarrollará a continuación, se pretende explicar exhaustivamente el desarrollo de la aplicación y las pruebas realizadas. En general, se pretenden los siguientes objetivos:

- Firma electrónica de documentos contenidos en un servidor WebDAV, ya sean facturas electrónicas, albaranes, etc.
- Cada usuario posee una cuenta propia, con nombre de usuario y contraseña, para acceder al servidor WebDAV donde se encuentran los documentos.
- La firma electrónica se realizará en el cliente móvil, mediante invocación de servicios web a un servidor Web XML que se encargará de gestionar las conexiones entre los distintos servidores.
- El dispositivo móvil debe firmar el hash del documento que seleccione el cliente. Dicho hash será calculado por el servidor @firma.
- El documento seleccionado, a su vez, debe ser registrado en el servidor de @firma.
- Una vez realizada la firma, ésta deberá ser devuelta al servidor WebDAV, donde deberá aparecer junto con el fichero original.

A continuación estudiaremos el escenario de prueba realizado.

9.2 Escenario de prueba

Para cumplir los objetivos anteriores, el escenario de pruebas creado realizará las siguientes acciones:

1. Simulación de un cliente móvil con tecnología J2ME, de forma que realice las llamadas a los servicios web correspondientes.
2. Despliegue de los servicios web que serán consumidos por el cliente.

Las pruebas se realizarán para la interfaz JSR-172, que será la tecnología J2ME de servicios web empleada.

El entorno en el que se desarrollarán las pruebas está compuesto por un ordenador personal, un servidor WebDAV instalado en una máquina exterior y un servidor @firma que también está instalado exteriormente. En el ordenador personal se ejecutará el simulador del dispositivo móvil J2ME y el servidor web Apache Tomcat en el que se desplegarán los servicios web. Como ya se ha comentado en un capítulo anterior, el servidor Web XML se comunica con el servidor WebDAV

mediante HTTP, mientras que con el servidor de @firma se comunica mediante HTTPS.

El emulador del cliente móvil se comunica con el servidor Web mediante SOAP sobre HTTP. El escenario de pruebas es, por tanto:

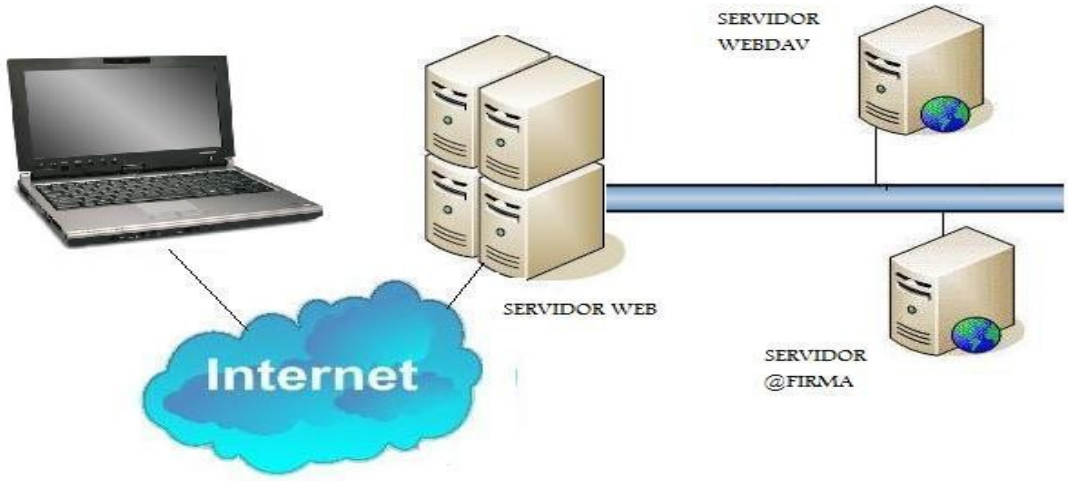


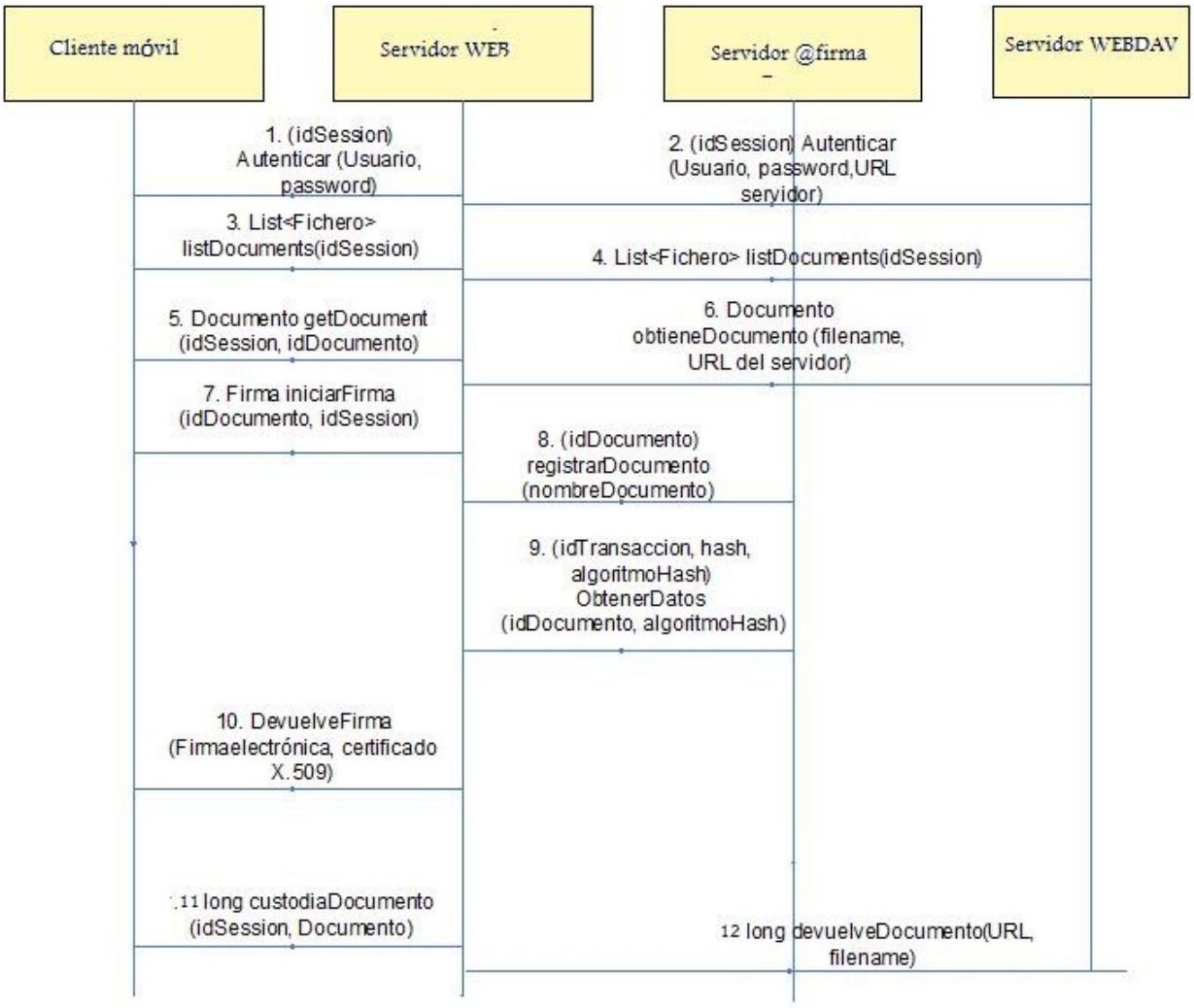
Figura 9.1. Escenario de pruebas

9.3 Desarrollo de los servicios web

9.3.1 Introducción

Para poder cumplir las expectativas del cliente móvil, ha sido necesario desarrollar cinco servicios web, con el fin de liberar al máximo las tareas a realizar por el cliente, principalmente por los problemas de cómputo de memoria existentes en dispositivos móviles.

El esquema siguiente muestra el intercambio de datos necesarios entre cliente y servidores , como paso previo al desarrollo de los servicios web realizados:



1. La autenticación devolvería el id de sesión, que contendrá el valor -1 si no se ha podido realizar la autenticación con el servidor WebDAV.

2. En la autenticación entre la aplicación y el servidor WebDAV será necesario el username y password.

3. El servicio web devuelve la lista de ficheros asociados al usuario.

4. Es necesario relacionar el identificador de sesión con el usuario en concreto.

5. Descarga el documento que se identificará por el identificador de sesión y el identificador del documento.

6. Obtiene del servidor webdav el documento en cuestión. El nombre del documento se obtendría a partir del idDocumento del servicio web anterior.

7. Invoca el servicio web necesario para registrar el documento sin firmar en @firma. Hay que indicar el identificador de documento y el identificador de sesión. Se devuelve un objeto que lo he llamado *Firma* que contendrá el hash en forma de String que representa al documento escogido.

8. La aplicación envía la petición al servicio web registrarDocumento, con el fin de registrar el documento a firmar. Devuelve el identificador del documento tras su registro.

9. Se invoca al servicio web obtenerDatos con el identificador del documento que devolvió el servicio web anterior y el algoritmo hash que se desea usar en la firma electrónica. El servicio Web devuelve tanto el hash del documento indicado como el algoritmo hash empleado. Además, se obtiene el identificador de la transacción que se usará en una fase posterior.

10. El cliente de firma debe devolver a la aplicación la información generada , es decir, la Firma Electrónica en forma de String.

11. El cliente de firma invoca el servicio necesario para custodiar el documento ya firmado en el servidor webdav.

12. Servicio web que guarda el documento firmado en el servidor webdav.

Tras este esquema que explica la interconexión cliente-servidor WEB, pasaremos a comentar cada servicio web desarrollado de forma única en los siguientes apartados.

9.3.2 Desarrollo de los métodos de los servicios web

9.3.2.1 Servicio web Autenticar

Este servicio web permite realizar los puntos 1 y 2 del esquema anterior. Para poder acceder al servidor WebDAV, cada usuario tiene un nombre de usuario y contraseña propio.

El nombre de usuario se asigna del siguiente modo: usuario@empresa de forma que “empresa” identifica el nombre de la empresa a la que pertenece el usuario. Esto se hace de esta forma para que cada usuario puede acceder única y exclusivamente a la consulta de sus propias facturas y documentos.

El servicio web autenticar, permite conectar con el servidor WebDAV e iniciar una nueva sesión. Puesto que cada sesión es única dependiendo del usuario, es necesario que el cliente pase como parámetro al servicio web los siguientes:

- Nombre de usuario
- Contraseña

Una vez que se tienen estos dos parámetros, se construye dinámicamente la URL de acceso, que es de la forma:

http://prometeo.utrera.org/empresa/dms/ktwebdav/ktwebdav.php/

En caso de que el inicio de sesión haya sido realizado sin ningún problema, se devolverá un identificador de la sesión, que será de la forma *usuario+contraseña* codificado en Base64. Esto se

hace de esta forma para posteriormente poder obtener los documentos asociados a ese usuario a través de su nombre de usuario y contraseña. Puesto que el identificador de sesión contiene el nombre de usuario y contraseña, es posible tener siempre acceso al servidor para obtener los documentos.

En caso de error en el inicio de sesión, el servicio web devolverá el valor -1. El método que permite implementar el servicio web es el siguiente:

```
public String login (String username, String password) throws Exception {

            ResourceBundle prop =
ResourceBundle.getBundle("resources.conexion");

            String host = prop.getString("conexion.host");
            String puerto=prop.getString("conexion.port");

            int port = Integer.parseInt(puerto);
            UtilesUrl utiles= new UtilesUrl();
            String uri=utiles.obtieneUrlServidor(username);
            String idsesion=username+"+"+password;
            byte[] data =idsesion.getBytes();

            Base64Coder code = new Base64Coder();
            byte[] codificado =code.encodeBase64(data);

            String texto_codificado = new String(codificado,"UTF8");

            DAVConnection dc=DAVConnectionPool.getDAVConnection(host, port,
false);

            dc.setUsername(username);
            AuthorizationInfo ai= new AuthorizationInfo(host,
                        port,
                        "Basic",
                        "KTWebDav Server",
                        Codecs.base64Encode(username + ":" + password));
            dc.setAuthorization(ai.toString());
            PropFindMethod method;

            method=new PropFindMethod(uri);

            dc.execute(method);

            int status=method.getStatus();
            if(status >= 200 && status < 300)
                  texto_codificado=texto_codificado;
            else
                  texto_codificado=String.valueOf(-1);


            return texto_codificado;
      }
```

El método obtieneUrlServidor permite obtener dinámicamente la url de acceso correspondiente al usuario concreto, tal y como se comentó anteriormente. Para establecer la conexión, se ha utilizado el método DAV *PropFindMethod*, que también se comentó en el capítulo 6 de esta memoria.

9.3.2.2 Servicio web Lista de Documentos

El siguiente servicio web contiene un método que permite obtener la lista de documentos asociada a un usuario concreto que se encuentran en el servidor WebDAV. Más concretamente, devuelve una lista de FicheroBean, que no es más que una estructura bean que contiene tres campos: el nombre del documento, la fecha de la última modificación, el identificador de documento y el contenido.

El identificador de documento se corresponde con la ruta Url del documento codificada en Base64.

El parámetro necesario para el método es el siguiente:

- Identificador de sesión

Tal y como se comentó anteriormente, mediante el identificador de sesión es posible obtener el nombre de usuario y contraseña para acceder al servidor WebDAV.

La función sólo devuelve los documentos que se encuentran en el servidor en formato .pdf, puesto que son los únicos que se pueden firmar. Además, el método es recursivo puesto que hay que recuperar los documentos de todas las carpetas y subcarpetas existentes en el servidor.

El código Java que implementa dicho método es el siguiente:

```
     public FicheroBean[] listDocument(String idSesion) throws Exception{

            ResourceBundle prop =
ResourceBundle.getBundle("resources.conexion");

            String host = prop.getString("conexion.host");
            String puerto=prop.getString("conexion.port");

            //Primero: se desencripta el idsession para obtener el nombre de
usuario
            Base64Coder code = new Base64Coder();

            byte[] codificado =code.decodeBase64(idSesion.getBytes());



            String listadoDocumentos=prop.getString("listaDocumentos.tamanio");
            int port = Integer.parseInt(puerto);
            int tamlist=Integer.parseInt(listadoDocumentos);
            FicheroBean [] tablaficheros = new FicheroBean[tamlist];

            String texto_decodificado = new String(codificado,"UTF8");
            System.out.println("Texto decodificado:"+texto_decodificado);
            UtilesUrl utiles= new UtilesUrl();
            String[] datos=utiles.obtieneUsuarioycontrasena(texto_decodificado);
//          Segundo:se vuelve a construir la URL a partir del nombre de usuario
desencriptado
            String path = utiles.obtieneUrlServidor(datos[0]);
```

```
            String [] nombreFicheros = new String [tamlist];
            String [] nombreFicherosDirectorios=new String [tamlist];



            DAVConnection dc=DAVConnectionPool.getDAVConnection(host, port,
false);

            dc.setUsername(datos[0]);
            AuthorizationInfo ai= new AuthorizationInfo(host,
                        port,
                        "Basic",
                        "KTWebDav Server",
                        Codecs.base64Encode(datos[0] + ":" + datos[1]));
            dc.setAuthorization(ai.toString());
            PropFindMethod method;

            method=new PropFindMethod(path);

            dc.execute(method);

            DAVFile file =method.getDAVFile();
            int i=0;
            Iterator<DAVFile> it = file.children();
            while(it.hasNext()){
                  DAVFile nombre = it.next();
                  nombreFicheros[i]=nombre.getName();
                  i++;
            }
            int j=0;
            int h=0;
            int k=0;
            int r=0;
            int comienzo=0;
            int tamaño=nombreFicheros.length;
            boolean esDirectorio=false;
            boolean existe=false;

            while(j<tamaño){
                  if(nombreFicheros[j]!=null && !esDirectorio){
                        if(nombreFicheros[j].contains(".pdf")){
                              FicheroBean fichero = new FicheroBean();
                              method=new PropFindMethod(nombreFicheros[j]);
                              dc.execute(method);
                              DAVFile archivo =method.getDAVFile();
                              if(archivo.getDisplayName()!=null){
                                    fichero.setName(archivo.getDisplayName());
                              }
                              else{
                                    fichero.setName(archivo.getFileName());
                              }

                              byte[] idintermedio =
code.encodeBase64(nombreFicheros[j].getBytes());
                              String id=new String(idintermedio,"UTF8");
                              fichero.setID(id);
                              fichero.setDate(archivo.getLastModified());
                              tablaficheros[r]=fichero;
                              j++;
```

```
                              r++;

                        }

                        else{
                              method= new PropFindMethod(nombreFicheros[j]);
                              dc.execute(method);
                              DAVFile directorio = method.getDAVFile();
                              Iterator<DAVFile> iter = directorio.children();
                              while(iter.hasNext()){
                                    DAVFile filedirectorio = iter.next();

      nombreFicherosDirectorios[h]=filedirectorio.getName();
                                    h++;
                                    existe=true;
                              }
                              j++;
                              esDirectorio=true;

                        }
                  }
                  else{
                        if(esDirectorio && existe){
                              existe=false;
                              k=0;
                              //Guardo el valor de j inicial(por donde se iba
recorriendo el valor de la lista
                              //de los documentos, y creo una variable comienzo
para a partir de ahí añadir los nuevos documentos
                              comienzo=j;
//                            j++;
                              while(nombreFicheros[comienzo]!=null){
                                    comienzo++;
                              }

                              while(nombreFicherosDirectorios[k]!=null){
                                    //El nombre de estos ficheros incluye el
directorio

      nombreFicheros[comienzo]=nombreFicherosDirectorios[k];
                                    k++;
                                    comienzo++;
                                    esDirectorio=false;
                              }

                        }else{
                              j++;
                              esDirectorio=false;
                        }
                  }

            }
            //Truncamos la tabla para que elimine los registros vacios
            int s=0;

            while(tablaficheros[s]!=null){
                  s++;
            }

            FicheroBean[] nuevatabla = new FicheroBean[s];
```

```
            s=0;
            while(tablaficheros[s]!=null){
                  nuevatabla[s]=tablaficheros[s];
                  s++;
            }


            return nuevatabla;
      }
```

En el método anterior puede observarse que a partir del identificador se sesión se construye la URL dinámica para acceder al servidor WebDAV, para recuperar posteriormente los documentos. Para obtener los hijos o "children" de una carpeta o subcarpeta, se usa el método PropFindMethod, y getDavFile, que permiten obtener todos los datos relativos a un documento.

9.3.2.3 Servicio web Obtiene Documento

Este servicio web implementa un método que permite obtener los datos relativos a un documento existente en el servidor WebDAV. De la lista de documentos devuelta por el servidor WebDAV anterior, el usuario debe seleccionar aquél que desee firmar. De este documento en concreto es del que se devuelven los datos para que el cliente los visualice.

Los parámetros necesarios para este método son:

- Identificador de sesión
- Identificador de documentos

El identificador de documento permite reconstruir la ruta donde se encuentra almacenado el documento en el servidor WebDAV, mientras que el identificador de sesión permite nuevamente iniciar una nueva sesión mediante la autenticación del usuario.

El método devuelve una estructura DocumentoBean que contiene el contenido del documento almacenado en forma de String codificado Base64, la fecha de la última modificación y el identificador de documento.

La implementación Java del método es la siguiente:

```
public DocumentoBean[] getDocument(String idSesion, String iddoc) throws
Exception{


            ResourceBundle prop =
ResourceBundle.getBundle("resources.conexion");

            String host = prop.getString("conexion.host");
            String puerto=prop.getString("conexion.port");
            int port = Integer.parseInt(puerto);
            DocumentoBean documentoBean = new DocumentoBean();
            //Transformamos a tabla para el servicio web
            DocumentoBean[] tabla = new DocumentoBean[1];

            Base64Coder code = new Base64Coder();
```

```
            byte[] codsesion =code.decodeBase64(idSesion.getBytes());
            String texto_decodificado = new String(codsesion,"UTF8");
            System.out.println("Texto decodificado:"+texto_decodificado);
            UtilesUrl utiles= new UtilesUrl();
            String[] datos=utiles.obtieneUsuarioycontrasena(texto_decodificado);


            byte[] coddocument = code.decodeBase64(iddoc.getBytes());
            String filename=new String(coddocument,"UTF8");



            DAVConnection dc=DAVConnectionPool.getDAVConnection(host, port,
false);

            dc.setUsername(datos[0]);
            AuthorizationInfo ai= new AuthorizationInfo(host,
                        port,
                        "Basic",
                        "KTWebDav Server",
                        Codecs.base64Encode(datos[0] + ":" + datos[1]));
            dc.setAuthorization(ai.toString());




            GetMethod method = new GetMethod(filename);
            dc.execute(method);

            byte[] contenido=method.getResponseBody();

            String content = new String(contenido,"UTF8");

            PropFindMethod method2=new PropFindMethod(filename);
            dc.execute(method2);
            DAVFile archivo=method2.getDAVFile();

            documentoBean.setName(archivo.getDisplayName());
            documentoBean.setDate(archivo.getLastModified());
            documentoBean.setID(iddoc);
            //Para decodificar en el lado del cliente:
            /* byte[] pr = content.getBytes();
                    FileOutputStream fos = new FileOutputStream("output.pdf");
                    InputStream inStream = new ByteArrayInputStream(pr);
                    byte buffer[] = new byte[2048];
                    int bytesRead;
                    while ((bytesRead = inStream.read(buffer)) >= 0) {
                        fos.write(buffer, 0, bytesRead);
                    }
                    inStream.close();
                    fos.close();

             //*/
            documentoBean.setContent(content);

            tabla[0]=documentoBean;

            return tabla;
      }
```

En el método anterior, además de utilizar nuevamente el método PropFindMethod de la clase DAV, también se ha utilizado el método GetMethod, que permite obtener el contenido del documento como un Stream.

9.3.2.4 Servicio web Obtener Hash

Una vez seleccionado el documento a firmar y después de visualizar que efectivamente se trata de la factura o albarán que se desea firmar, se debe proceder a realizar la firma del documento. Para liberar de cualquier carga extra al cliente se firmará el hash del documento que será calculado por el servidor de @firma, que permite obtener el hash del documento que se desea firmar. Previamente a este paso, y al proceso de verificación de la firma, es necesario registrar el documento en el servidor de @firma.

Estos dos pasos se realizarán en el mismo método Java que recibirá como parámetro:

- Identificador de sesión
- Identificador de documento

A partir del identificador de documento, es posible obtener todos los datos del documento, en especial el contenido y el nombre, que serán a su vez, los parámetros necesarios para registrar el documento en @firma.

La implementación Java del método es la siguiente:

```
public String obtenerHash (String idSesion, String iddoc) throws Exception{

            Aplicacion appl=new Aplicacion();
            DocumentoBean[] documentoBean =appl.getDocument(idSesion, iddoc);
            Firma3Fases firmaapp = new Firma3Fases();

            byte[] contentToSigned = documentoBean[0].getContent().getBytes();
            Base64Coder code = new Base64Coder();
            byte[] codificado =code.encodeBase64(contentToSigned);
            String contentToBeSigned = new String(codificado);

            //iddoc necesario para la llamada getData
            String idafirma=null;
            String hash=null;
            String id =
firmaapp.almacenar(documentoBean[0].getName(),"pdf",contentToBeSigned);
            if(id!=null && !id.trim().equals("")){
                  idafirma=id;
                  hash=appl.getData(idSesion, idafirma);
            }
            else{
                  idafirma= String.valueOf(-1);
            }
            return hash;

      }
```

En el código anterior, se hace una llamada a su vez al método getDocument() para obtener la información relativa al documento y que ya se comentó justo en el apartado anterior. Además, también se hace uso de la clase Firma3Fases de @firma, y más concretamente del método almacenar, que permite registrar el documento en @firma.

A continuación se implementa el código Java del método almacenar de la clase Firma3Fases de @firma:

```
public String almacenar(String fileName,String fileType,String content) throws
Exception
      {

            Document custodyDocumentRequest;
            String response;
            String docId=null;
            System.out.println("[COMIENZO DE PROCESO DE FIRMA 3 FASES]");

            ResourceBundle prop =
ResourceBundle.getBundle("resources.conexion");

            appId = prop.getString("conexion.idaplicacion");

            System.setProperty("javax.net.ssl.trustStore", KEYSTORE_PATH);

            System.out.println("[COMIENZO DE PROCESO DE FIRMA 3 FASES]");

            //Preparación de la petición al servicio Web de AlmacenarDocumento
            System.out.println(".[Preparando la petición al servicio Web " +
custodyDocumentWebServiceName + "...]");
            custodyDocumentRequest =
UtilsWebService.prepareCustodyDocumentRequest(appId,fileName,fileType,content);
            System.out.println(".[/Petición correctamente preparada]");

            //Lanzamiento de la petición WS
            System.out.println(".[Lanzando la petición...]");
            System.out.println("..[peticion]");

      System.out.println(XMLUtils.DocumentToString(custodyDocumentRequest));
            System.out.println("..[/peticion]");
            response =
UtilsWebService.launchRequest(ENDPOINT,custodyDocumentWebServiceName,custodyDoc
umentRequest);
            System.out.println("..[respuesta]");
            System.out.println(response);
            System.out.println("..[/respuesta]");
            boolean coderror = false;
            if (!UtilsWebService.isCorrect(response))
            {
                  System.err.println();
                  System.err.println("La petición de Almacenamiento del
documento " + fileToBeSigned + " no ha sido satisfactoria. Saliendo ...");
                  coderror=true;
            }
            if(!coderror){
            System.out.println(".[/Petición correctamente realizada]");

            //Obtención del identificador del documento custodiado
```

```
            System.out.println(".[Extrayendo el identificador del documento de
la respuesta...]");
            docId = UtilsWebService.getInfoFromDocumentNode(response,
"idDocumento");
            System.out.println(".[/Identificador del documento correctamente
extraído de la respuesta]");
            }


            return docId;
      }
```

Este método invoca a su vez los servicios web disponibles en el servidor de @firma y que están disponibles en el CD de instalación de la aplicación.

9.3.2.5 Servicio web Custodiar Documento

Una vez firmado el documento por el cliente, es necesario devolver el fichero .p7 que contenga la firma al servidor WebDAV. Para ello, es necesario verificar previamente que la firma realizada ha sido correcta y que se mantiene la integridad de los datos firmados.

Los parámetros necesarios para la implementación del método son:

- Identificador de sesión
- DocumentoBean

El parámetro DocumentoBean es una estructura que contiene como nombre la ruta del documento que se ha firmado y que se utilizará para devolver el fichero de firma al servidor, que tendrá como nombre el del documento más una extensión llamada *_firma*. Además, el contenido de esta estructura será el del fichero de firma .p7, codificado en Base64.

El código Java del método que se encarga de implementar dicha función es el siguiente:

```
public String custodiarDocumento(String idSesion, DocumentoBean documento)
throws Exception{

            ValidarFirma valida = new ValidarFirma();
            ResourceBundle prop =
ResourceBundle.getBundle("resources.conexion");
            Base64Coder code = new Base64Coder();

            String host = prop.getString("conexion.host");
            String puerto=prop.getString("conexion.port");

            int port = Integer.parseInt(puerto);

            byte[] codsesion =code.decodeBase64(idSesion.getBytes());
            String texto_decodificado = new String(codsesion,"UTF8");
            System.out.println("Texto decodificado:"+texto_decodificado);
            UtilesUrl utiles= new UtilesUrl();
            String[] datos=utiles.obtieneUsuarioycontrasena(texto_decodificado);
```

```
            DAVConnection dc=DAVConnectionPool.getDAVConnection(host, port,
false);

            dc.setUsername(datos[0]);
            AuthorizationInfo ai= new AuthorizationInfo(host,
                        port,
                        "Basic",
                        "KTWebDav Server",
                        Codecs.base64Encode(datos[0] + ":" + datos[1]));
            dc.setAuthorization(ai.toString());


            String path = utiles.obtieneUrlServidor(datos[0]);
            String filename=documento.getName();
            GetMethod method = new GetMethod(filename);
            dc.execute(method);

//          Obtenemos el contenido original
            byte[] contenidoOriginal=method.getResponseBody();

            byte[] contenidoOriginalBase64=code.encodeBase64(contenidoOriginal);


//          Obtenemos el contenido firmado
            String contenidoFirmado = documento.getContent();
            byte[] contentToSigned = contenidoFirmado.getBytes();
            String respuesta=idSesion;

            byte[] codificado =code.encodeBase64(contentToSigned);
            String datosSigned= new String(codificado);


            boolean error =valida.run(datosSigned,contenidoOriginalBase64);

            if(error)
                  respuesta=String.valueOf(-1);

            else{

                  String nombreDocumento=path+documento.getName()+"_firma.p7";
                  byte[] contenidoDevuelto =
code.decodeBase64(documento.getContent().getBytes());
                  PutMethod method2=new
PutMethod(nombreDocumento,contenidoDevuelto);

                  dc.execute(method2);
                  int status=method2.getStatus();

                  if(status >= 200 && status < 300)
                        respuesta=idSesion;
                  else
                        respuesta=String.valueOf(-1);

            }

            return respuesta;

      }
```

Este método invoca la clase ValidarFirma que contiene el método run() que permite validar la firma a partir del contenido pasado en @firma. Su implementación Java es la siguiente:

```
public Boolean  run(String datosSigned, byte[] data) throws Exception
      {

            System.out.println("[COMIENZO DE PROCESO DE VALIDACIÓN DE FIRMA]");

            ResourceBundle prop =
ResourceBundle.getBundle("resources.conexion");


            appId = prop.getString("conexion.idaplicacion");

            eSignatureFormat = "PKCS7";
            mode = Integer.parseInt("0");


            System.setProperty("javax.net.ssl.trustStore", KEYSTORE_PATH);

            // Lectura de la Firma Electrónica a validar
            System.out.println(".[Obteniendo la Firma Electrónica del fichero "
+ eSignaturePath + "...]");

                  System.out.println(".[Calculando el hash del fichero original
con el algoritmo de hash " + hashAlgorithm + "...]");
                  System.out.println(".[/Hash correctamente calculado]");
                  System.out.println(".[Codificando el fichero original " +
filePath + " en Base64...]");

                  System.out.println(".[/Fichero original correctamente
codificado]");
            }

            // Preparación de la petición al servicio Web de ValidarFirma
            System.out.println(".[Preparando la petición al servicio Web " +
signatureValidationWebServiceName + "...]");
            Document eSignatureValidationRequest = null;
            try
            {
                  if (mode == 1)
                                                eSignatureValidationRequest =
UtilsWebService.prepareValidateSignatureRequest(appId,datosSigned,
eSignatureFormat, null, null, data);
            }
            catch (Exception e)
            {
                  System.err.println(".[/Petición incorrectamente preparada]");
                  System.exit(-1);
            }
            System.out.println(".[/Petición correctamente preparada]");

            // Lanzamiento de la petición WS
            System.out.println(".[Lanzando la petición...]");
            System.out.println("..[peticion]");


System.out.println(XMLUtils.DocumentToString(eSignatureValidationRequest));
```

```
            System.out.println("..[/peticion]");
            String response = UtilsWebService.launchRequest(ENDPOINT,
signatureValidationWebServiceName, eSignatureValidationRequest);
            System.out.println("..[respuesta]");
            System.out.println(response);
            System.out.println("..[/respuesta]");
            boolean coderror=false;
            if (!UtilsWebService.isCorrect(response))
            {
                  System.err.println();
                  System.err.println("La petición de verificación de la Firma
Electrónica " + eSignaturePath + " no ha sido satisfactoria. Saliendo ...");
                  coderror=true;
            }
            System.out.println(".[/Petición correctamente realizada]");

            // Obtención del detalle de la respuesta
            System.out.println(".[Extrayendo la información detallada de la
respuesta...]");
            String descriptionValue =
UtilsWebService.getXMLChildsFromDocumentNode(response, "descripcion");
            System.out.println(descriptionValue);
            System.out.println(".[/Información detallada correctamente extraída
de la respuesta]");

            System.out.println("[/PROCESO DE VALIDACIÓN DE FIRMA FINALIZADO]");
            return coderror;
      }
```

El método que valida la firma, como puede comprobarse, invoca los servicios web disponibles en @firma que permiten verificar la integridad de la firma.

La comprobación de firma se debe realizar, en dos pasos:

- Comprobar que el documento no ha sido alterado (Integridad).
- Comprobar que el certificado del firmante es válido (Identidad).

La verificación de integridad puede traducirse en *garantizar que lo que se envío es lo que se ha recibido*. Comprobar la integridad de la firma garantizará que firma y datos "son uno", es decir, que lo que hemos recibido no ha sido manipulado o corrompido por el camino. Aún así, un "intermediario malintencionado" podría reemplazar datos y firmar por su propia firma y sus propios datos, de modo la integridad de la firma daría correcta. Es por eso que resulta imprescindible comprobar la identidad del firmante.

La comprobación de identidad, a su vez, pasa por:

- Comprobación de que certificado no está caducado.
- Comprobar que el certificado ha sido realmente emitido por una entidad certificadora de máximo nivel (RootCA).
- El certificado no está revocado ni suspendido, mediante comprobación de las CLRs de la

CA.

Una vez verificada la firma, se puede devolver al servidor WebDAV, donde aparecerá con la extensión "_firma" y junto al documento original.

9.3.3 Despliegue de los servicios web

Para desplegar los servicios web anteriores, es necesario tener un servidor de aplicaciones Apache Tomcat con el módulo de Axis instalado. Más concretamente, en el desarrollo de este proyecto se ha utilizado axis2 versión 1.3. En un capítulo posterior se explicará con detenimiento la instalación de este módulo.

Antes de comenzar, es necesario entender correctamente la estructura del módulo de axis2. El lado servidor de Axis2 se puede desplegar en cualquier motor de Servlets, y tiene la estructura siguiente:

```
axis2-web
META-INF
WEB-INF
    classes
    conf
        axis2.xml
    lib
        activation.jar
        ...
        xmlSchema.jar
    modules
        modules.list
        addressing.mar
        ...
        soapmonitor.mar
    services
        services.list
        aservice.aar
        ...
        version.aar
    web.xml
```

El objetivo, por tanto, es poder obtener un archivo .aar propio que, una vez copiado en la carpeta ../services, pueda ser desplegado al arrancar el servidor Apache-Tomcat.

El primer paso será conseguir el archivo .wsdl a partir de la clase java que implemente el método. Para poder generar los servicios web a partir de las clases .java, es necesario tener desplegado el módulo de axis2-1.3 (un archivo .zip que no tiene nada que ver con el módulo anterior que hay que desplegar en Tomcat) en alguna carpeta cualquiera, ya que posee la variable WSDL2Java y Java2WSDL, que serán necesarias para desplegar los servicios web.

Para conseguir el archivo .wsdl a partir de la clase Java compilada, se ejecutará del siguiente comando:

```
%AXIS2_HOME%/bin/java2wsdl -cp . -cn clase.class -of StockQuoteService.wsdl
```

Donde AXIS2_HOME es la variable de entorno que contiene el directorio C:/axis2-1.3.

Una vez obtenido el archivo .wsdl para generar y desplegar el servicio utilizando el Axis2 Databinding Framework (ADB), es necesario generar el *skeleton* utilizando la herramienta WSDL2Java tecleando lo siguiente en el directorio AXIS2_HOME/:

```
%AXIS2_HOME%/bin/WSDL2Java -uri clase.wsdl -p service.adb -d adb -s -ss -sd
-ssi -o build/service
```

El significados de las distintas opciones es el siguiente:

- La opción `-d adb` especifica la utilización de Axis Data Binding (ADB).
- La opción `-s` especifica síncrono o llamadas con bloqueo solamente.
- La opción `-ss` crea el código del lado del servidor (skeleton y archivos relacionados).
- La opción `-sd` crea un descriptor del servicio (archivo services.xml).
- La opción `-ssi` crea un interfaz para el skeleton del servicio.

Los archivos del servicio se deberán estar ahora en subdirectorio build/service. Uno de dichos archivos es el llamado *skeleton* que deberá ser rellenado con el código .java de cada clase.

Una vez rellenado el skeleton con el código correspondiente, se puede construir el proyecto tecleando el comando siguiente en el directorio build/service:

```
%ANT_HOME%/ant jar.server
```

donde previamente se ha tenido que definir la variable de entorno ANT_HOME. Si todo va bien, deberá ver el mensaje BUILD SUCCESSFUL en la ventana de comandos, y el archivo de Clase.aar en el directorio build/service/build/lib.

Lo próximo será copiar ese archivo al directorio de webapps/axis2/Web-INF/services del motor de servlets. De esta forma se ha conseguido el objetivo, conseguir desplegar los distintos servicios web.

A continuación, es necesario arrancar el servidor de aplicaciones Apache-Tomcat. Tecleando la siguiente dirección, deberíamos ver que el módulo de axis2 se encuentra correctamente instalado y funcionando:

http://localhost:8080/axis2

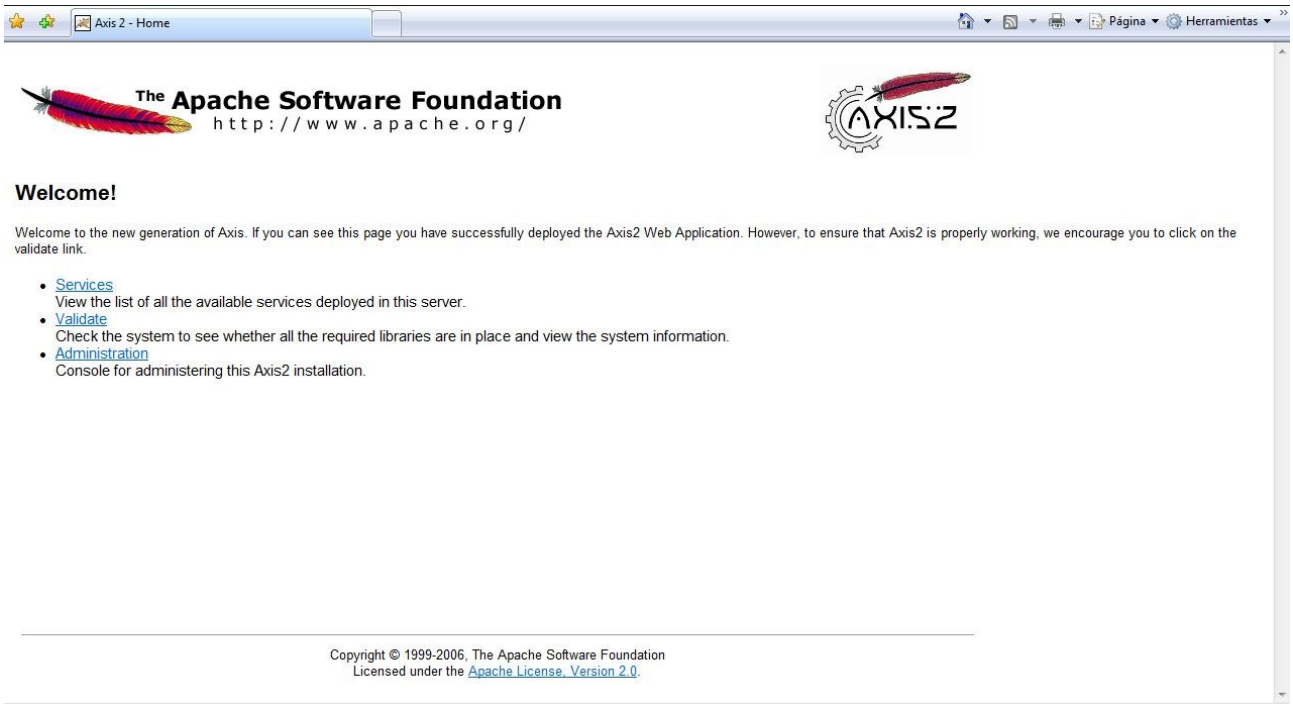


Figura 9.2 Módulo Axis2 integrado en Apache-Tomcat

Para poder ver los servicios web desplegados, basta con acceder a la opción *Services* mostrada en la imagen anterior:

http://localhost:8080/axis2/services/listServices

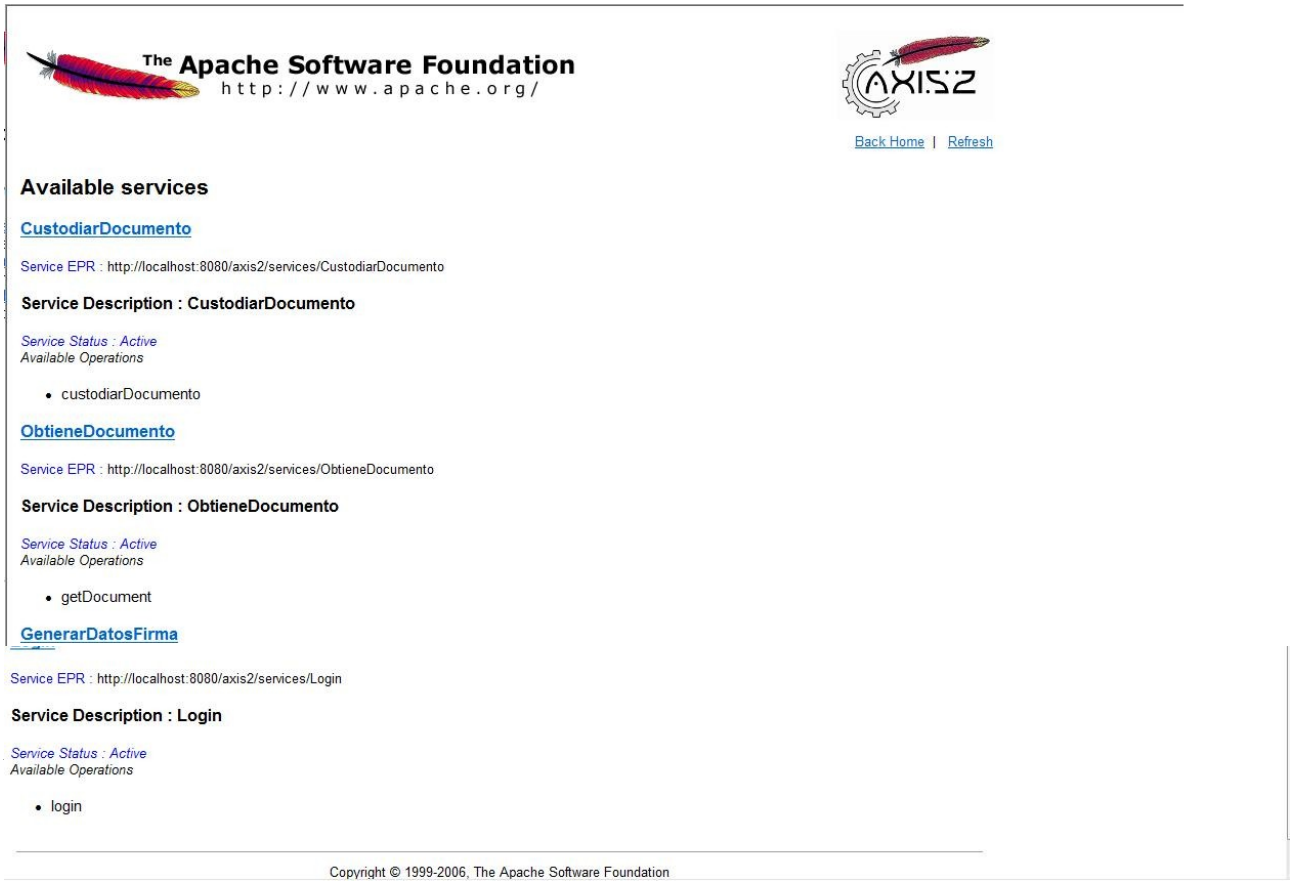


Figura 9.3. Lista de servicios web desplegados

Para poder visualizar el .wsdl de alguno de ellos, basta con hacer click sobre el que queramos ver. Un ejemplo del contenido mostrado sería el siguiente:

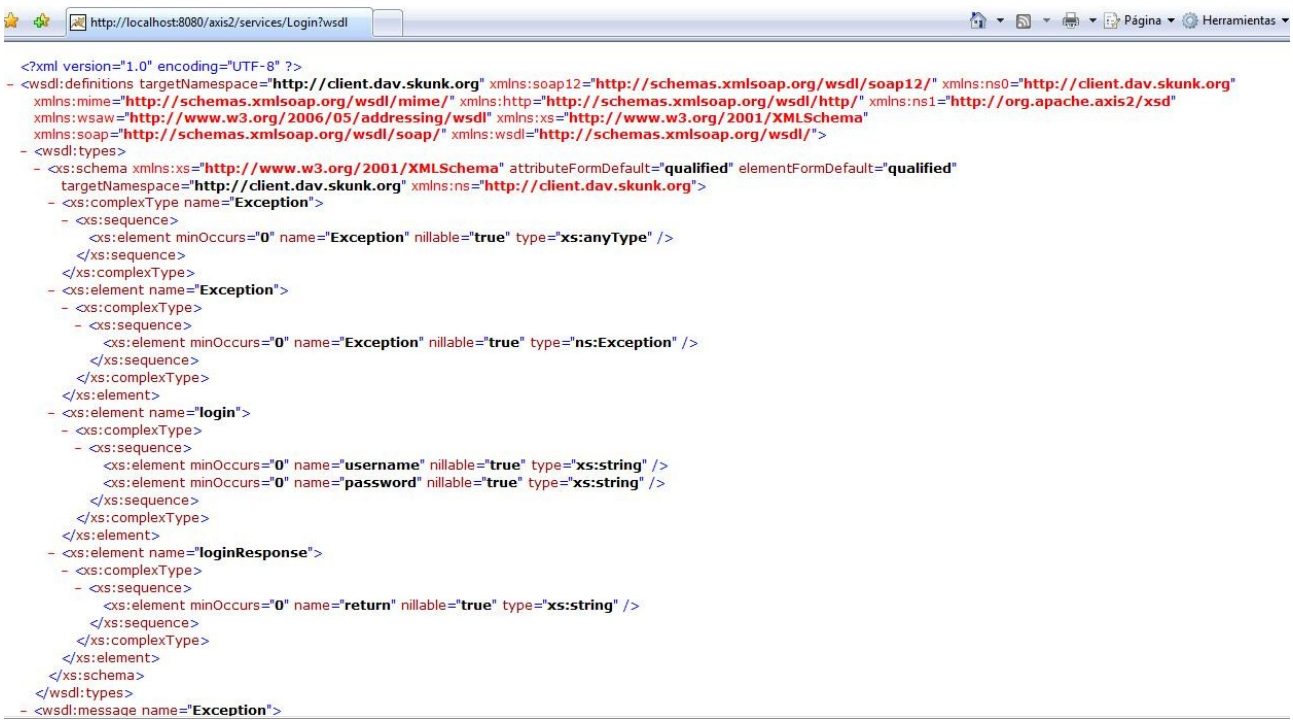


Figura 9.4. Archivo .wsdl

Hasta aquí la parte correspondiente al desarrollo del servidor web XML. A continuación, pasaremos a analizar el cliente móvil.

9.3.4 Desarrollo del cliente

9.3.4.1 Introducción

El cliente desarrollado se trata de una aplicación escrita en lenguaje Java diseñada para dispositivos móviles que soporten J2ME. El objetivo de dicho cliente es realizar la firma electrónica del documento que el usuario desee firmar. Para ello, teniendo en cuenta las limitaciones computacionales de los dispositivos móviles, se han desarrollado una serie de servicios web con el fin facilitar las tareas del lado del cliente. El dispositivo móvil deberá invocar dichos servicios web siempre que sea necesario:

- Para obtener la lista de documentos del servidor WebDAV disponibles para firmar.
- Para obtener información sobre el documento en concreto que el usuario seleccione para firmar.
- Para obtener el hash del documento que se desea firmar y registrarlo en @firma.

- Para verificar la validez de la firma y devolver el documento firmado al servidor WebDAV

Las especificaciones de terminal móvil para que pueda funcionar la aplicación son la siguientes:

- JSR-75: permite acceder al sistema de ficheros del dispositivo y tarjetas extraibles. Esta especificación es requerida para disponer de acceso al certificado de usuario, obtener y almacenar el fichero original y generar el fichero de firma PKCS7.
- JSR-172: permite el desarrollo de clientes que se comuniquen con servicios web externos.

9.3.4.2 Bouncy Castle en J2ME

Para poder realizar la firma electrónica en el dispositivo móvil, se ha utilizado la librería Bouncy Castle, explicada en el capítulo 5 de esta memoria. Es una librería de seguridad libre muy completa: cifra, descifra, calcula hash y muestra la salida en diferentes formatos, además de poder parsear y utilizar todo tipo de certificados.

Otros de los principales motivos por los que se ha escogido esta librería es que es de licencia libre y una de las librerías criptográficas más estables que existen actualmente. No obstante, se han eliminado clases innecesarias, así como nuevas implementaciones necesarias para adaptar su uso a J2ME.

Bouncy Castle tiene una versión especial para J2ME (la versión de Java para móviles), pero no trae todo lo necesario para poder firmar con certificados y enviarlo en formato PKCS7. Para poder firmar y tratar los certificados del usuario, se ha adaptado parte de la librería completa de BouncyCastle a J2ME.

Los paquetes modificados son:

- java, que se renombra por javafake ya que J2ME no permite una ampliación del paquete java.

Los paquetes añadidos son :

- javafake.util
- javafake.io.StringReader
- org.bouncycastle.openssl

Los paquetes que seeliminan para ahorrar memoria son:

- org.bouncycastle.crypto
- org.bouncycastle.bcpg

Este proceso de adaptación consta fundamentalmente de tres pasos:

1. Eliminar o reemplazar las referencias no válidas:

Hay referencias a librerías que son inexistentes en J2ME o que su portabilidad completa a J2ME supondría un coste demasiado alto. Si son librerías para soportar aplicaciones que no vamos a usar, las eliminamos. Si son librerías que vamos a necesitar, o se simplifican o se adaptan a su vez a J2ME.

2. Simplificar funcionalidades:

Muchas de las funcionalidades de las librerías no serán utilizadas en nuestra aplicación, pero complican el proceso de adaptación. Por regla general, dichas funcionalidades que no vamos a usar se eliminarán.

3. Adaptar el código a J2ME

- No existen las clases genéricas, por ejemplo, especificar que una lista será de un tipo concreto.
- Desaparecen muchas excepciones, que no están definidas en J2ME.
- Algunas clases tienen menos funcionalidades en J2ME. Si hacen falta dichas funcionalidades, hay que reimplementar la clase.

Hay que tener también cuidado porque muchas de las implementaciones de los cifrados están limitados por las leyes de EEUU, no dejando firmar con claves de más de 512 bit

A continuación, veremos con más detalle cada una de las clases utilizadas.

9.3.4.3 Manejo de ficheros

Para poder utilizar la API de acceso al sistema de ficheros, el móvil tiene que soportar, tal y como se comentó anteriormente, JSR-75.

Para poder utilizar el sistema de ficheros sin que pida confirmación continuamente, hay que conseguir los permisos necesarios:

- En Netbeans6 (plataforma de desarrollo) esto se hace en Tools->Java Platforms. Seleccionamos el emulador que usamos y en Tools&Extensions, pulsamos en Open Preferences. La Seguridad la ponemos como máxima.

- En el móvil se hace firmando el ejecutable .jar.

1. Lectura de ficheros

Para poder leer un fichero, es necesario abrir una conexión (FileConnection) indicando la ruta del fichero que se quiere leer (se podría conseguir con la clase FileBrowser o similar) y mediante un InputStream vamos obteniendo los bytes que corresponden al fichero. Este array de bytes se puede convertir posteriormente a String.

En J2ME (la versión de Java para móviles) no es posible saber el tamaño del fichero antes de leerlo, así que es necesario realizar dos accesos: primero acceder para ver el tamaño del fichero y posteriormente para leerlo.

La clase Java que implementa la lectura de los ficheros es la siguiente:

```
    byte[] datos = new byte[1];
        InputStream is;
        FileConnection fc = new ... //Con los parámetros que haga falta
        byte[] documento;
        int tamaño = 0;

        try
        {
             if (fc.exists() && fc.canRead())
             {
                 is = fc.openInputStream();

                 //Primero recorremos el fichero para ver su tamaño:
                 int auxiliar = 0;
                 while((auxiliar = is.read(datos)) > 0)
                    tamaño += auxiliar;

                 //Una vez tenemos el tamaño, creamos el resultado:
                 documento = new byte[tamaño-1];

                 //Cerramos y volvemos a abrir, para reiniciar el is:
                 is.close();
                 is = fc.openInputStream();

                 //Y ahora pasamos los datos a documento:
                 is.read(documento);
                 is.close();
             }
             else
                 error = "El fichero no existe o no es accesible/legible";

             if(tamaño == 0)
                 error = "El fichero estaba vacío, pero existía";
            }
        }
        catch (Exception e)

                                        ...
```

2. Escritura

La escritura es más sencilla, puesto que no hay que hacer tantas comprobaciones previas. La clase que la implementa es la siguiente:

```
  public void writeFile(byte[] data, String pathAndFilename)
    {

        try {
            //Abrimos una conexión con permiso de lectura/escritura
               FileConnection fc = (FileConnection) Connector.open("file:///" +
pathAndFilename, Connector.READ_WRITE);

            //Si el fichero no existe, lo creamos
            if (!fc.exists())
```

```
                fc.create();
            else
                fc.truncate(0); //Si el fichero existe, sobreescribimos en él

            OutputStream os = fc.openOutputStream();
            os.write(data);
            os.flush();

        } catch (Exception e)
          ...
```

3. FileBrowser

Los dispositivos CDC (PDAs) tienen una clase llamada FileBrowser muy útil para recorrer el sistema de ficheros. Puesto que esta clase no existe en los móviles (CDCL), hay que implementarla a mano.

Nuestro código constará de dos funciones: una función para mostrar el contenido de un directorio y otra función para navegar entre directorios. Combinadas, simulan el FileBrowser. La variable browser será un List en el sentido GUI de J2ME, es decir, un componente visual que muestra una lista de elementos en la que puedes seleccionar uno de ellos. El método que implementa lo anterior se denomina showCurrDir(), y muestra en browser el contenido del directorio en el que nos encontramos actualmente. Su implementación Java es la siguiente:

```
public void showCurrDir()
{
   Enumeration e;
   FileConnection currDir = null;

   //Si hay algo seleccionado, lo ponemos en la variable fileName
   if(browser.getSelectedIndex() != -1)
       fileName = browser.getString(browser.getSelectedIndex());

   //Si es un directorio o es la primera vez que pasamos por aquí:
   if (fileName.endsWith(SEP_STR) ||
       fileName.equals(UP_DIRECTORY) ||
       fileName.equals(""))
      {
       //Cambiamos de directorio:
       traverseDirectory();

       //Borramos las posibles entradas de la lista:
       browser.deleteAll();

       try
       {

         //Si es el root, saca los elementos del root
         if (MEGA_ROOT.equals(currDirName))
         {
           e = FileSystemRegistry.listRoots();
         }
         else
         {
           //Si no es el root, saca los elementos de la ruta
```

```
            currDir = (FileConnection)Connector.open(
              "file://localhost/" + currDirName);
            e = currDir.list();

            //Añadimos la entrada ".." para subir de directorio
            browser.append(UP_DIRECTORY, null);
          }

          //Extraemos los elementos de la carpeta y los ponemos en la lista
          while (e.hasMoreElements())
          {
            fileName = (String)e.nextElement();

            //El if/else es por si quieres poner iconos diferentes según
            //si es carpeta o fichero
            if (fileName.charAt(fileName.length()-1) == SEP)
            {
              browser.append(fileName, null);
            }
            else
            {
              browser.append(fileName, null);
            }
          }

          //Cerramos el directorio
          if (currDir != null)
          {
            currDir.close();
          }
        }
        catch (IOException ioe)
        {}
        //Cambiamos para mostrar el siguiente directorio
        switchDisplayable(null, getBrowser());
     }
    //Entonces no era un directorio, así que abrimos el fichero seleccionado:
     else
        openFile();
    }
```

En el método anterior se hace uso de la funcion traverseDirectory, que es la encargada de cambiar de directorio. Su implementación Java se muestra a continuación:

```
   public void traverseDirectory()
    {
        //Si estamos en la raiz
        if (currDirName.equals(MEGA_ROOT))
          currDirName = fileName;
        //Si estamos yendo al directorio superior
        else if (fileName.equals(UP_DIRECTORY))
        {
          //Miramos donde están los "/"
          int i = currDirName.lastIndexOf(SEP, currDirName.length()-2);
          //Si el directorio superior no es la raiz, quitamos la última parte
          if (i != -1)
            currDirName = currDirName.substring(0, i+1);
          else
```

```
            currDirName = MEGA_ROOT;
        }
        //Si no estamos en la raiz ni vamos al directorio superior
        else
        {
            //Pues construimos el camino:
            currDirName = currDirName + fileName;
        }
    }
```

9.3.4.4 Certificados. Extracción de claves

Aunque BouncyCastle tiene capacidad para procesar todo tipo de certificados, resultaba imposible adaptar esa parte a J2ME, ya que utilizaba librerías que no podían ser portadas. Por tanto, parte de esa implementación ha tenido que ser reimplementada de forma más sencilla.

La aplicación es capaz de procesar todo tipo de certificados en formato PEM. El formato PEM no es más que un fichero en codificación Base-64 del certificado en formato DER. El formato DER codifica en ASN1 un certificado X.509:

```
* Certificado
      o Versión
      o Número de serie
      o ID del algoritmo
      o Emisor
      o Validez
            + No antes de
            + No después de
      o Sujeto
      o Información de clave pública del sujeto
            + Algoritmo de clave pública
            + Clave pública del sujeto
      o Identificador único de emisor (opcional)
      o Identificador único de sujeto (opcional)
      o Extensiones (optional)
            + ...
* Algoritmo usado para firmar el certificado
* Firma digital del certificado
```

En JavaSE existe un almacén de claves (KeyStore) en donde se pueden importar los datos de un certificado para usarlo posteriormente con la clase X509Certificate de BouncyCastle. Pero este método era demasiado complejo para J2ME (que no trae ninguna clase para poder implementarlo), así que tuvo que ser desechado. Había una aproximación mediante la clase UserManagementStore que sí viene incluída en J2ME, pero sólo si el certificado era generado en el propio móvil, lo que no nos era útil en nuestro caso concreto.

Finalmente se ha optado por leer y procesar el certificado con una implementación propia, almacenando en la clase KeyPair (par de claves) tanto la clave pública como la privada, para su uso posterior.

Para leer el certificado utilizamos la clase PEMReader. Con la clase X509CertificateStructure se

puede procesar el certificado del usuario así que lo que devuelve PEMReader sería un certificado de ese tipo.

También necesitaremos las claves pública y privada del certificado. Se extraerán con PEMReader, que devolvería un KeyPair (clase que contiene al par de claves).

Para extraer el módulo y los exponentes (privado y público) de la cadena que viene en el certificado se puede usar el siguiente código donde *reader* es un InputStream al fichero del certificado y *pf* es una variable que contiene la clave para descifrar el certificado.

```
                    PEMReader pemreader = new PEMReader(reader, pf);
                    Object obj;
                    while((obj = pemreader.readObject()) != null)
                    {
                        if(obj instanceof Certificado)
                            certificado = (Certificado)obj;
                        else if(obj instanceof KeyPair)
                            keypair = (KeyPair)obj;
                        else
                            throw new Exception("Error en la extracción de los
datos del certificado.");
                    }
```

La función readObject() de PEMReader lee el fichero que se le pasa (como InputStream) y extrae lo que encuentre: bien el certificado o bien la clave privada.

```
public Object readObject() throws IOException
            {
                String  line;

                while ((line = readLine()) != null)
                {
                    //Si empieza la clave privada, pasamos a readKeyPair
                    if (line.indexOf("-----BEGIN RSA PRIVATE KEY") != -1)
                    {
                        try
                        {
                            //Devolvería el par de claves (KeyPair):
                            return readKeyPair("-----END RSA PRIVATE KEY");
                        }
                        catch (Exception e)
                        {
                            throw new IOException("Problema al leer la Clave
Privada RSA: " + e.getMessage());
                        }
                    }
                    //Si lo que empieza es el certificado, pues leemos los
datos
                    else if (line.indexOf("-----BEGIN CERTIFICATE") != -1)
                    {
                        //Devuelve el certificado (X509CertificateStructure):
                        return readCertificate("-----END CERTIFICATE");
                    }
                }
                //Si no encuentra nada:
                return null;
```

```
            }
```

PEMReader interpreta y extrae las claves del certificado con la función readKeyPair():

```
private KeyPair readKeyPair(String  type, String  endMarker) throws Exception
    {
        boolean         isEncrypted = false;
        String          line = null;
        String          dekInfo = null;
        StringBuffer    buf = new StringBuffer();

        //Lee el fichero y según la cadena, sacará un dato u otro:
        while ((line = readLine()) != null)
        {
            if (line.startsWith("Proc-Type: 4,ENCRYPTED"))
                isEncrypted = true;
            else if (line.startsWith("DEK-Info:"))
                dekInfo = line.substring(10);
            else if (line.indexOf(endMarker) != -1)
                break;
            else
                buf.append(line.trim());
        }

        // extrae la clave
        byte[] keyBytes = Base64.decode(buf.toString());

       //Si está cifrado, entonces hay que descifrar la clave:
        if (isEncrypted)
        {
            if (pFinder == null)
                throw new IOException("Se necesita una contraseña que no se ha
especificado");

            char[] password = pFinder.getPassword();

            if (password == null)
                throw new IOException("La contraseña es nula, pero se necesita
una contraseña");

            StringTokenizer tknz = new StringTokenizer(dekInfo, ",");
            String          dekAlgName = tknz.nextToken();
            byte[] iv = Hex.decode(tknz.nextToken());

            keyBytes = PEMUtilities.crypt(false, provider, keyBytes, password,
dekAlgName, iv);
        }

        //Lo convertimos a un formato donde sea fácil extraer los datos:
        ASN1Sequence asn1seq = (ASN1Sequence)
ASN1Sequence.fromByteArray(keyBytes);
        RSAPrivateKeyStructure asn1PrivKey = new
RSAPrivateKeyStructure(asn1seq);

            // extraemos los parametros relevantes de las claves:
            BigInteger modulo = asn1PrivKey.getModulus();
            BigInteger privateExponent = asn1PrivKey.getPrivateExponent();
```

```
            BigInteger publicExponent = asn1PrivKey.getPublicExponent();

        //Generamos el KeyPair para devolverlo:
        RSAKeyParameters rsapriv = new RSAKeyParameters(true, modulo,
privateExponent);
        RSAKeyParameters rsapub = new RSAKeyParameters(false, modulo,
publicExponent);

        KeyPair kp = new KeyPair(rsapub, rsapriv);

        return kp;
    }
```

Para descifrar las claves se hace uso de PEMUtilities (función crypt). PEMUtilities es otra clase que hemos tenido que adaptar a J2ME. EL PEMUtilities original es capaz de procesar muchos más tipos de claves y algoritmos que nuestra versión reducida, ya que lo que buscábamos era tener que portar a J2ME el menor número de clases posibles.

```
public final class PEMUtilities
{
    public static byte[] crypt(boolean encrypt, String  provider, byte[]
bytes, char[]  password, String  dekAlgName, byte[]  iv)
        throws IOException
    {
      AlgorithmParameterSpec paramSpec = new IvParameterSpec(iv, 0, iv.length);
        String                  blockMode = "CBC";
        String                  padding = "PKCS5Padding";

        if (dekAlgName.startsWith("DES-EDE"))
        {
            String alg = "DESede";
            boolean des2 = !dekAlgName.startsWith("DES-EDE3");
            Key sKey = getKey(password, alg, 24, iv, des2);
        }
        else //Debe ser algún tipo de encriptación que no soportamos
            throw new IOException("unknown encryption with private key");

        byte[] resultado = new byte[bytes.length];

        try
        {
            byte[] clave = sKey.getEncoded();
            byte[] datos = bytes;

            //Procedemos a descifrar la clave:
            DESEngine desen = new DESedeEngine();
            org.bouncycastle.crypto.CipherParameters cparam = new
ParametersWithIV(new KeyParameter(clave), iv);
            cCBCBlockCipher bc = new CBCBlockCipher(desen);

            cbc.init(encrypt, cparam);

            if(datos.length % 8 == 0)
                for(int i = 0; i < datos.length;i+=8)
                    cbc.processBlock(datos, i, resultado, i);
            else
                throw new IOException("La longitud no era adecuada, no se ha
```

```
pasado un argumento correcto");
        }
        catch
             ...
```

9.3.4.5 Implementación de la firma electrónica

Aunque en Java SE hay muchas formas de firmar un documento, en principio en J2ME no hay ninguna forma directa de hacerlo. Por eso se utilizan algunas clases de Bouncy Castle adaptadas a J2ME y algunas clases más de implementación propia.

Todo el tratamiento de datos tiene que hacerse con arrays de bytes, para evitar los problemas derivados de las diferentes codificaciones.

Para unificar todas las funciones de firma electrónica, se ha añadido la clase org.bouncycastle.openssl.Certificado.java. Esta clase está diseñada de forma transparente para que si se quieren añadir nuevas formas de firmar con otros tipos de certificados, se pueda hacer fácilmente añadiendo más clases que implementen CertificadoGenerico y que sustituyan a Certificado donde haga falta:

```
public interface CertificadoGenerico {

    boolean firmar(byte[] d);

    String getErrors();

    boolean isVoid();

    void set(byte[] c);

    void setKeyPair(KeyPair k);

    ASN1Object toASN1Object();

    SignedData toSignedData();

    String toString();

}
```

La parte más compleja de estas clases reside en la funcionalidad firmar:

```
//Devuelve verdadero si la firma es correcta:
    public boolean firmar(byte[] d)
    {
        boolean resultado = false;
        documento = d;
        signer = new PSSSigner(rsaEng, digEng, d.length);

        try
        {
            if(keypair == null)
                throw new Exception("No hay claves para firmar.");
```

```
            //Creamos el PSSSigner
            CipherParameters cparam = (CipherParameters) keypair.getPrivate();
            signer.init(true, cparam);

            //Firmamos y lo guardamos en "firma"
            signer.update(d, 0, d.length);
            firma = signer.generateSignature();

            //Ahora comprobamos la firma:
            RSAKeyParameters pubKey = (RSAKeyParameters)keypair.getPublic();

            //Sacamos el modulo y el exponente publico:
            String mod = new
String(Base64.encode(pubKey.getModulus().toByteArray()));
            String pubExp = new
String(Base64.encode(pubKey.getExponent().toByteArray()));

            if(verify(mod, pubExp))
                return true;
            else
                this.setErrors("No se pudo verificar la firma: " + new
String(Hex.encode(firma), "ASCII"));
        }
        catch(Throwable e)
        {
           this.setErrors("Error firmando: "+ e.getMessage());
        }
        return resultado;
    }
```

Certificado.java también tiene la función para validar la firma. Esta función se llama justo después de firmar para comprobar que no hubo errores imperceptibles en el proceso.

```
    //Funcion que verifica la firma
    private boolean verify(String mod, String pubExp)
    {
        //Extraemos los parametros
        BigInteger modulus = new BigInteger(Base64.decode(mod));
        BigInteger exponent = new BigInteger(Base64.decode(pubExp));

        //Sacamos la clave publica (que es la que verifica)
        RSAKeyParameters pubKey = new RSAKeyParameters(false, modulus,
exponent);

        signer.init(false, pubKey);

        //Actualizamos PSSSigner
        signer.update(documento, 0, documento.length);

        //verifica
        return signer.verifySignature(firma);
    }
```

El formato PKCS7 está definido en la RFC 2315. Consiste en un tipo ContentInfo:

```
   ContentInfo ::= SEQUENCE {
     contentType ContentType,
     content
       [0] EXPLICIT ANY DEFINED BY contentType OPTIONAL }

   ContentType ::= OBJECT IDENTIFIER
```

el cual tiene un objeto de tipo signedData en su contenido:

```
SignedData ::= SEQUENCE {
     version Version,
     digestAlgorithms DigestAlgorithmIdentifiers,
     contentInfo ContentInfo,
     certificates
        [0] IMPLICIT ExtendedCertificatesAndCertificates
          OPTIONAL,
     crls
       [1] IMPLICIT CertificateRevocationLists OPTIONAL,
     signerInfos SignerInfos }
```

Bouncy Castle tiene funciones para convertir el signedData a DER, así que sólo nos tenemos que preocupar de extraer los datos.

Para poder utilizar las funciones que hacen más cómodo el tratamiento del formato PKCS7 se ha adaptado parte de la librería Bouncy Castle a J2ME.

La clase org.bouncycastle.openssl.Certificado contiene una función toSignedData() que permite extraer un objeto de la clase SignedData con todos los datos de nuestra firma:

```
       public SignedData toSignedData() {
        SignedData signedData = null;

        if (isVoid() | firma == null) {
            return signedData;
        }

        TBSCertificateStructure tbs = estructura.getTBSCertificate();


        //Empezamos a construir el signed data:
        DERInteger _version = new DERInteger(1);


        ASN1EncodableVector algorithms = new ASN1EncodableVector();
        algorithms.add(new AlgorithmIdentifier(new
DERObjectIdentifier("1.3.14.3.2.26"), new DERNull()));
        ASN1Set _digestAlgorithms = new DERSet(algorithms);


        ContentInfo _contentInfo = new ContentInfo(
                PKCSObjectIdentifiers.data,
                null);


        ASN1Encodable[] certificates = new ASN1Encodable[1];
        certificates[0] = estructura;
```

```
        ASN1Set _certificates = new DERSet(certificates);


        ASN1Set _crls = null;

        X509Name x509Name = tbs.getIssuer();

      IssuerAndSerialNumber issuerAndSerialNumber = new
IssuerAndSerialNumber(x509Name, estructura.getSerialNumber().getValue());

        ASN1EncodableVector _signerInfos = new ASN1EncodableVector();
        _signerInfos.add(new SignerInfo(new DERInteger(1),
                issuerAndSerialNumber,
                new AlgorithmIdentifier(new
DERObjectIdentifier("1.3.14.3.2.26"), new DERNull()),
                null,
                new AlgorithmIdentifier(PKCSObjectIdentifiers.rsaEncryption,
new DERNull()),
                new DEROctetString(firma),
                null)
                );



        signedData = new SignedData(_version,
                _digestAlgorithms,
                _contentInfo,
                _certificates,
                _crls,
               new DERSet(_signerInfos));

        return signedData;
    }
```

A partir de este objeto, es fácil generar el pkcs7. El siguiente código aparecería en la clase principal:

```
        //Obtenemos los datos a firmar:
        byte[] cadena = aFirma.GenerarHash(id_sesion, id_doc);

        if(cadena != null)
        {

        certificado.firmar(cadena);

        SignedData signeddata = certificado.toSignedData();
        ContentInfo cont = new ContentInfo(PKCSObjectIdentifiers.signedData,
signeddata);

         cadena = cont.getDEREncoded();
         }
```

cadena contendría el array de bytes correspondientes a nuestro pkcs7. Este contenido será enviado en forma de DocumentoBean para su verificación por el servidor de @firma y posterior custodia en el servidor WebDAV.

9.3.4.6 GUI

J2ME no permite demasiada flexibilidad respecto a lo que la interfaz gráfica se refiere. No obstante, la navegación entre menús de la aplicación es la siguiente:

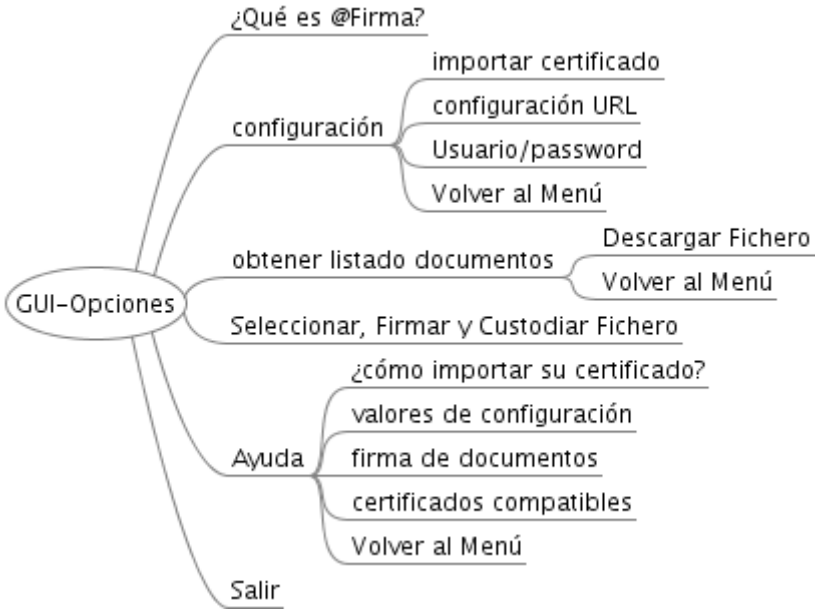


Figura 9.5. GUI-opciones

La funcionalidad principal reside en la clase Firmar.java, ya que implementa los métodos startApp(), pauseApp() y destroyApp() entre otros.

El método commandAction implementa el GUI de opciones anteriores, que ejecutará los comandos necesarios dependiendo de la acción señalada:

```
public void commandAction(Command command, Displayable displayable) {
        // write pre-action user code here
        if (displayable == Ayuda) {
            if (command == List.SELECT_COMMAND) {

                AyudaAction();

            }
        } else if (displayable == Bienvenida) {
            if (command == okCommand4) {

                switchToPreviousDisplayable();

            }
        } else if (displayable == CertificadosCompatibles) {
            if (command == okCommand13) {
```

```
            switchToPreviousDisplayable();

        }
    } else if (displayable == ComoimportarsuCertificado) {
        if (command == okCommand10) {

            switchToPreviousDisplayable();

        }
    } else if (displayable == Configuración) {
        if (command == List.SELECT_COMMAND) {

            ConfiguraciónAction();

        }
    } else if (displayable == Error) {
        if (command == okCommand15) {

            switchDisplayable(null, getMenuPrincipal());

        }
    } else if (displayable == Exito) {
        if (command == exitCommand) {

            switchDisplayable(null, getMenuPrincipal());

        }
    } else if (displayable == FirmadeDocumentos) {
        if (command == okCommand12) {

            switchToPreviousDisplayable();

        }
    } else if (displayable == MenuPrincipal) {
        if (command == List.SELECT_COMMAND) {

            MenuPrincipalAction();

        }
    } else if (displayable == ValoresdeConfiguracion) {
        if (command == okCommand11) {

            switchToPreviousDisplayable();

        }
    } else if (displayable == advertencia) {
        if (command == c) {

            procesarFirma();

        } else if (command == cancelCommand3) {

            switchDisplayable(null, getMenuPrincipal());

        }
    } else if (displayable == ayudaCertificado) {
        if (command == okCommand6) {

            switchToPreviousDisplayable();
```

```
        }
    } else if (displayable == ayudaFichero) {
        if (command == backCommand3) {

            switchToPreviousDisplayable();

        }
    } else if (displayable == buscaCertificado) {
        if (command == List.SELECT_COMMAND) {

            buscaCertificadoAction();

        } else if (command == cancelCommand) {

            switchDisplayable(null, getConfiguración());

        }
    } else if (displayable == buscaFichero) {
        if (command == List.SELECT_COMMAND) {

            buscaFicheroAction();

        } else if (command == cancelCommand2) {
            switchDisplayable(null, getMenuPrincipal());

        }
    } else if (displayable == elegirCertificado) {
        if (command == buscarCertificado2) {

            switchDisplayable(null, getBuscaCertificado());

        } else if (command == usarCertificado2) {

            switchDisplayable(null, getIntroduceClave());

        }
    } else if (displayable == faltaConfigurar) {
        if (command == okCommand14) {

            switchDisplayable(null, getUrlVerificacion());

        }
    } else if (displayable == introduceClave) {
        if (command == cancelCommand1) {

            switchDisplayable(null, getConfiguración());

        } else if (command == okCommand) {

            extraerKeyPair();

        }
    } else if (displayable == listaDocumentos) {
        if (command == List.SELECT_COMMAND) {

            listaDocumentosAction();

        } else if (command == backCommand1) {
```

```
                switchDisplayable(null, getMenuPrincipal());

            }
        } else if (displayable == login) {
            if (command == cancelCommand5) {

                switchDisplayable(null, getConfiguración());

            } else if (command == okCommand7) {

                guardarConfiguracion();
            }
        } else if (displayable == seleccionarCarpeta) {
            if (command == List.SELECT_COMMAND) {

                seleccionarCarpetaAction();

            }
        } else if (displayable == urlVerificacion) {
            if (command == cancelCommand4) {

                switchDisplayable(null, getConfiguración());

            } else if (command == okCommand2) {
                if(estado == 1)
                {
                    if(!url_verifica.getString().equals(""))
                    guardarConfiguracion();
                    else
                        switchDisplayable(null, getConfiguración());
                }
                else
                    switchDisplayable(null, getLogin());
            }
        }
        else if(displayable == PantallaEspera)
            if(command == okCommand19)
            {
                PantallaEspera.deleteAll();
                 PantallaEspera.append("Espere mientras se realiza la conexion.
La duracion de este proceso dependera de la velocidad de su conexion a
internet.");
                PantallaEspera.removeCommand(okCommand19);
                extraerDocumentos();

            }
            else if(command == okCommand20)
            {
                PantallaEspera.deleteAll();
                PantallaEspera.removeCommand(okCommand20);
                 PantallaEspera.append("Espere mientras se realiza la conexion.
La duracion de este proceso dependera de la velocidad de su conexion a internet
y del tamaño del fichero.");
                descargarDocumento();
            }
            else if(command == okCommand21)
            {
                PantallaEspera.deleteAll();
                PantallaEspera.removeCommand(okCommand21);
                 PantallaEspera.append("Espere mientras se realiza la conexion.
```

```
La duracion de este proceso dependera de la velocidad de su conexion a internet
y del tamaño del fichero.");
                registrarDocumento();
            }
    }
```

En el código anterior puede observarse que, dependiendo de la opción señalada por el usuario, se invocará un método u otro.

Así, si el usuario decide acceder a la configuración, se llamará a ConfiguracionAction; si desea descargar la lista de documentos se llamará al método listaDocumentosAction; si desea importar su certificado se llamará al método estraerKeyPair; si desea buscar un certificado ya instalado se llamará la método BuscarCertificadoAction(); si desea buscar alguna carpeta se llamará al método seleccionarCarpetaAction()...y así sucesivamente.

Un ejemplo del método que implementa las opciones que deben aparecer en el menú principal es el siguiente:

```
  public List getMenuPrincipal() {
        if (MenuPrincipal == null) {
            // write pre-init user code here
            MenuPrincipal = new List("Men\u00FA Principal", Choice.IMPLICIT);
            MenuPrincipal.append("\u00BFQu\u00E9 es @Firma?", null);
            MenuPrincipal.append("Configuraci\u00F3n", null);
            MenuPrincipal.append("Obtener Documentos", null);
            MenuPrincipal.append("Registrar Documento", null);
            MenuPrincipal.append("Firmar Documento", null);
            MenuPrincipal.append("Ayuda", null);
            MenuPrincipal.append("Salir", null);
            MenuPrincipal.setCommandListener(this);
           MenuPrincipal.setSelectedFlags(new boolean[] { true, false, false,
false, false, false, false });
            // write post-init user code here
        }
        return MenuPrincipal;
    }
```

Entre todas los métodos contenidos en la clase Firmar.java, además de los ya comentado, vamos a analizar algunos de los más importantes:

1. En caso de que el cliente selecciones la opción primera de *configuración*, será necesario introducir la Url del servidor Web así como el nombre de usuario y contraseña e importar el certificado. La función llamada es la siguiente:

```
 public void ConfiguraciónAction() {
        // enter pre-action user code here
                                            String      __selectedString      =
getConfiguración().getString(getConfiguración().getSelectedIndex());
        if (__selectedString != null) {
            if (__selectedString.equals("Importar Certificado")) {

                switchDisplayable(null, getElegirCertificado());
                elegirCertificado.setString(generarPeticionURL());
                if(ruta_certificado.equals(""))
```

```
                    getElegirCertificado().removeCommand(getUsarCertificado2())
;
                else
                    getElegirCertificado().addCommand(getUsarCertificado2());
            } else if (__selectedString.equals("Configuraci\u00F3n URL")) {

                switchDisplayable(null, getUrlVerificacion());
               url_verifica.setLabel(generarPeticionVerificacion());
            } else if (__selectedString.equals("Usuario y Password")) {

                switchDisplayable(null, getLogin());

                    } else if (__selectedString.equals("Volver al Men\u00FA
Principal")) {

                switchDisplayable(null, getMenuPrincipal());

            }
        }
        // enter post-action user code here
    }
```

En caso de configurar la Url, el método mira si ya existe alguna configuración anterior:

```
    private String generarPeticionVerificacion()
    {
        String cadena = "";
        try
        {
            String s = new String(url_verificacion, "UTF-8");
            if(s.equals(""))
                  cadena =  "No hay ninguna URL guardada. Tiene que especificar
la URL donde se encuentra el Servicio Web. \n Un ejemplo de URL sería
http://prometeo.utrera.org:8080/axis2/services";
                else
            {
                        cadena += "La url por defecto para verificar la firma
es: \n\n";
                cadena += s;
                 cadena += "\n\n Si no es la url correcta, proceda a elegir una
nueva URL.";
            }
        } catch (Exception ex)
        {
            cadena = "No hay URL por defecto para verificar la firma. Tiene que
seleccionar una nueva URL.";
        }
        finally
        {
            return cadena;
        }
    }
```

En caso de elegir importar el certificado, inmediatamente se generaría la estructura Certificado y se extraen las claves correspondientes:

```
 public void extraerKeyPair() {
        //Esta función saca el par de claves del certificado:
```

```
        //Si hay datos en los formularios, los saca
        extraerDatos();

        PasswordFinder pf = new Password(contraseña);
        InputStream is = null;
        Reader reader = null;
        javax.microedition.io.file.FileConnection fc = null;
        boolean resultado = true;

        try{
            //Sacamos el par de claves del certificado:
                 if(ruta_certificado.equals("") && buscaCertificado != null &&
estado == 1)
            {
                        //Si estamos en el proceso de configuración, hay que
seleccionar el certificado
                                                                    fileName    =
buscaCertificado.getString(buscaCertificado.getSelectedIndex());
                       ruta_certificado = "file://localhost/" + currDirName +
fileName;
            }

            //Abrimos el fichero del certificado
                         fc  =  (FileConnection)Connector.open(ruta_certificado,
Connector.READ);

            //Si es un fichero legible:
            if (fc.exists() && fc.canRead())
               is = fc.openInputStream();
            else throw new IOException("El certificado no era legible");

            //Leemos el certificado
            reader = new InputStreamReader(is);

            //PEMReader lee e interpreta el certificado
            //para extraer el par de claves RSA
            PEMReader pemreader = new PEMReader(reader, pf);
            Object obj;
            KeyPair k = null;
            while((obj = pemreader.readObject()) != null)
            {

                if(obj instanceof CertificadoGenerico)
                    certificado = (CertificadoGenerico)obj;
                else if(obj instanceof KeyPair)
                    k = (KeyPair)obj;
                else
                    throw new Exception("El pemreader hace cosas raras.");
            }

            //Si no pudo extraer los datos, a pesar de no fallar el PEMReader:
            if(certificado.isVoid())
                throw new Exception("No se pudo leer el certificado.");


            //Si el PEMReader no ha pillado ni esto:
            if(certificado != null)
                certificado.setKeyPair(k);
        }
```

```
        catch (Exception a)
        {
            resultado = false;
        }
        finally
        {
            try
            {
                is.close();
                fc.close();
                reader.close();
            }
            catch(Throwable e)
            {

            }
        }
        if (resultado) {
            guardarConfiguracion();
            if(estado != 1)
            {
                //Pasamos a buscar el fichero a firmar
                fileName = "";
                currDirName = "/";
                buscaFichero = null;
                       if(url_verificacion.length == 0 || pass.length == 0 ||
username.length == 0)
                    switchDisplayable(null, getFaltaConfigurar());
                else
                    switchDisplayable(null, getBuscaFichero());
            }

        } else {

            switchDisplayable(null, getBuscaCertificado());

        }
        // enter post-if user code here
    }
```

2. En caso de escoger la opción segunda, se descargaría la lista de documentos pendientes de ser firmados almacenados en el servidor WebDAV:

```
 public void extraerDocumentos() {
        //Esta función extrae todos los documentos del usuario de @Firma
        String user = "";
        String contras = "";


        try
        {
            if(username != null)
                user = new String(username, "UTF-8");
            if(pass != null)
                contras = new String(pass, "UTF-8");
        }
```

```
        catch(Exception e)
        {
            if(username != null)
                user = new String(username);
            if(pass != null)
                contras = new String(pass);
        }
        finally
        {
            //Mostramos una pantalla de espera:


            //Primero nos autenticamos:
            try
            {
                aFirma = new aFirmaImpl(new String(url_verificacion));
            }
            catch(Exception e)
            {
                    cadenaError += " No se pudo autentificar contra el sistema
@Firma. Compruebe la url. ";
                switchDisplayable(null, getError());
            }
            String id_sesion = aFirma.login(user, contras);
            boolean resultado = id_sesion != null;

            if (resultado) {

                switchDisplayable(null, getListaDocumentos());
                Fichero[] fs = aFirma.listDocuments(id_sesion);
                lista_docs.clear();
                if(fs != null)
                    for(int con = 0; con < fs.length; con++)
                        lista_docs.add(fs[con]);

                listaDocumentos.deleteAll();
                //Añadimos el contenido de los documentos a la lista:
                for(int i = 0; i < lista_docs.size(); i++)
                     listaDocumentos.append(((Fichero)lista_docs.get(i)).getName
(), null);
            } else {
                    cadenaError += " No se pudo autentificar contra el sistema
@Firma. Compruebe su usuario y contraseña. ";
                switchDisplayable(null, getError());
            }
        }
    }
```

donde aFirma representa una interfaz donde se encuentran implementados los métodos que invocan a los servicios web y que reproduciremos más tarde.

A su vez, cuando el usuario decide descargar un documento en concreto de la lista, la función invocada es la siguiente:

```
public void descargarDocumento() {
        //Dado un id de documento, lo descarga de @Firma:
        boolean resultado = true;
```

```
        String user = "";
        String contras = "";
       try
       {
            aFirma = new aFirmaImpl(new String(url_verificacion));
       }
       catch(Exception e)
       {
           cadenaError += "Compruebe la url de verificacion de la firma.";
           switchDisplayable(null, getError());
       }
        if(username != null && pass != null)
        {
            try
            {
                user = new String(username, "UTF-8");
                contras = new String(pass, "UTF-8");
            }
            catch(Exception e)
            {
                user = new String(username);
                contras = new String(pass);
            }
        }
        //Primero nos autenticamos:
        String id_sesion = aFirma.login(user, contras);
        if(id_sesion == null)
        {
             cadenaError = " No se pudo autentificar contra el sistema @Firma.
Compruebe su nombre de usuario y contraseña. ";
            resultado = false;
        }
        else
        {
            String id_doc = null;
            Fichero f = null;

            //Obtenemos el nombre del fichero seleccionado:
                                                          String     s     =
listaDocumentos.getString(listaDocumentos.getSelectedIndex());

            //Buscamos el documento en la lista de documentos:
            for(int i = 0; i < lista_docs.size(); i++)
            {
                f = (Fichero) lista_docs.get(i);
                if(f.getName().equals(s))
                    id_doc = f.getID();
            }

            if(id_doc == null)
            {
                    resultado = false;
                     cadenaError = " Hubo un error con el fichero seleccionado y
no se pudo descargar. ";
            }
            //y nos bajamos el documento:
            Documento[] docs;
            docs = aFirma.getDocument(id_sesion, id_doc);

            if(docs.length >= 1)
```

```
                d = docs[0];

            if(d == null)
            {
                    resultado = false;
                      cadenaError = " No se pudo descargar el documento a pesar
de haber podido autentificarse en @Firma. ";
            }
            else
            {

                byte[] contenido = Base64.decode(d.getContent());

                aFirma.registraDocumentoRMS(contenido, d.getID());



                fileName="";

                if(!writeFile(contenido, d.getName()))
                {
                       cadenaError ="No se pudo descargar el fichero, compruebe
que se ha seleccionado una carpeta con permiso de escritura.";
                    resultado = false;
                }
            }
        }

        if (resultado) {

            switchDisplayable(null, getExito());

               String s = "El documento se ha guardado en " + currDirName + ".
Antes de firmarlo, sería conveniente abrirlo con alguna aplicación para
comprobar que es realmente el fichero que quiere firmar.";
            Exito.setMaxSize(s.length());
            Exito.setString(s);
            Exito.setTitle(d.getName());
        } else {

            switchDisplayable(null, getError());

        }

    }
```

3. Ya por último, una vez seleccionado y visualizado el documento que se desea firmar, se invoca a la función procesarFirma():

```
public void procesarFirma() {
        //Esta función calcula la firma y la envía a @Firma

       boolean resultado = true;

        //Bloqueamos el móvil:
        advertencia.removeCommand(c);
```

```
            advertencia.setString("Espere mientras se firma el documento y se
custodia en @Firma... Este proceso accederá a internet para la custodia.");
        cadenaError = "";

        String user = "";
        String contras = "";
        try
        {
            user = new String(username, "UTF-8");
            contras = new String(pass, "UTF-8");
        }
        catch(Exception e)
        {
            user = new String(username);
            contras = new String(pass);
        }
        finally
        {
            try{
                     aFirma = new prometeo.aFirma.Implementacion.aFirmaImpl(new
String(url_verificacion));
                //Primero nos autenticamos:
                String id_sesion = aFirma.login(user, contras);
                if(id_sesion == null)
                {
                     cadenaError += " No se pudo autentificar contra el sistema
@Firma. ";
                    resultado = false;
                }
                else
                {
                    //y registramos el documento
                              String id_doc = aFirma.getIdDocument(id_sesion,
"file://localhost/" + currDirName + fileName);
                    if(id_doc == null)
                    {
                        resultado = false;
                           cadenaError += " No se pudo obtener el identificador
del documento";
                            cadenaError += " Recuerde que para poder firmar un
documento, dicho documento debe estar registrado en el servidor. Si esta seguro
de que el documento ya esta registrado, puede a descargarlo otra vez. ";
                    }
                    else
                    {
                        //Obtenemos los datos a firmar:
                        byte[] cadena = aFirma.GenerarHash(id_sesion, id_doc);

                        if(cadena != null)
                        {
                            certificado.firmar(cadena);

                            //por último, custodiamos el documento firmado:
                            SignedData signeddata = certificado.toSignedData();
                                                    ContentInfo cont = new
ContentInfo(PKCSObjectIdentifiers.signedData, signeddata);
                            cadena = cont.getDEREncoded();

                            Base64Coder coder=new Base64Coder();
                            byte[] cadena64=coder.encodeBase64(cadena);
```

```
                            writeFile(cadena, ".p7");
                            try {
                                d = new DocumentoBean(cadena, fileName, id_doc,
Long.toString(System.currentTimeMillis()));
                            } catch (Throwable ex) {
                                ex.printStackTrace();
                            }

                                  if(aFirma.custodiarDocumento(id_sesion, d) ==
null)
                            {
                                       cadenaError += "No se pudo custodiar el
documento en el sistema, a pesar de haber podido autentificarse.";
                                    cadenaError += " Recuerde que para poder firmar
un documento, dicho documento debe estar registrado en el servidor. Si esta
seguro de que el documento ya esta registrado, puede a descargarlo otra vez. ";
                                    resultado = false;
                            }
                        }
                        else
                        {
                            resultado = false;
                                  cadenaError += "No se pudo obtener el hash a
firmar.";
                             cadenaError += " Recuerde que para poder firmar un
documento, dicho documento debe estar registrado en el servidor. Si esta seguro
de que el documento ya esta registrado, puede a descargarlo otra vez. ";
                        }
                    }
                }
            }
            catch(Throwable e)
            {
                resultado = false;
                cadenaError += e.getClass().getName() + "::" + e.getMessage();
            }
            finally
            {
                if (resultado) {
                String s = "Se ha firmado y custodiado correctamente el fichero
" + fileName + ".";
               switchDisplayable(null, getExito());
                Exito.setMaxSize(s.length());
                Exito.setTitle("Operación terminada");
                Exito.setString(s);
                } else {
                    switchDisplayable(null, getError());
                Error.setMaxSize(getCadenaError().length());
                Error.setString(getCadenaError());
                }
                //Volvemos a poner el botón de avanzar:
                advertencia.addCommand(c);
            }
        }
    }
```

Este último método, devuelve el fichero .p7 al servidor WebDAV una vez verificada la validez e integridad de la firma.

Por último, queda mostrar la invocación de los servicios web correspondientes. Todos los métodos que consumen los servicios web se encuentran desplegados en la clase aFirmaImpl.java:

```
public class aFirmaImpl implements prometeo.aFirma.aFirma
{
    private AplicacionFirma_Stub aFirma;
    private Login_Stub login;
    private listaDocumentos_Stub listaDoc;
    private ObtieneDocumento_Stub obtieneDoc;
    private GenerarHash_Stub generarDatos;
    private CustodiarDocumento_Stub custodiarDoc;

    private byte[] documento;

    public aFirmaImpl(String direccion)
    {
        super();
        aFirma = new AplicacionFirma_Stub(direccion);
        login = new Login_Stub(direccion);
        listaDoc = new listaDocumentos_Stub(direccion);
        obtieneDoc = new ObtieneDocumento_Stub(direccion);
        generarDatos = new GenerarHash_Stub(direccion);
        custodiarDoc = new CustodiarDocumento_Stub(direccion);
    }
```

Las declaraciones anteriores permiten generar el cliente consumidor de los servicios web mediante el stub generado automáticamente a partir del .wsdl de cada servicio desplegado en el servidor Web. El stub es un fichero que permite abstraer al cliente de la ejecución del servicio web, de forma que sólo tenga que preocuparse por la llamada.

NetBEans permite generar un cliente J2ME Web Services (JSR-172) automáticamente, del siguiente modo:

1. Click derecho sobre el proyecto. New --> Java ME Web Service
2. Aparecerá la imagen siguiente, donde hay que indicar la URL del wsdl del servicio web del que queramos generar el cliente correspondiente, así como el nombre que va a tener el stub generado y el paquete donde se va a guardar:

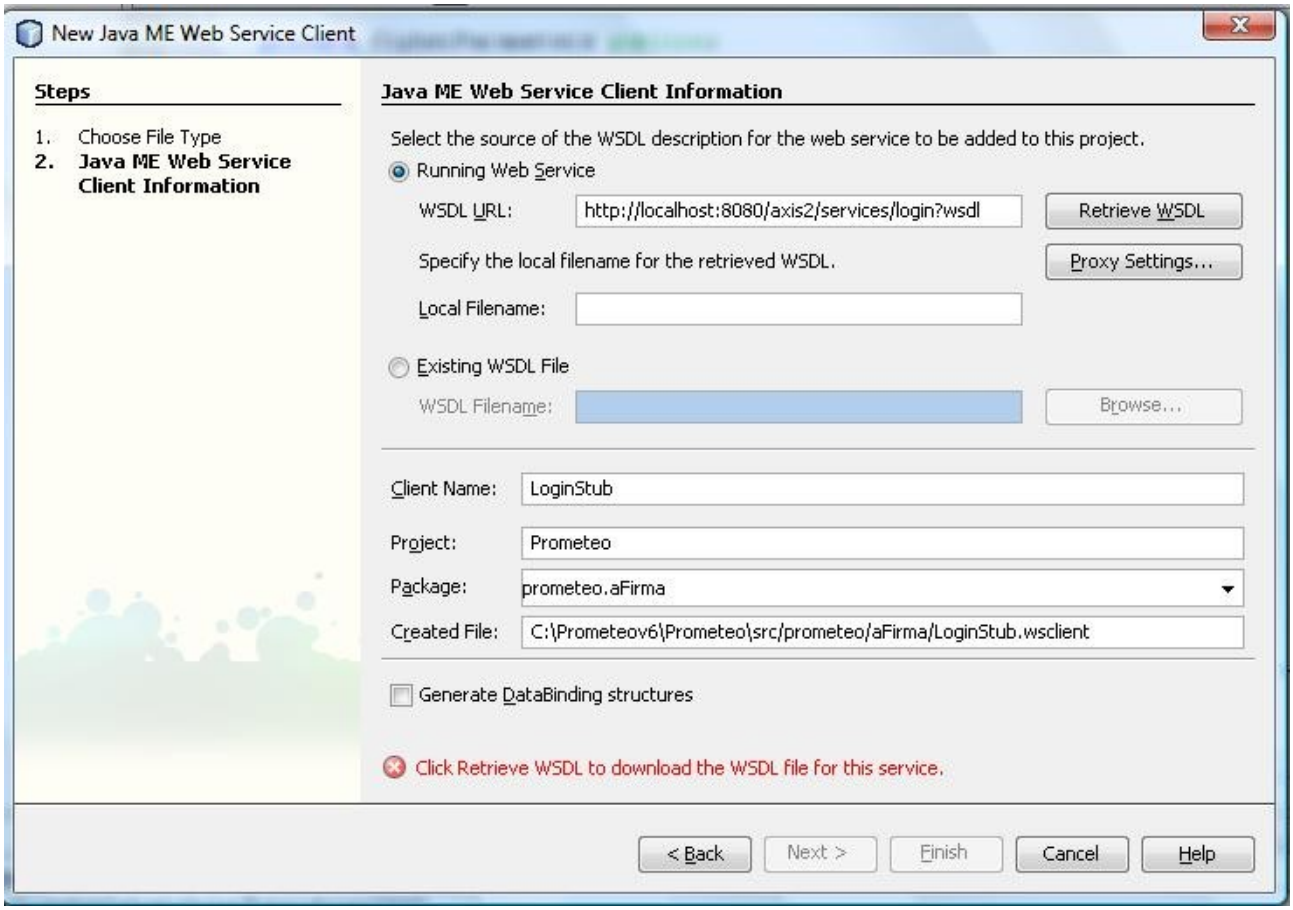


Figura 9.6. Generación de stub a partir del .wsdl

Un ejemplo de stub generado es el siguiente, del servicio web que permite la autenticación:

```
package prometeo.aFirma.Implementacion;

import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.microedition.xml.rpc.Operation;
import javax.microedition.xml.rpc.Type;
import javax.microedition.xml.rpc.ComplexType;
import javax.microedition.xml.rpc.Element;

public class Login_Stub implements Login, javax.xml.rpc.Stub {

    private String[] _propertyNames;
    private Object[] _propertyValues;

    public Login_Stub(String direccion) {
        _propertyNames = new String[] { ENDPOINT_ADDRESS_PROPERTY };
        _propertyValues = new Object[] { direccion  + "/Login" };
    }



    public void setAddress(String address)
```

```
    {
        _setProperty(ENDPOINT_ADDRESS_PROPERTY, address);
    }

    public void _setProperty( String name, Object value ) {
        int size = _propertyNames.length;
        for (int i = 0; i < size; ++i) {
            if( _propertyNames[i].equals( name )) {
                _propertyValues[i] = value;
                return;
            }
        }
        String[] newPropNames = new String[size + 1];
        System.arraycopy(_propertyNames, 0, newPropNames, 0, size);
        _propertyNames = newPropNames;
        Object[] newPropValues = new Object[size + 1];
        System.arraycopy(_propertyValues, 0, newPropValues, 0, size);
        _propertyValues = newPropValues;

        _propertyNames[size] = name;
        _propertyValues[size] = value;
    }

    public Object _getProperty(String name) {
        for (int i = 0; i < _propertyNames.length; ++i) {
            if (_propertyNames[i].equals(name)) {
                return _propertyValues[i];
            }
        }
                            if    (ENDPOINT_ADDRESS_PROPERTY.equals(name)    ||
USERNAME_PROPERTY.equals(name) || PASSWORD_PROPERTY.equals(name)) {
            return null;
        }
        if (SESSION_MAINTAIN_PROPERTY.equals(name)) {
            return new Boolean(false);
        }
        throw new JAXRPCException("Stub does not recognize property: " + name);
    }

    protected void _prepOperation(Operation op) {
        for (int i = 0; i < _propertyNames.length; ++i) {
            op.setProperty(_propertyNames[i], _propertyValues[i].toString());
        }
    }

          public   String   login(String   username,   String   password)   throws
java.rmi.RemoteException {
        Object inputObject[] = new Object[] {
            username,
            password
        };

               Operation  op  =  Operation.newInstance(  _qname_operation_login,
_type_login, _type_loginResponse );
        _prepOperation( op );
        op.setProperty( Operation.SOAPACTION_URI_PROPERTY, "urn:login" );
        Object resultObj;
        try {
            resultObj = op.invoke( inputObject );
        } catch( JAXRPCException e ) {
```

```
            Throwable cause = e.getLinkedCause();
            if( cause instanceof java.rmi.RemoteException ) {
                throw (java.rmi.RemoteException) cause;
            }
            throw e;
        }

        return (String )((Object[])resultObj)[0];
    }

            protected static final QName _qname_operation_login = new
QName( "http://client.dav.skunk.org", "login" );
             protected static final QName _qname_loginResponse = new
QName( "http://client.dav.skunk.org", "loginResponse" );
                protected static final QName _qname_login = new
QName( "http://client.dav.skunk.org", "login" );
    protected static final Element _type_loginResponse;
    protected static final Element _type_login;

    static {
                    _type_loginResponse = new Element( _qname_loginResponse,
_complexType( new Element[] {
              new Element( new QName( "http://client.dav.skunk.org", "return" ),
Type.STRING, 0, 1, true )}), 1, 1, false );
        _type_login = new Element( _qname_login, _complexType( new Element[] {
                        new Element( new QName( "http://client.dav.skunk.org",
"username" ), Type.STRING, 0, 1, true ),
                        new Element( new QName( "http://client.dav.skunk.org",
"password" ), Type.STRING, 0, 1, true )}), 1, 1, false );
    }

    private static ComplexType _complexType( Element[] elements ) {
        ComplexType result = new ComplexType();
        result.elements = elements;
        return result;
    }
}
```

En el apartado siguiente, veremos cómo se genera NetBeans genera el stub a partir del archivo .wsdl automáticamente.

9.3.5 Simulación del cliente

Para simular la aplicación, se ha utilizado el programa NetBeans IDE 6.0.1. El móvil real será simulado a partir del emulador Sun Java Wireless Toolkit 2.5.2 for CDLC. Una vez instalado (Véase capítulo Guía de instalación), se creará un nuevo proyecto al que llamaremos *Prometeo* por especificaciones externas.

Durante la creación del nuevo proyecto, se pedirá que se indique la configuración de los perfiles. Seleccionaremos CLDC-1.0 y MIDP-2.0. Posteriormente, se añaden los paquetes y clases con el código fuente.

La estructura del proyecto generada será la siguiente:

Los paquetes que contienen las clases correspondientes al paquete Java implementado en J2ME:

- javafake.io
- javafake.math
- javafake.security
- javafake.util

Los paquetes que contienen las clases correspondientes a la librería criptográfica Bouncy Castle adaptada a J2ME:

- org.bouncycastle.asn1
- org.bouncycastle.asn1.oiw
- org.bouncycastle.asn1.pkcs
- org.bouncycastle.asn1.x509
- org.bouncycastle.crypto
- org.bouncycastle.crypto.digests
- org.bouncycastle.crypto.encodings
- org.bouncycastle.crypto.engines
- org.bouncycastle.crypto.generator
- org.bouncycastle.crypto.io
- org.bouncycastle.crypto.macs
- org.bouncycastle.crypto.modes
- org.bouncycastle.crypto.paddings
- org.bouncycastle.crypto.params
- org.bouncycastle.crypto.prng
- org.bouncycastle.crypto.signers
- org.bouncycastle.openssl
- org.bouncycastle.util
- org.bouncycastle.util.encoders

Los paquetes que contienen las clases correspondientes a la implementación de los servicios web y la clase principal que contiene la implementación del MIDlet en sí:

- prometeo.aFirma
- prometeo.aFirma.Implementacion
- prometeo.examples

Para poder empezar a simular el cliente, es necesario tener un certificado original expedido por alguna autoridad de confianza y convertirlo a formato PEM. Tal y como se explicó en capítulos anteriores, esto es necesario para aportar simplicidad a nuestra aplicación y para adaptarlo a las clases de la librería Bouncy Castle en J2ME.

Para convertir un certificado cualquiera en formato .PFX a formato .PEM hay que seguir los pasos siguientes:

Descargar e instalar el programa Win32 OpenSSL desde la URL: http://gnuwin32.sourceforge.net/packages/openssl.htm.

Una vez instalado, desde la línea de comandos, dirigirnos hasta el directorio donde se encuentra

instalado. Por defecto: C:\Archivos de Programas\GnuWin32\bin.

Una vez dentro del directorio, teclear la siguiente orden de comando:

```
openssl pkcs12 -in c:\certificado.pfx -out c:\certificado.pem -nodes
```

suponiendo que nuestro certificado se encuentra en C:\.

Una vez convertido que tenemos nuestro certificado en formato .PEM, estamos en disposición de ejecutar la aplicación.

Para ejecutar la aplicación, basta con hacer click derecho sobre el nombre del proyecto y posteriormente en la opción *Run*. Inmediatamente después, aparecerá el emulador del dispositivo móvil.

Tras ejecutarse la aplicación, es necesario copiar al sistema de ficheros del móvil el certificado .PEM. Por defecto, la ruta es:

```
C:\Users\Elena\j2mewtk\2.5.2\appdb\temp.DefaultColorPhone131\filesystem\root1
```

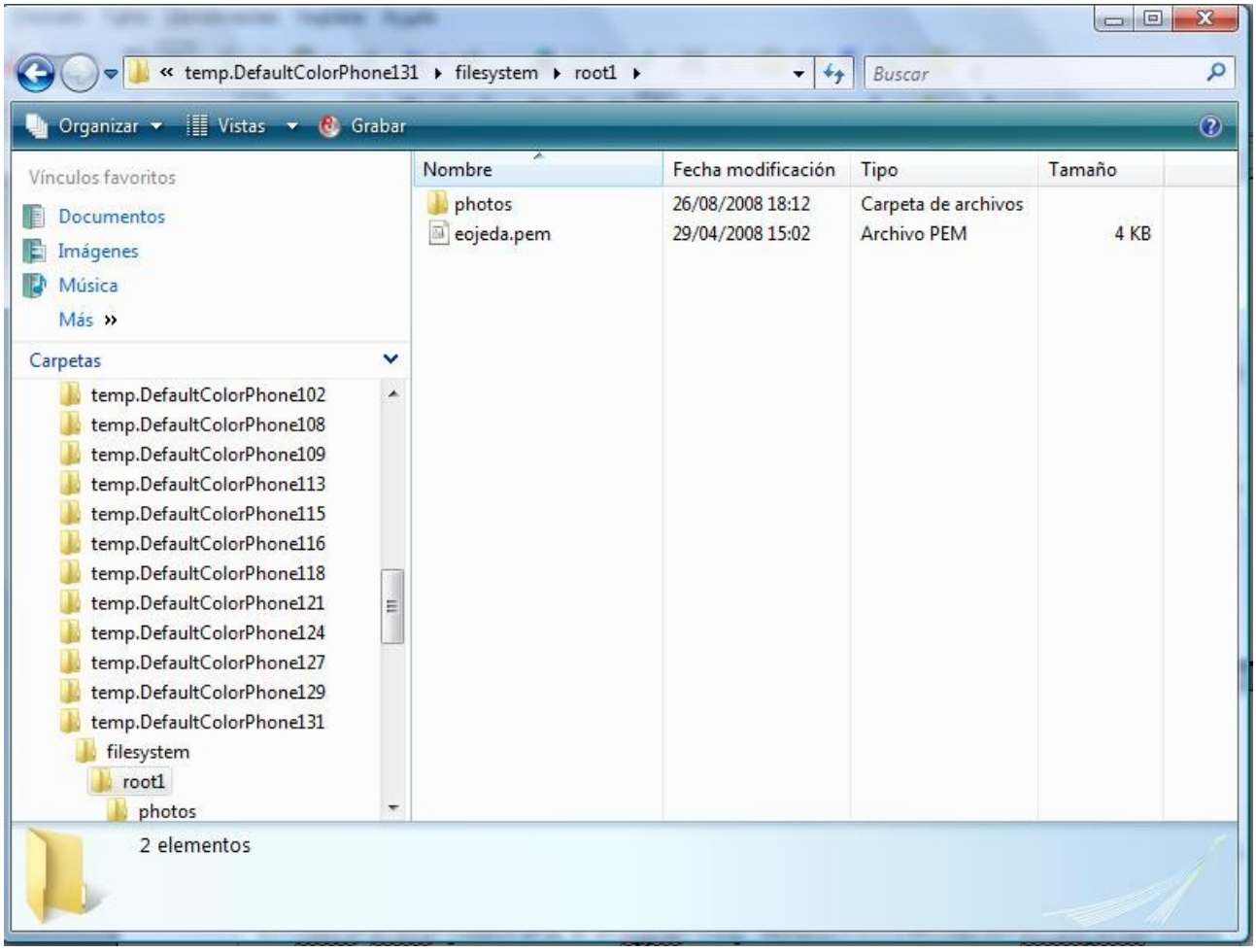


Figura 9.7. Sistema de ficheros del dispositivo móvil

Una vez que tenemos copiado el certificado al sistema de directorios del dispositivo móvil, podemos lanzar la aplicación:

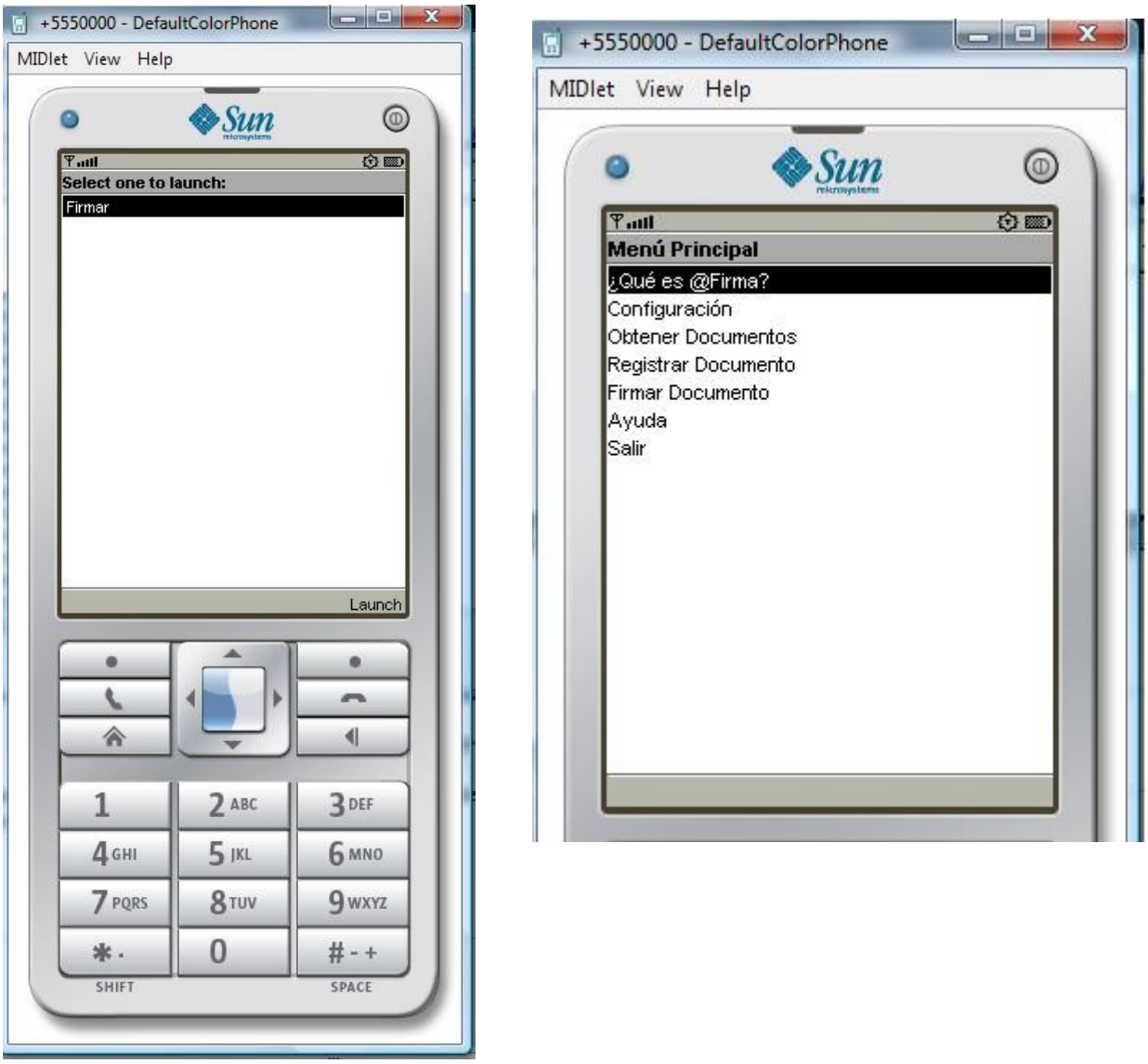


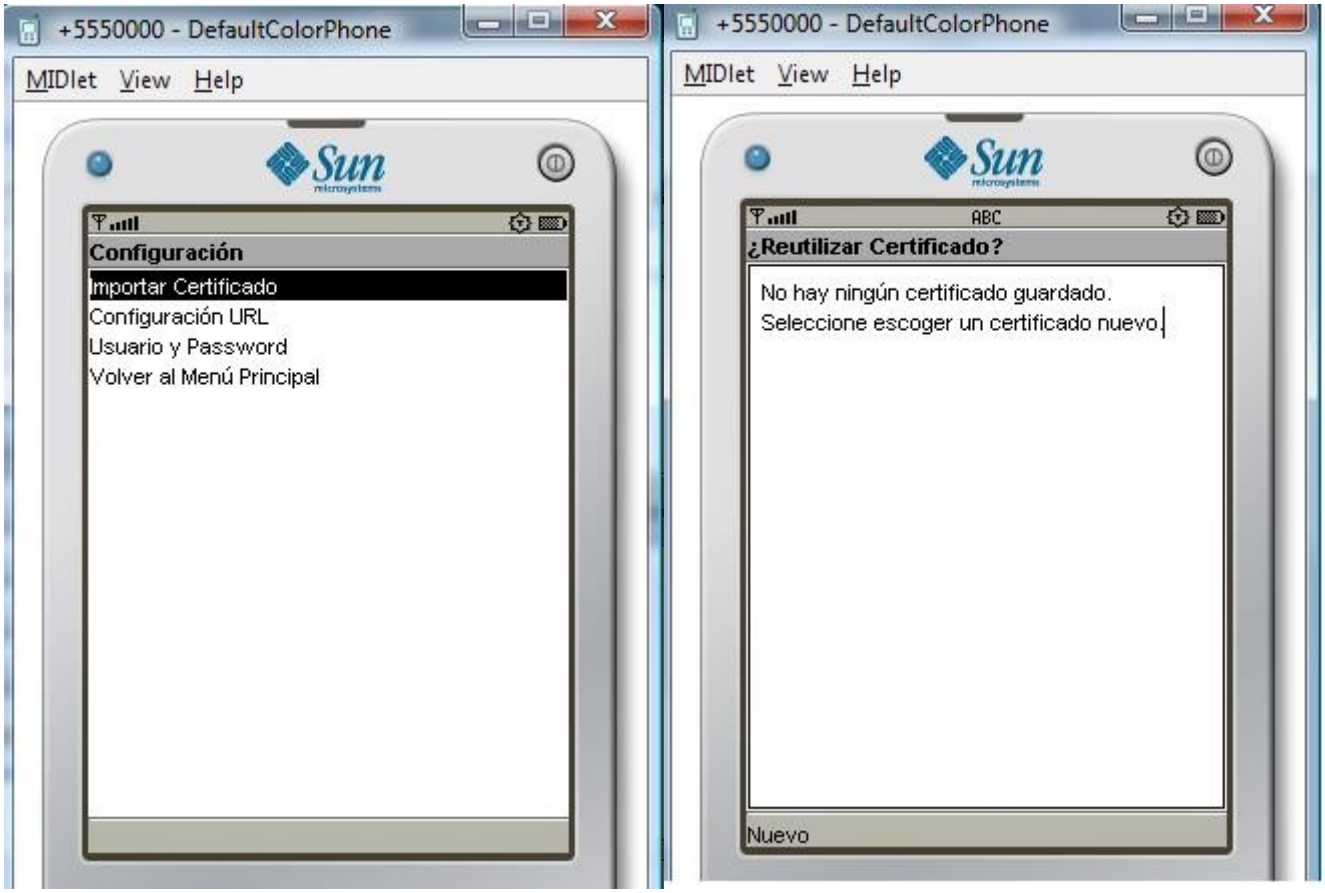
Figura 9.8. Simulación de la aplicación. Pantalla inicial

A continuación se mostrará la secuencia de imágenes que mostrarán las sucesivas opciones:

1. Configuración. En primer lugar, es necesario introducir los parámetros de configuración necesarios para que la aplicación funcione correctamente. Dichos parámetros son:

   - Importación del certificado desde el sistema de archivos
   - URL del servidor donde se encuentran desplegados los servicios web
   - Nombre de usuario y contraseña necesarios para acceder al servidor WebDAV

   La secuencia que reproduce todos los pasos anteriores es la siguiente:


+5550000 - DefaultColorPhone
MIDlet View Help
Configuración
Importar Certificado
Configuración URL
Usuario y Password
Volver al Menú Principal
+5550000 - DefaultColorPhone
MIDlet View Help
ABC
¿Reutilizar Certificado?
No hay ningún certificado guardado.
Seleccione escoger un certificado nuevo.
Nuevo


+5550000 - DefaultColorPhone
MIDlet View Help
ABC
Selecciona el Certificado
:: AYUDA - Búsqueda del Certificado ::
root1/
Cancel


+5550000 - DefaultColorPhone
MIDlet View Help
ABC
Selecciona el Certificado
:: AYUDA - Búsqueda del Certificado ::
..
eojeda.pem
photos/
Cancel


+5550000 - DefaultColorPhone
MIDlet View Help
ABC
Contraseña del Certificado
Contraseña:
Introduce la Contraseña del Certificado para comprobar que es válido. La clave que debe introducir aquí es la clave que le debieron de dar junto con el certificado. Si no está seguro del valor que debe introducir aquí, consulte con su proveedor del certificado. El certificado debe estar en formato PEM para poder ser utilizado.
Cancel
Continuar


+5550000 - DefaultColorPhone
+5550000 - DefaultColorPhone
MIDlet View Help
ABC
URL Verificacion
No hay URL por defecto para verificar la firma. Tiene que seleccionar una nueva URL.
Cancel
Continuar


+5550000 - DefaultColorPhone
MIDlet View Help
ABC
URL Verificacion
No hay URL por defecto para verificar la firma. Tiene que seleccionar una nueva URL.
http://localhost:8080/axis2/services
Cancel
Continuar

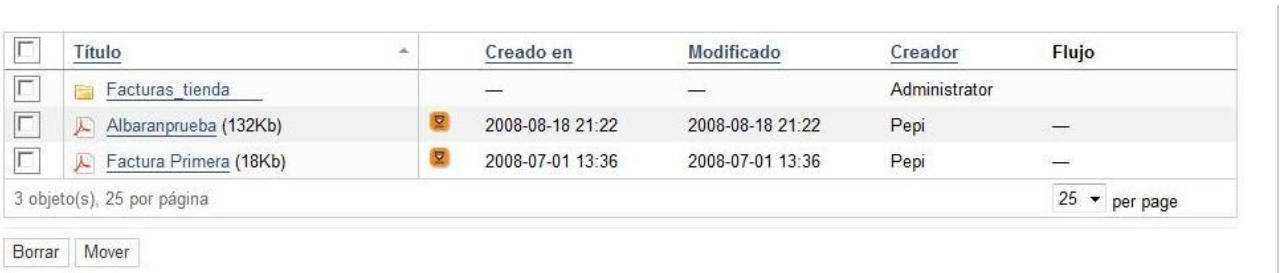

+5550000 - DefaultColorPhone
MIDlet View Help
ABC
Configuración
Importar Certificado
Configuración URL
Usuario y Password
Volver al Menú Principal


+5550000 - DefaultColorPhone
MIDlet View Help
ABC
acceso a @Firma
usuario:
contraseña:
Cancel
Continuar


+5550000 - DefaultColorPhone
MIDlet View Help
ABC
acceso a @Firma
usuario:
pepi@naturpeka
contraseña:
****
Cancel
Continuar

Hasta aquí, toda la parte correspondiente a la implementación de la configuración. A partir de ahora, comenzarán a utilizarse los servicios web correspondientes en cada llamada.

2. Obtener Documentos. Esta llamada permite obtener los documentos pendientes de ser firmados almacenados en el servidor WebDAV correspondientes a ese usuario en concreto. Extrae los documentos .pdf que se encuentran en carpetas y subcarpetas. Un ejemplo de los documentos almacenados en el servidor WebDAV correspondiente a ese usuario son los siguientes:



y dentro de la carpeta Facturas_Tienda, aparecerán otros documentos:

Título	Creado en	Modificado	Creador	Flujo
albaran de la factura f-1.xml (132Kb)	2008-02-04 17:12	2008-02-04 17:12	Pepi	—
documento pdf de la factura f-1.xml (132Kb)	2008-05-08 13:04	2008-05-08 13:04	Pepi	—
documento pdf de la factura f-5.xml (132Kb)	2008-07-17 08:15	2008-07-17 08:15	Pepi	—
f-1.xml (3Kb)	2008-02-04 17:07	2008-02-04 17:07	Administrator	—
f-2.xml (3Kb)	2008-02-21 09:12	2008-02-21 09:12	Administrator	—
f-3.xml (4Kb)	2008-03-29 17:36	2008-03-29 17:36	Administrator	—
f-4.xml (5Kb)	2008-03-29 17:41	2008-03-29 17:41	Administrator	—
f-5.xml (3Kb)	2008-07-16 14:51	2008-07-16 14:51	Administrator	—
f-6.xml (3Kb)	2008-07-22 10:57	2008-07-22 10:57	Administrator	—
f-7.xml (3Kb)	2008-08-14 10:23	2008-08-14 10:23	Administrator	—
Factura de prueba (18Kb)	2008-07-20 18:31	2008-07-20 18:31	Pepi	—

11 objeto(s), 25 por página — 25 per page

Borrar Mover

El cliente móvil deberá devolver, de todos los documentos anteriores, aquéllos que tengan formato .pdf. La secuencia en imágenes del simulador será la siguiente:

A continuación se muestra la lista de documentos .pdf descargados correspondientes a ese usuario. Una vez seleccionado el documento que se desea firmar, es necesario indicar donde se desea guardar. Seleccionaremos dentro de la ruta de directorios del sistema de ficheros del dispositivo móvil, la carpeta root1, donde mismo se encuentra almacenado el certificado.

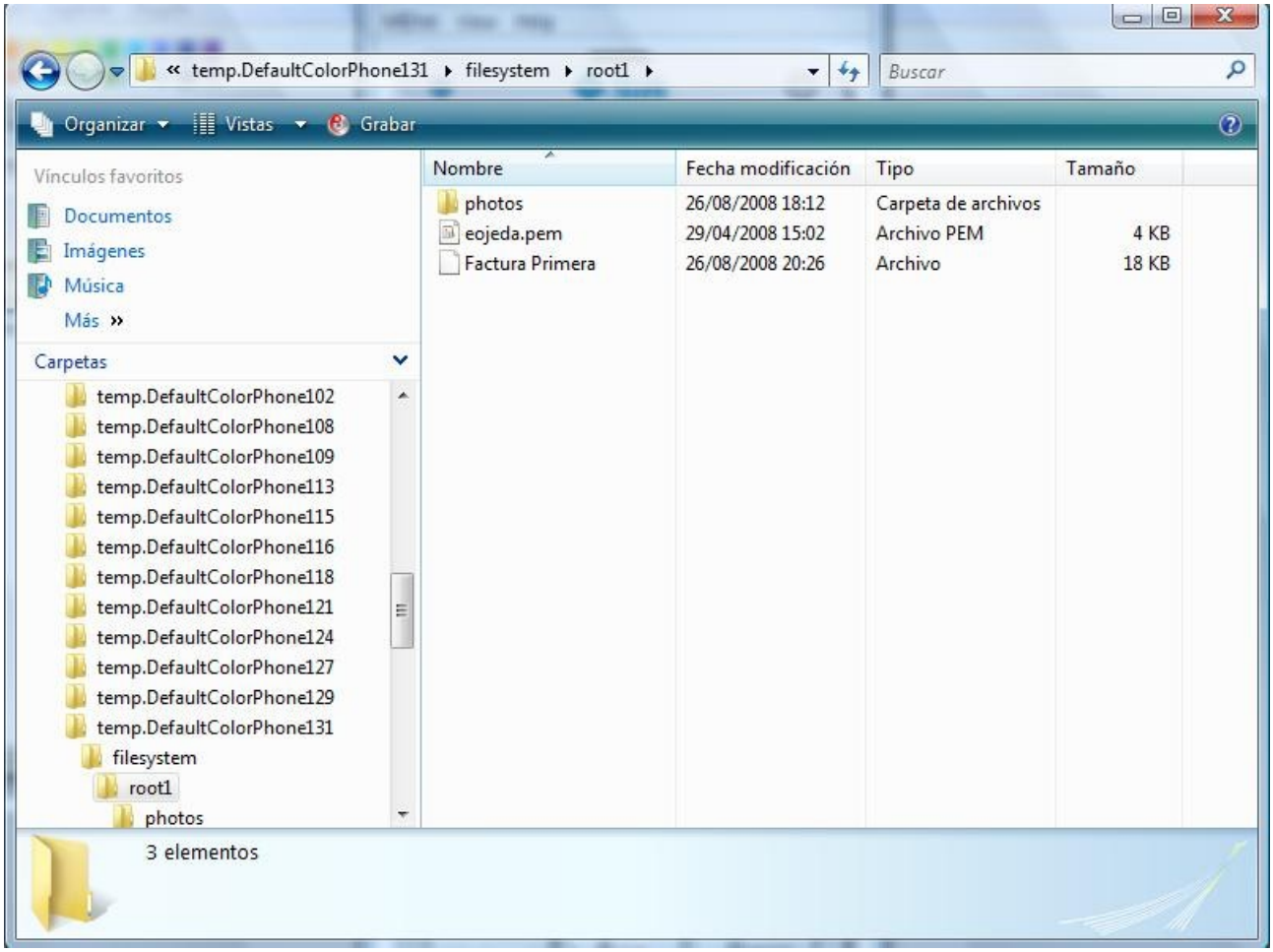

+5550000 - DefaultColorPhone
MIDlet View Help
Selecciona para descargar
Factura Primera
Albaranprueba
Factura de prueba
albaran de la factura f-1.xml
documento pdf de la factura f-1.xml
documento pdf de la factura f-5.xml
Atrás


+5550000 - DefaultColorPhone
MIDlet View Help
list
:: Descargar aqui ::
root1/

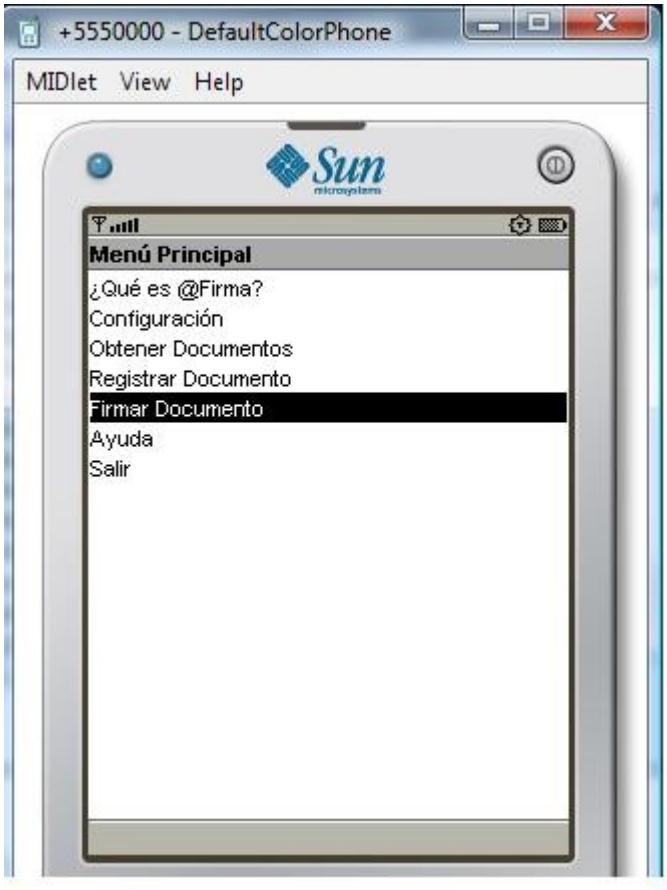

+5550000 - DefaultColorPhone
MIDlet View Help
list
:: Descargar aqui ::
root1/


+5550000 - DefaultColorPhone
MIDlet View Help
list
:: Descargar aqui ::
..
photos/

Una vez descargado el documento, aparecerá almacenado el sistema de directorios del móvil. El siguiente paso, corresponde a la visualización del documento por parte del usuario antes de ser firmado. No obstante, ese paso corresponde al usuario, no se encuentra implementado explicitamente en la aplicación, puesto que las aplicaciones para leer documentos .pdf en sistemas móviles son muy diversas y no es estándar. Por tanto, al usuario, antes de procesar la firma del documento, le corresponde la visualización del mismo para comprobar que efectivamente se trata de lo que desea firmar.



3. Firmar Documento. El siguiente paso tras descargar el documento seleccionado, consiste en realizar la firma. Cabe destacar que tras seleccionar el documento, hay que escoger el certificado con el que se desea firmar y volver a introducir la contraseña.


+5550000 - DefaultColorPhone
MIDlet View Help
Menú Principal
¿Qué es @Firma?
Configuración
Obtener Documentos
Registrar Documento
Firmar Documento
Ayuda
Salir


+5550000 - DefaultColorPhone
MIDlet View Help
ABC
¿Reutilizar Certificado?
La última vez se firmó un documento en este móvil se utilizó el siguiente certificado:
file:/localhost/root1/eojeda.pem
¿Se debe reutilizar el certificado? Si elige reutilizarlo y este certificado no está disponible, se le ofrecerá la pantalla de búsqueda para encontrar uno nuevo
Nuevo
Usar


+5550000 - DefaultColorPhone
MIDlet View Help
ABC
Contraseña del Certificado
Contraseña:
Introduce la Contraseña del Certificado para comprobar que es válido. La clave que debe introducir aquí es la clave que le debieron de dar junto con el certificado. Si no está seguro del valor que debe introducir aquí, consulte con su proveedor del certificado. El certificado debe estar en formato PEM para poder ser utilizado.
Cancel
Continuar


+5550000 - DefaultColorPhone
MIDlet View Help
ABC
Contraseña del Certificado
Contraseña:
****
Introduce la Contraseña del Certificado para comprobar que es válido. La clave que debe introducir aquí es la clave que le debieron de dar junto con el certificado. Si no está seguro del valor que debe introducir aquí, consulte con su proveedor del certificado. El certificado debe estar en formato PEM para poder ser utilizado.
Cancel
Continuar


+5550000 - DefaultColorPhone
MIDlet View Help
Selecciona el Fichero
:: AYUDA - Selección del Documento ::
root1/
Cancel


+5550000 - DefaultColorPhone
MIDlet View Help
Selecciona el Fichero
:: AYUDA - Selección del Documento ::
..
eojeda.pem
Factura Primera
photos/
Cancel

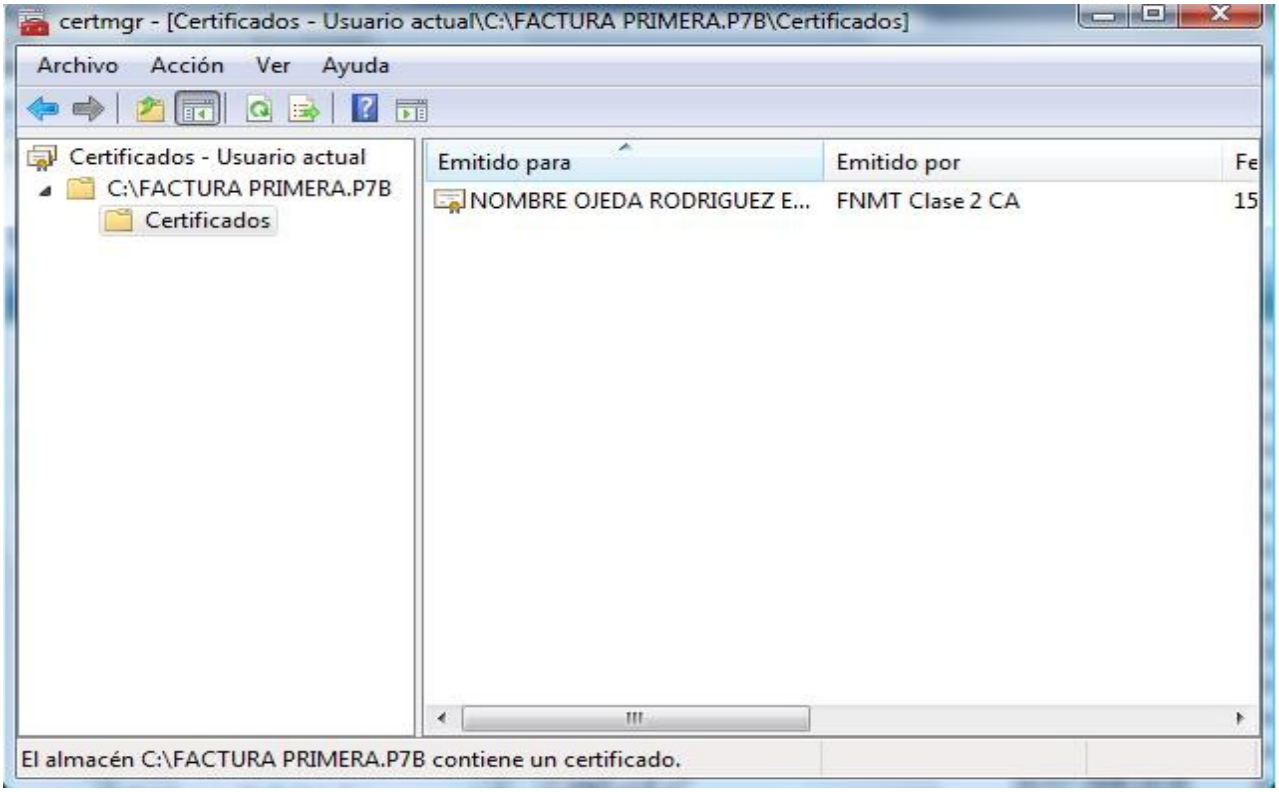

+5550000 - DefaultColorPhone
MIDlet View Help
ABC
procesando documento
A continuación se va a proceder a firmar y custodiar en @Firma el documento seleccionado.
Este proceso bloqueará su teléfono y puede tardar desde unos segundos a unos minutos, dependiendo de su capacidad.
Durante este proceso se accederá al servidor de @Firma. Esto implica conectar a internet el dispositivo móvil.
Pulse en el botón de continuar para comenzar el proceso. Si lo desea, también puede cancelar este proceso con el botón de cancelar que le devolverá al menú principal.
Cancel
Comenzar

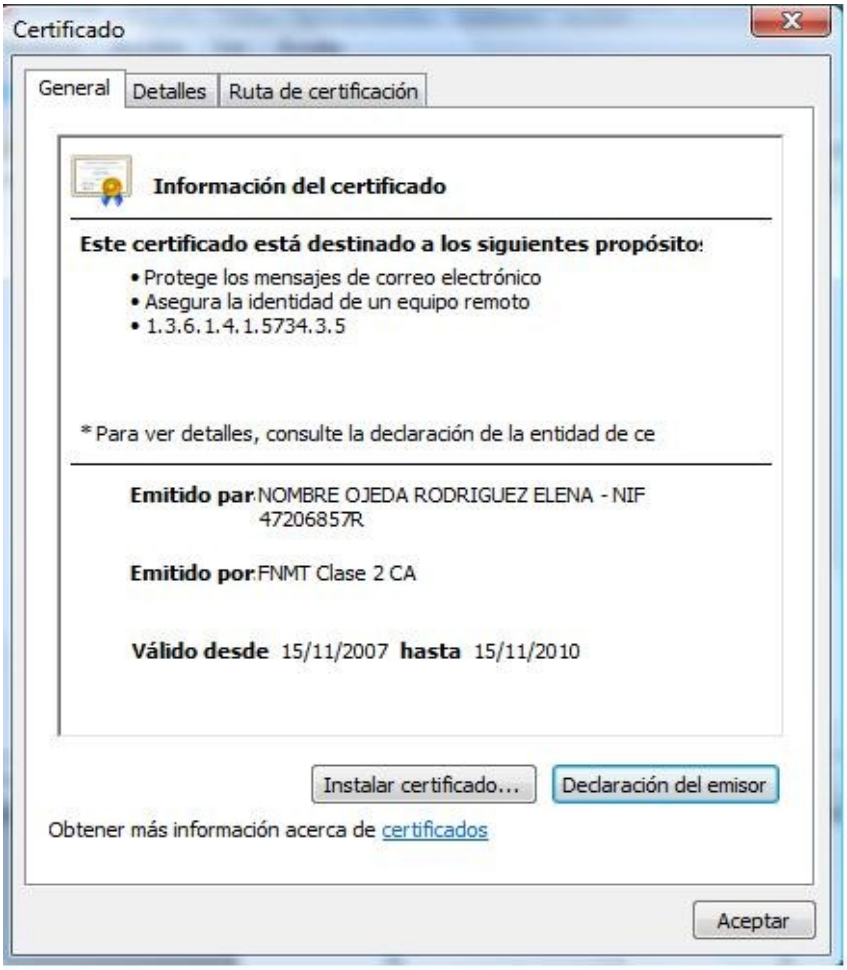

+5550000 - DefaultColorPhone
MIDlet View Help
ABC
procesando documento
Espere mientras se firma el documento y se custodia en @Firma... Este proceso accederá a internet para la custodia.
Cancel
Comenzar

El siguiente paso, será visualizar el fichero de firma .p7 devuelto al servidor WebDAV. Allí permanecerá, mientras el usuario propietario desee, para que pueda ser consultado cuando desee:

Título	Creado en	Modificado	Creador
Facturas_tienda	—	—	Administrator
Albaranprueba (132Kb)	2008-08-18 21:22	2008-08-18 21:22	Pepi
Factura Primera (18Kb)	2008-07-01 13:36	2008-07-01 13:36	Pepi
Factura Primera_firma.p7 (1Kb)	2008-08-27 18:26	2008-08-27 18:26	Pepi

Si descargamos el fichero .p7 y pasamos a ver su contenido, se mostrará lo siguiente:


certmgr - [Certificados - Usuario actual\C:\FACTURA PRIMERA.P7B\Certificados]
Archivo Acción Ver Ayuda
Certificados - Usuario actual
C:\FACTURA PRIMERA.P7B
Certificados
Emitido para
Emitido por
Fe
NOMBRE OJEDA RODRIGUEZ E...
FNMT Clase 2 CA
15
El almacén C:\FACTURA PRIMERA.P7B contiene un certificado.


Certificado
General
Detalles
Ruta de certificación
Información del certificado
Este certificado está destinado a los siguientes propósito:
• Protege los mensajes de correo electrónico
• Asegura la identidad de un equipo remoto
• 1.3.6.1.4.1.5734.3.5
* Para ver detalles, consulte la declaración de la entidad de ce
Emitido par: NOMBRE OJEDA RODRIGUEZ ELENA - NIF 47206857R
Emitido por: FNMT Clase 2 CA
Válido desde 15/11/2007 hasta 15/11/2010
Instalar certificado...
Declaración del emisor
Obtener más información acerca de certificados
Aceptar